Formal Modeling, Simulation, and Validation of Communication Platforms

vorgelegt von Diplom-Informatiker

Tony Modica

Von der
Fakultät IV — Elektrotechnik und Informatik
der Technischen Universität Berlin

zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
— Dr.-Ing. —

genehmigte Dissertation.

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Stefan Jähnichen	Technische Universität Berlin
Gutachter:	Prof. Dr. Hartmut Ehrig	Technische Universität Berlin
	Prof. Dr. Ina Schieferdecker	Technische Universität Berlin
	Prof. Dr. Julia Padberg	Hochschule für Angewandte Wissenschaften Hamburg

Tag der wissenschaftlichen Aussprache: 7. Juni 2012

Berlin 2012
D 83

Abstract

Human-centric communication-based systems, including social networks as Skype, Facebook, and Second Life, have been of increasing interest throughout the last years. Up to now, however, there are no adequate modeling techniques for this kind of communication platforms that would allow formal modeling, simulation, and validation of the user behavior.

In this thesis, we propose a suitable integration of algebraic high-level Petri nets and high-level replacement systems, based on well-known graph transformation techniques, in order to close this gap. On the theoretical side, this requires to develop a theory of algebraic high-level nets with individual tokens, such that the corresponding high-level replacement systems satisfy the requirements of $\mathcal{M}$-adhesive transformation systems and allow the platform designer to model and to analyze token-firing and rule-based modification within a single framework. On the conceptual side, we are able to show that typical features of communication platforms such as direct calls, multi-user chats, conferences, data multicasting, platform constraints and other requirements can be modeled and validated in this framework combining suitable modeling techniques. This is demonstrated for Skype, which is used as a running example and case study. Last but not least, it is shown how to develop a visual Eclipse-based tool for communication platforms, which supports modeling, simulation, and validation of typical scenarios.

Zusammenfassung

Kommunikationsbasierte Systeme, welche sich nach Menschen und deren Bedürfnissen orientieren, erlangten in den letzten Jahren immer größere Bedeutung und Interesse. Bislang gibt es allerdings keine angemessene umfassende Modellierungstechnik für diese Art von Kommunikationsplattformen, welche es erlauben würden, das Benutzerverhalten in derartigen Systemen formal zu modellieren, zu simulieren und zu validieren.

In dieser Dissertation stellen wir eine geeignete Integration von algebraischen high-level Petrinetzen und high-level Ersetzungssystemen vor, um diese Lücke zu schließen. Von der theoretischen Seite her erfordert dies, eine Theorie algebraischer high-level Netze zu entwickeln, so dass einerseits die entsprechenden high-level Ersetzungssysteme den Eigenschaften $\mathcal{M}$-adhesiver Transformationssysteme genügen und andererseits der Entwickler von Kommunikationsplattformen deren Schaltverhalten und regelbasierte Modifikation innerhalb eines Frameworks modellieren und analysieren kann. Von der konzeptuellen Seite her können wir nachweisen, dass typische Eigenschaften von Kommunikationsplattformen innerhalb dieses Frameworks unter Vewendung geeigneter Modellierungstechniken modelliert und validiert werden können, wie etwa Anrufe, Chats zwischen verschiedenen Benutzern, Konferenzen, Datenmulticasting, sowie Plattform-Constraints und andere. Dies demonstrieren wir anhand von Skype, welches uns durch die ganze Arbeit als Leitfaden, Beispiel und umfassende Fallstudie dient. Abschließend beschreiben wir die Entwicklung eines visuellen Werkzeugs für Kommunikationsplattformen in Eclipse, welches den Entwickler beim Modellieren, Simulieren und Validieren unterstützt.

Acknowledgments

I am most grateful to my supervisor Hartmut Ehrig for giving me an academic home base in his research group TFS (Theoretische Informatik/Formale Spezifikation) at the Technische Universität Berlin, for conducting my research journeys through the exciting fields of algebraic graph transformation, Petri nets, and category theory, and for always turning me in the right direction when I needed it — sometimes without even knowing it.

Furthermore, I thank the other referees Ina Schieferdecker and Julia Padberg for willingly attending to examining my thesis.

The opportunity to work on this thesis was kindly provided by the Integrated Graduate Program on Human-centric Communications (H-C3) at the Technische Universität Berlin by granting me a PhD scholarship for three years, by the DFG research project "Formal modeling and analysis of flexible processes in mobile ad-hoc networks" (forMAlNET) at the research group TFS, and by Fernando Orejas by inviting and hosting me twice at the Universitat Politècnica de Catalunya in Barcelona for four months. I deeply appreciate all this support.

With regard to the substance of this work, the most important resource were the members of my research group TFS as the habitat of the aforementioned theories and as the ideal nutrient medium for fructiferous ideas. In this sense, sincere thanks go to Kathrin Hoffmann for useful advice and many hints on how to assembly a PhD thesis and how to survive it, to Karsten Gabriel for interesting ideas and discussions in the theoretical abyss, and to Claudia Ermel for being always approachable, supportive and encouraging.

Likewise, I thank the rest of the TFS team — Frank Hermann, Ulrike Golas, Mascha Maximova, Olga Runge, and Leen Lambers — for their kind support, ideas, and feedback provided in numerous colloquia and discussions.

My mother Edith, Andreas and Vivien have my sincere affection for helping to keep up my morale outside of the ivory tower. In addition to this, I am devoted especially to Hanna for propping up and for helping to jump the last hurdles.

Finally, I owe Lisa Friedrich the beautiful realization of the idea for the cover picture.

Contents

1. Introduction

> Io sono vivo ma non vivo perchè respiro,
> mi sento vivo solo se sfilo la stilo e scrivo,
> quando sono nato non capivo
> ed ora che continuo a non capire
> non mi resta che...
>
> *(Caparezza)*

Today, in this advanced stage of the age of information and telecommunications, we are on the verge to a new level of maturity and pervasiveness of the technology founding this era. An indicator for this might be seen in the upcoming exhaustion of the approximately 4 billion IPv4 addresses, which constitute the Internet as a network of addressable systems by uniquely identifying all the devices connected to it. Apparently, the growth of the Internet and its structural demands were underestimated. After the migration to IPv6 addresses, much more devices will become eligible to be equipped with network interfaces in yet unanticipated ways. Presumably, the usage of isolated — i. e., noncommunicating — devices will decrease in many domains and the ability to communicate will become a more and more important feature and criterion for the users.

Within the initiative of the Human-Centric Communication Cluster (H-C3) at the Technical University of Berlin, the notion of Human-Centric Communication Spaces was coined to serve as an abstract view and ontology on general communication-based systems that essentially deal with needs of human communication and interaction. In this thesis, we focus on a special kind of Communication Spaces with additional characteristics that are typical for considered Internet-based systems, which we call Communication Platforms.

As to the importance of Communication Spaces and Platforms, any kind of support for the development of such systems is desirable that adequately takes into account the main characteristics of these systems. For specifying and modeling of communication systems, it can be advantageous to employ formal modeling techniques to be able to analyze and validate model properties related to security and privacy issues. There is a plethora of formal modeling techniques available that are adequate for particular aspects of Communication Spaces and Platforms. But, when we consider all their main aspects, it seems that there does not yet exist a single modeling technique that satisfactorily covers all of them.

Aims of this Thesis

The main goal of this thesis is to develop a formal modeling approach for human-centric communication-based systems that we subsume under the notion of Communication

Platforms. This modeling technique shall represent the behavior of the systems' users on a high abstract level so that we can abstract from the different concrete technical foundations of Communication Platforms. Still, the characteristical features such as the actors and typical behavior of Communication Platforms have to be respected by the modeling approach and should be easily identified in concrete models.

A suitable modeling technique for Communication Platforms should adequately cover the core aspects of Communication Spaces and Platforms, namely topology, content spaces, and interactions. There are excellent behavior-oriented modeling approaches such as Petri nets, which — besides some special extended approaches — assume that the structure of the nets is static, and there are reconfiguration-oriented approaches such as graph transformations, which are useful for modeling dynamic structures but which do not feature per se a behavioral semantics. On their own, these techniques cover the core aspects of Communication Spaces and Platforms only partially in a satisfactory way. As a main goal of this thesis, we try to develop an integrated modeling technique based on Petri nets, which can reconfigured by transformations similar to the well-known graph transformation approach.

The analysis of Skype as a concrete and typical existing Communication Platform can be taken as an example what a modeling methodology for Communication Platforms with the integrated modeling approach should be able to represent consistently. The results of this analysis should on the one hand lead to general modeling principles and ideas and on the other hand show open challenges for a concrete formalization of the integration of the single techniques.

These challenges have to be met by a suitable formal definition of the integrated modeling technique and a development of a notion of Communication Platform models based on the integrated modeling technique. It is essential to combine the behavioral semantics of the integrated modeling techniques to an appropriate formal semantics for the whole concept of Communication Platform models in order to simulate the user behavior in modeled platforms.

For validating properties of Communication Platforms, first of all common analysis results for the single techniques might still be applied to analyze parts of a Communication Platform model. But, we can expect that the integrated approach is more complex and that the modeling principles make use of the single concepts intertwinedly. Hence, results should be extended or transfered to enable us to appropriately formulate and validate the properties found in the initial analysis.

A thorough case study on modeling the concrete Communication Platform Skype completes and justifies the principles, choices, and definitions. This includes a further reconsideration of the modeling requirements for the Skype platform to be modeled w. r. t. specialties of the new formal approach of Communication Platform models.

It is highly important to develop a software tool that supports the modeler in the design and testing of Communication Platform models. This tool should allow the modeler not only to edit models but also to simulate them and to perform validation checks.

Main Results

The main results in this thesis for the given goals are summarized as follows.

Communication Spaces and Platforms An examination of the abstract concept of Human-Centric Communication Spaces (HCCS) coined in the H-C3 initiative yields the core modeling aspects of content and contextuality, topology, and interaction. The notion of Communication Platforms on which we focus in this thesis is a specialization of this abstract concept by additional assumptions such that it characterizes important concrete communication-based systems such as Skype, Second Life, and Facebook. A comparison of different modeling techniques and how they cover the main modeling aspects of HCCS leads to the first choice of algebraic data, Petri nets, and Petri net transformation based on algebraic graph transformation, all to be combined to an approach of reconfigurable high-level Petri systems.

Further analysis leads to concrete modeling principles for modeling Communication Platforms with such an approach. We find that existing approaches of reconfigurable high-level Petri systems are not expressive enough to support the modeling principles, especially regarding the modeling of data distribution in the platform and the necessary control of reconfigurations. This challenges us to define a suitable class of reconfigurable high-level Petri nets together with extensions available from the domain of algebraic graph transformations.

Transformation of Petri Nets with Individual Tokens In existing approaches of reconfigurable Petri systems, rules for changing markings by reconfiguration can only be formulated with some restrictions, which is inconvenient in terms of usability and intuitiveness of the transformation approach. Marking-changing rules are essential for modeling communication systems and platforms, especially for realizing multicasting of data. As a solution for this, we define P/T nets and AHL nets with markings of individual tokens (in contrast to “collective tokens”), which we call PTI and AHLI nets. For rule-based transformations following the double pushout approach, we get the important result that firing steps correspond to the application of special firing rules. Other main results are the instantiation of the framework of $\mathcal{M}$-adhesive transformation systems for reconfigurable PTI and AHLI nets and that the individual net classes are syntactically and semantically compatible to each other and their collective counterparts, which we show by suitable functors.

Communication Platform Models Based on the principles from the initial analysis, we develop modeling concepts for Communication Platforms as reconfigurable AHLI nets, regarding the representation of actions and their owners, modeling the perception of data by users, and relating reconfigurations to firing steps.

For the last point, we extend the AHLI transformation systems by advanced concepts from algebraic graph transformation. First, for controlling of reconfiguring rule applications, we propose to use the approach of nested application conditions for $\mathcal{M}$-adhesive transformation systems. Due to technical reasons, it not possible to formulate nested application conditions for the kind of rule that we use to model multicasting and other transmission of data. Therefore, we adapt the definitions of nested application conditions and satisfiability to variable application conditions and structural satisfiability and prove the conceptual compatibility of structural and normal satisfiability.

Secondly, we need the concept of interaction schemes and amalgamated rules for $\mathcal{M}$-adhesive transformation systems to be able to model reconfigurations such as multicasting. The original definition of amalgamation for rules allows for nested application conditions. Therefore, we loosen the definitions of interaction schemes and application of amalgamated rules to support our new variable application conditions and show that the result of the loosely amalgamated rules still are well-defined and have the desired effect.

Finally, we define token request handlers as a control structure that mediates between the firing behavior in an AHLI net and the rules to be applied for reconfiguration.

All these concepts are integrated to a formal definition of Communication Platform models and the user actions that can occur in it. A construction of a special controlling AHLI net over the platform model — a *higher-order* net that contains the AHLI net representing the platform configuration and the transformation rules as tokens — defines the operational behavior of the platform model such that all firing steps in the control AHLI net — and only these — are valid user actions or system reconfigurations in the platform model.

Validation In order to validate the properties we found in the initial analysis of the notion of Communication Platforms, we develop two formal techniques. The first is based on our variable application conditions, which are extended to platform constraints that can be checked against a concrete AHLI net. This method can be used to validate structural properties, e. g., regarding the uniqueness of identities in platform models or the success of user actions.

The second is based on the notion of independent transformation rules, which is extended to independence of firing steps and rule applications, for which we formulate a local Church-Rosser theorem. We use this result for defining parallel and sequential independence of user actions — i. e., firing steps with following reconfigurations — and show a compatibility result that allows us to validate in a platform whether two user actions are independent. Furthermore, these notions of independence contain staged conditions, which can help to locate the errors in defective models.

Tool Support There are several available software tools for algebraic graph transformation and reconfigurable Petri nets, which we extend and combine to an visual Eclipse tool for supporting the modeling of Communication Platform models. In this tool, the platform models can be edited according to the definitions and principles for Communication Platform models. By employing the graph transformation engine of the AGG tool, on the one hand user actions and reconfigurations can be simulated, and on the other hand also platform constraints can be formulated and checked. Further, in a configuration it is possible to check the independence of two possible user actions and to get a report on the details why they possibly are not independent.

Overview of the Thesis

Chapter 2 (Human-Centric Communication Spaces and Platforms) In this chapter, we discuss the special need to improve communication systems under the particular human-centric viewpoint and we review related work that has led to the notion of Human-Centric Communication Spaces. A special kind of systems in this section, namely Communication Platforms, suggest themselves to be considered under the aspects of Communication Spaces. We give a characterization of such systems and lay out some of the possible challenges posed by this general notion. With this in mind, we discuss several modeling techniques as options for a formal approach of Human-Centric Communication Spaces and narrow them down to some essential elements.

Chapter 3 (Modeling Requirements of Communication Platforms and Skype) As a concrete instance and representative for Communication Platforms that provides the most common and important features of such systems, we present Skype in detail. In the end, this yields modeling principles and particular challenges for a realization with the technique of reconfigurable Petri nets to be solved in the following chapters.

Chapter 4 (Transformation of Petri Nets with Individual Tokens) As a basis for models of communication platforms, we present a new Petri net formalism based on P/T nets and AHL nets that we call "Petri nets with individual tokens", together with a rule-based transformation approach that is based on the graph transformation approach with double pushouts. We further give $\mathcal{M}$-adhesive system for Petri nets with individual tokens. After discussing equivalence of firing behavior for nets with individual tokens, we describe functorial relationships between the individual net classes and their collective counterparts. Finally, we recapitulate the modeling approach of algebraic higher-order nets as a possibility to control firing steps and reconfigurations.

Chapter 5 (Transformation Systems for Modeling Communication Platforms) We further develop and concretize the modeling principles from Sect. 3.2.6 for the modeling approach of reconfigurable AHL nets with individual tokens. We tailor this approach to Communication Platform models by extending it with the advanced transformation concepts of amalgamated transformations, variable application conditions, and token request handlers in order to deal with the principal challenges of reconfigurable Petri nets that we gathered in Sect. 3.2.7. Moreover, we present higher-order nets with a specially suited algebra as a control structure to combine firing steps and reconfigurations to user actions of a Communication Platform model.

Chapter 6 (Validation Techniques for Communication Platform Models) For expressing and checking conditions for the integrity of a Communication Platform model, we present platform constraints in Sect. 6.1, which are similar to the technique of (variable) application conditions. Afterwards, we discuss functional behavior of user actions in Communication Platform models, which leads to a formal functionality condition for token request handlers. Finally, we examine the independence of behavior in a Communication Platform model — i. e., transition firings and possibly triggered reconfigurations — and give conditions for checking the (in)dependency of two given user actions.

Chapter 7 (Modeling Case Study: Skype) We apply the techniques and results of the previous chapters on a case study on modeling the Communication Platform Skype. For this, we first examine the behavior of the Skype clients in more detail and with the modeling concepts in mind for delimiting the extent of the case study. After defining a Communication Platform model for Skype, we demonstrate the system specification and validation techniques at concrete use case scenarios.

Chapter 8 (Tool Support) In this chapter, we describe the development of a prototypical tool environment called the CPEditor, which supports editing, simulation, and validation of Communication Platform models. The CPEditor is currently under development. We give an outlook on the complete set of required features that it will support, together with a status of their implementations.

Chapter 9 (Related Work) This chapter summarizes the related work that we discuss in detail in diverse parts of this thesis.

Chapter 10 (Conclusion) The final chapter gives an overview over the conceptual and technical main results of this thesis and concludes with an outlook on future work.

2. Human-Centric Communication Spaces and Platforms

> Ihr habt die Natur geschmäht, daß sie Sinne eingesetzt und den Menschen nicht nach dem Muster eurer Abstraktionen geschaffen. [...] Ihr bedachtet nicht, daß in Sachen der Wissenschaft, der Religion und der Kunst so wenig als in weltlichen Geschäften je ohne überwiegende Naturkraft etwas Großes vollbracht worden und daß die erhabensten Äußerungen der Seele ohne eine kräftige Sinnlichkeit tot und unwirksam für die Welt sind.
>
> *(Friedrich Wilhelm Schelling)*

In this chapter, we discuss the special need to improve communication systems under the particular human-centric viewpoint and we review related work that has led to the notion of Human-Centric Communication Spaces. We give a characterization of such systems and lay out some of the possible challenges posed by this general notion. With this in mind, we finally discuss several modeling techniques as options for a formal approach of Human-Centric Communication Spaces and narrow them down to some essential elements as a bridge to the next chapter.

Because the notion of Communication Spaces is quite (but intentionally) diffuse and is intended to cover too many kinds of systems (which we do not want to restrict) to grasp it at once, for approaching a formal modeling we concentrate on a particular kind of concrete computer-related communication systems in the work at hand. In these systems, which we subsume under the notion of Communication Platforms, we identify concrete requirements for appropriate formal description of behavior, representation of elements in Communication Spaces, and interesting properties to validate.

2.1. A New Concept for New Needs

Modularization of systems has been used for a long time to avoid monolithic and overly involved systems. The younger concept of connecting systems and their components via the Internet (or a network in general) allows the system designers to consider flexible configurations of connections for the specification of dynamic and long-running systems. Note that (probably in contrast to traditional system design) we entitle systems as *dynamic* if they may not only change their state or data but also their structure. In software engineering, web services are used for this purpose instead of the traditional libraries and modules bound at compile-time. In hardware engineering, the same holds for network interfaces versus fixed wiring. Where communicating systems once were closed w. r. t. the components at the time of development, today's communicating systems may be open to be coupled loosely at runtime. The popularity and the inflationary usage of the term *service-oriented architecture* (SOA), which denotes a paradigm based on the loose coupling of services, proves this trend. There are also similar approaches for hardware, e. g., *mobile ad-hoc networks* (MANETS), which are networks of mobile devices with wireless (possibly unstable) access that configure their connections in a manner to keep up the network's connectivity as best as possible. Although the support for formal specification and verification may be a bit behind, building such dynamic systems is the state of the art in system development. After all, as long as the details of the network interaction and service handling are not essential either for the system's handling nor its overall goal, we can abstract from these details and concentrate on the specification of the components' interfaces (e. g., by using the standardized Web Services Description Language (WSDL)) just as in traditional software development.

We get another point of view on systems if we assume that dynamics and communication are not used just in order to achieve some goal, but if they are the main feature and purpose, or short, if communication is not a tool but the service itself. The existence of such systems becomes apparent if we think of humans as the users of a system that is meant to support some sort of communication between them. There are some typical distinguishing characteristics of such systems. First, by and by the user domain changes as new users are introduced who want or need to use the service or current users leave the system. Secondly, data is considered to belong to a user of the system and data is more important to be transmitted at all and — in some sense — correctly than to care about some data as result of distributed computation and its accumulation. Some of the most important representatives of these human-related communication systems with the mentioned properties are systems for e-mails, bank transfers, chats, and social networks in general.

Traditional system and software engineering focuses on communication between components as a tool or medium for improving the realization of business processes. There is clearly a need for a concept of the special kind of systems that are just meant to connect people in order to aim at improving the medium itself especially w. r. t. the human needs. We discuss such a concept in the next section.

Towards Human-Centric Communication Spaces

We already mentioned for the systems of our interest the property of being "human-related". But this notion is still too weak to serve as the foundation of a new concept because it just presumes some relation to the human users. Instead, "human-centric" is a better choice that stresses the intention of focusing on the human users and that provides a clear separation from the "technic-centric" systems. Of course, the term "human-centric" is not new, it rather already has become a catchy marketing phrase in software and system development, which is widely used in an intuitive and implicit way, often to point out easy-to-use user interfaces. A prominent example for human-centric engineering is the Project Oxygen [Rud01] at the Massachusetts Institute of Technology (MIT). But this project's view on human-centricity still is focused on technology to achieve certain goals that are meant to improve mainly the handling of the developed devices. In the next section, we discuss a more general view on human-centricity by abstracting from any technology to get a more high-level viewpoint.

While discussing the appropriateness of terms for naming a new concept, we may look for an alternative for the term “system”. According to Merriam-Webster, we can define a system as

> a group of devices or artificial objects or an organization forming a network especially for distributing something or serving a common purpose, such as a telephone system, a heating system, a highway system, a computer system.

This means if we come to modeling systems, we have to model for a common purpose. But what is the common purpose of, e. g., a social network[1] other than to offer its users the possibility to communicate in some way?[2] Clearly, there is no common purpose as in non–human-related systems such as most control computer systems, which are supposed to be used in a context that provides a service as a whole or that yields a final product. Talking of human-centric communications, we come to the conclusion that a more general term for a structure not implying a common purpose is more adequate as long at it still allows for the relations of its parts. Without doubt, in human-centric communications the relations between the users play a central role. A general term with this property is the ordinary “space”.

Hence, in the following we discuss the concept of Human-Centric Communication Spaces rather than of human-related communication systems.

2.2. History and Related Work

In this section, we take a look at some previous related work, in which the terms from the fields of human-centricity and communications that we use have been coined. We refer to and discuss them especially from the viewpoint of the Human-Centric Communication Cluster (H-C3) of Sect. 2.2.4.

2.2.1. The Reference Model of Open Distributed Processing (RM-ODP)

The Reference Model of Open Distributed Processing (RM-ODP) has been adopted as a standard for the specification of distributed systems by the International Organization for Standardization (ISO). The RM-ODP is focused on system specifications expressed in terms of “objects” and many important concepts and notions for the field of communicating systems have been defined in this standard in an abstract but universal way. As this standard has existed for almost 15 years, it should be used as a mere reference because today many of these notions seem to have become ubiquitous and self-evident in the domain of system modeling, possibly because of the wide usage of UML as a particular notation for the RM-ODP concepts [IOS08].

[1] As a special kind of common interaction platform, which characterizes virtual communities [SS01] in addition to common interest and norms.

[2] We do not follow the conspiratorial idea that those networks are meant to pile personal data for marketing and advertising purposes, only.

We recall some of the most important and general notions from the first two parts “Overview” [IOS98] and “Foundations” [IOS96] of the RM-ODP as a common ground for later use.

Object: A system is composed of interacting objects that represent entities in the real world. An object is distinct from any other object, and it is characterized by its behavior and, dually, by its state. An object is encapsulated, i. e., any change in its state can only occur as a result of an internal action or as a result of an interaction with its environment. An object is distinct from any other object, and it is characterized by its behavior and, dually, by its state. An object is encapsulated, i. e., any change in its state can only occur as a result of an internal action or as a result of an interaction with its environment.

Action: Something which happens. Every action of interest for modeling purposes is associated with at least one object. The set of actions associated with an object is partitioned into internal actions and interactions. An internal action always takes place without the participation of the environment of the object. An interaction takes place with the participation of the environment of the object.

Behavior (of an object): A collection of actions with a set of constraints on when they may occur.

State (of an object): At a given instant in time, the condition of an object that determines the set of all sequences of actions (or traces) in which the object can participate.

State and behavior are interrelated concepts. The state of an object is the condition of the object at a given instant that determines the potential future sequences of actions that object may be involved in. At the same time, actions bring about state changes and, hence, the current state of an object is partly determined by its past behavior. Of course, the actions an object will actually undertake are not entirely determined by the object’s present state; they also depend on which actions the environment is prepared to participate in.

Communication: The conveyance of information between two or more objects as a result of one or more interactions, possibly involving some intermediate objects.

Role: A formal placeholder in the specification of a composite object. It identifies those aspects of the behavior of some component object required for it to form part of the composite and links them as constraints on an actual object in an instance of the composite. In order to satisfy the specification, the actual object is required to exhibit the specified behavior. It is then said to fulfill the role in the instance of the composite.

A role may correspond to a subset of the total behavior of a component object. When an object is viewed in terms of a role, only a named subset of its actions is of interest, and other actions are abstracted away, possibly to other roles. A component object may have several roles at a given time depending upon its interactions, and may take different roles at different times. These roles may be associated with interfaces.

Contract: An agreement governing part of the collective behavior of a set of objects. A contract specifies obligations, permissions and prohibitions for the objects involved. The set of actions associated with an object is partitioned into internal actions and interaction.

2.2.2. Spaces

The notion of space has two main aspects that are important for our purposes.

Generally, space is thought of as being something which is enclosed by borders giving elements the possibility to be inside or also to belong to it. Sometimes an "outside" is not even perceivable as is for the space in the sense of our universe. Membership to this kind of space is merely equivalent to existence. In the foundational theories of computer science, such as set theory and category theory, the concept of space is present in the notions of set, class and universe. A space also supports the idea of identity because intuitively we agree that no two different things can be at the same place. Following this, Leibniz understands spaces as domains of order[3].

Moreover, in Leibniz' relationistic conception, spaces are constituted by relations between their objects. A space therefore has a structure, it is relative to perspectives, and it is being observed as a system of relationships that actually or potentially exist.

In total, spaces are the most basic concept for distribution and linkage of objects, or — in more technically terms — for scopes and bindings.

2.2.3. Individual Communication Spaces and I-centric Communication

[AMSPZ01, Arb03, Ste03] developed the notions of *Individual Communication Spaces* and *I-centric Communications*, which later have served as basic terminology in the Wireless World Research Forum [WWR03]. The theses propose reference models for I-centric communications and user interactions, respectively, in which communication is generally modeled as acting in Individual Communication Spaces but being independent from specific technology. They motivate the notion of Individual Communication Spaces as follows:

> "The vision of I-centric communication have been developed, which put the individual ("I") user in the center of all activities a communication system has to perform. [...] technology has eliminated distances in time and space or at least made these boundaries almost imperceivable. By this means, today's communication services act as a prolongation of human senses and extend the individual communication space. Following this view, a new approach is to build communication systems not based on specific technologies, but

[3] "Der Raum ist die Ordnung gleichzeitig existierender Dinge, wie die Zeit die Ordnung des Aufeinanderfolgenden."

> on the analysis of the individual communication space. The result is a communication system that adapts to the specific demands of each individual (I-centric)."

Figure 2.1.: Individual Communication Space (from [Arb03])

From the viewpoint of I-centric communication, each individual has its unique communication space, which contains all objects an individual might want to interact with. Objects generally represent individuals and directly addressable hardware–software entities performing services. The population of a communication space may change as objects enter or leave it. The use of objects depends on the actual context and the acting individual. Contexts define relationships between individuals and objects of their communication space.

Other central ideas related to general human-centricity described in these documents are

- to respect a human's demands such as featuring appropriate interfaces for specific mental and physical conditions, contextual interpretation of actions, and semantical definition of goals (e. g., "Invite X to my party" being executed according by automatic selection of existing and possible communication means such as e-mail, telephone etc.);
- to reflect recent actions to enable profiling;
- to provide self-adaption to environment (ambient-awareness) and preferences (personalization).

All in all, the I-centric communication spaces together with the idea of human(-centric) demands was a promising and important step towards a general concept for systems as we characterized them in the beginning of this chapter. Although, there are some details we want to abstract even more or, at least, we do not want to fix them as much. For example, in [Arb03, Ste03] a context is defined as

> "a certain 'universe of discourse'. It defines relationships and causalities of an individual to and between particular numbers of objects of its communication space."

and is then treated as a kind of architectural feature to realize activation of communication facilities; in our communication spaces we want to consider contexts just as a communicational commonality (shared by communicating objects) that is (possibly) based on knowledge and is close to the linguistic term *common ground*.

But for us it is even more important to abstract from I-centricity in order to be able to apply the idea of communication spaces to the aforementioned kind of systems with arbitrary sets of human members, rather than to limit it to single individuals. To still emphasize the attention to human needs, we generalize the terms I-centric and Individual Communication Space to Human-Centric Communication Space. With this, we are following the idea of communication spaces as formalized in [PSH06], where a communication space is defined over a set of agents that represents a logical or physical space for communication. Although, the authors have in mind multi-agent systems (MAS) without any relation to human-centricity and in contrast to our intention, the main purpose of MAS is distributed calculation and problem solving.

2.2.4. The Innovation Center Human-Centric Communication (H-C3)

The Innovation Center Human-Centric Communication[4] has been founded as a follow-up to the Human-Centric Communication Cluster (H-C3) at the Technical University of Berlin. Both hosted a graduate program that supported the development of this thesis.[5]. The original proposal for this cluster formulated the motivation for the research and development of communication spaces and hence is a fundamental source of the work at hand. The accompanying presentation slides for this proposal laid out drafts of most of the topics of this remaining section. Because nor this proposal nor the mentioned slides have been published, we summarize their most important parts in the following.

The proposal's principal vision is

> "Human-Centric Communication offering flexible and intuitive support in accessing situation dependent information as well as in exchanging information with others."

for which the work of the associated members has been organized into six research areas. The characterization of Human-Centric Communication Spaces in Sect. 2.3 has been formulated as a central notion and object of concern of H-C3, and it is treated in particular by the H-C3 focus area "Modeling of Communication Spaces", whose main objective is

> "to develop modeling techniques for the new notion of computer-supported communication spaces (CSCS) as introduced in the scientific vision of H-C3.

[4] http://www.h-c3.org/index_en.html

[5] http://www.h-c3.org/IGP/index_en.html

> This new concept should serve as a unifying concept of human-centric communication within H-C3 on an informal level including especially the notions of context, content and conception and related system requirements. Furthermore, formal modeling techniques for CSCS should be developed which can be applied to model and compare communication concepts in various applications areas. Up to now there is no well-defined and well-established notion of communication space which combines the concepts of communicator, community, and communication on the one hand and those of context, content and conception on the other hand and their computer support. Hence, we will develop a theoretical basis and the modeling techniques for CSCS."

Relying on the perception of Human-Centric Communication Spaces in the environment of H-C3, this thesis is supposed to contribute to this endeavor.

2.3. Human-Centric Communication Spaces

> Collecting data is only the first step toward wisdom, but sharing data is the first step toward community.
>
> *(Henry Louis Gates Jr.)*

One might say that the general idea of Human-Centric Communication Spaces (HCCS) is quite clear, but so far no concrete application has been formulated or even what particular aspects and advantages this would yield. In this section, we give an abstract definition and characterization of what a HCCS is, relying on the one hand on classical terms in distributed systems (Sect. 2.2.1) and on the one hand on the perception of the notion of Communication Spaces stemming from H-C3 (Sect. 2.2.4). For this definition, we first look back at the main concepts used in H-C3. These basic "c" concepts are related as follows:

> "Communication, i. e., the search for, and exchange of, *content* expressed in terms of *conceptions* takes place as interactions between communicators (persons or machines) in reflecting their common *context*. Communication interactions are structured using communication services being personalized based on participant's characters and made available using *communication infrastructure*."[6]

The far goals for a comprehensive framework of HCCS that follow from this description are threefold:

- It should give an answer on how to provide sufficient context to avoid misinterpretation especially between human actors, i. e., to serve as a semantical framework for communication of contents taking into account different interpretations by different actors.

[6] Further "c" concepts are *communities*, *communication potential*, and *computer-supported communication spaces*.

- It should conceptualize communication means and behavior to integrate the various aspects into a common framework for communication-based systems in different application domains including hardware, software and community aspects.
- As a conceptual framework, it should provide concepts to handle communication across different separated and overlapping communication spaces and between actors acting simultaneously in different roles of communication, and especially to provide means for adaption of communication capabilities to communication interests.

These are the main questions that the following characterization of HCCS aims at. In the rest of this section, we concentrate on and characterize the notion of HCCS in general as a foundation to discuss its main aspects and (in the next section) how modeling can be supported by (possibly established) formal techniques.

2.3.1. The General Notion of Human-Centric Communication Space

As an abstract concept, we regard a Human-Centric Communication Space as

> “a common context (real or virtual) in which community members can communicate, i. e., exchange content or search for content. The set of communicators may change dynamically. People can be simultaneously in several communication spaces.”

The viewpoint we assume when regarding a concrete system as a Human-Centric Communication Space is the following:

Extensional: Each HCCS has a *set of actors*, i. e., humans, machines, communities, each of them having its personal *content space*. Several *roles* and *action rights* (such as send, receive, store, retrieve, transform) are assigned to the actors. Further, a HCCS has a *set of communication channels*, i. e., communication media and facilities, that allow the actors to send and receive messages from one another. These channels consist of data and context information and should be according to content spaces and access rights of both linked actors. An important aspect of HCCS is their dynamics, as channels and actors as well as their content spaces, roles and rights may be added/created and deleted/updated dynamically (cf. Fig. 2.2).

Intensional: A (formal definition of a) HCCS should be characterized by a set of prescriptions such as rules, contracts, conventions. These justify the membership of actors and their assigned communication roles, guard the communication of data and contexts, and establish borders with the environment of the space.

Separation of concerns As starting points towards a realization of the aforementioned goals, we separate and summarize three main aspects of HCCS in which this notion shows special interest:

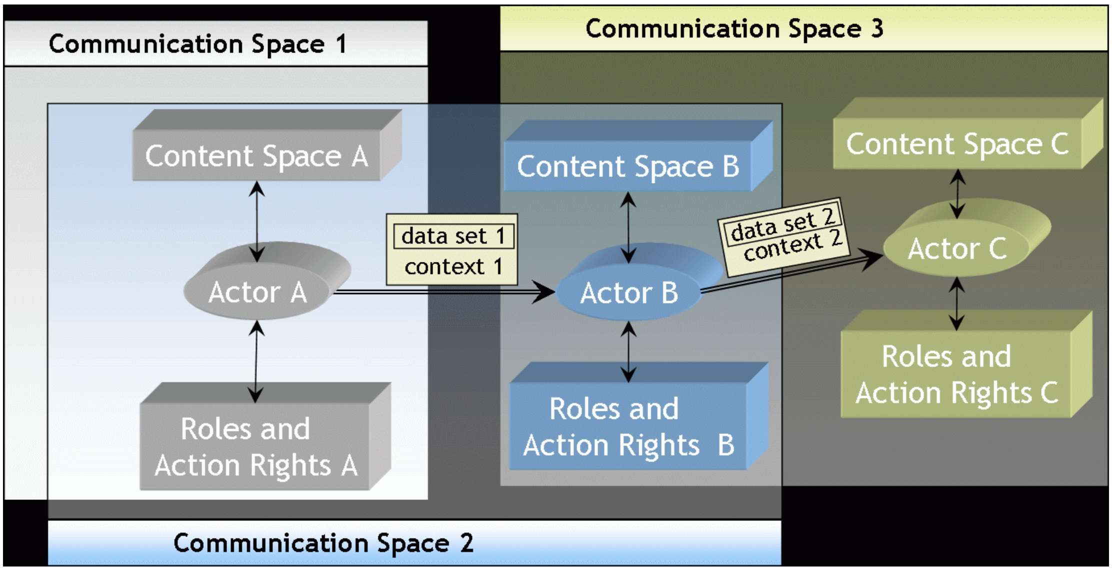


Figure 2.2.: Interaction in and between Communication Spaces

Content and contextuality: Human-centric communication requires data processing and communication based on content and interpretation and that change of context implies change of semantics. Content is generally the result of interpretation of data (in an environment of action and judgment). So HCCS should be able to handle communication of contents, especially to provide a possibility to model generating, selecting, presentation, and delivering.

As a special case of this, we have interaction between Communication Spaces: We allow actors to participate in several communication spaces, so the spaces' actor sets may overlap. Preferably, the actors should be able to transform and update content when communicating in different communication spaces, e. g., to transfer data between them. For this we may use model and content transformation techniques but also should think about semantical correctness and how to achieve this.

Dynamic structure: The concrete system has to be dynamic in a structural sense, so that actors can possibly move in communication spaces, join several communication spaces, and leave them again. Note that also roles and action rights are treated dynamically.

Communication-specific features: Defined access rights and roles are to be respected, as well as human-centric demands such as preferences and adaption to the current environment. This implies that behavioral descriptions can be subject to dynamic change due to environmental changes.

Note that we should think of “Human-Centric Communication Space” as a general and diffuse[7] concept. This may lead to different ideas about details of HCCS, which are very welcome, because with this notion, we intend to provide a viewpoint on arbitrary systems that deal with communication where human beings are involved. Based on a given infrastructure, a Communication Space may be constituted in different ways: by certain means of media, facilities and technology, by certain kinds of data, by goals of interpretation and judgment, by regulations, or by the intentions of its actors. In each case of a HCCS, the bindings of content, actors and technology, as well as the scopes of communication policies have to be specified in the terms of the concrete situation at hand. This should allow to instantiate the abstract notion of HCCS to different systems and to construct transformations between different instantiations.

Goals for a General Framework of Communication Spaces

Of course, it is an ambitious goal to develop a universal unifying concept and framework of HCCS that takes into account human-centric factors and that is applicable to very different concrete systems as instantiations of it. Just in order to start working towards this goal, we have to identify some subgoals dividing the intimidatingly big field of possible studies and applications into feasible parts and research steps. Some promising directions into the realms of HCCS are the following:

Towards a conceptual model: We need a comprehensive model of HCCS to formulate the requirements and the design of communication-based systems in different applications domains. This model should emerge from selected reference models for corresponding languages, architectures, and technologies. Future research may treat integration and extension of current techniques such as various data formats and algorithms, Web Services, and Model-Driven Architecture (MDA) into one formal model, which is parameterized by concrete communication context and correctness calculi for such a model. This may allow to build tools for constructing communication systems.

Towards a mathematical model: Such a model would possibly provide us formal advantages such as notions of and analysis of concurrent operational semantics, behavioral properties (liveness, safety, security), efficiency of communication-based systems, verification of formal requirements. Promising starting points for this are Categorical Reaction Systems (Bigraphs), a unified approach by Milner for different process calculi; and Algebraic Graph Transformation, which is an unified approach for different visual modeling techniques and model transformations. These may be integrated and extension by content-modeling with semantic web and knowledge engineering techniques, and verification of requirements based on temporal logic for graphical constraints.

Towards system architectures: As a basis for technical realization, appropriate architectural concepts are needed to handle trustworthy communication of contents

[7] In the “good” sense of this word, meaning “everywhere or throughout everything, not concentrated or restrained” rather than “being at once verbose and ill-organized”.

between actors in different communication environments and application domains taking into account human-centric factors. To some extent, this has already been approached by [Arb03, Ste03] (see Sect. 2.2.3).

2.4. Towards a Formalization of Human-Centric Communication Spaces

> Some people are always critical of vague statements. I tend rather to be critical of precise statements; they are the only ones which can correctly be labeled 'wrong'.
>
> *(Raymond Smullyan)*

Communication systems — and spaces all the more — are not technologies in the classical meaning but just rely on several technologies for their realization as software and electrical engineering and signaling technique. The development of tools for software engineering, compilers, and production techniques for electric circuits has advanced to a certain degree; after all, there are many systems in use that we can identify as communication systems and spaces. But most of this progress has been achieved on the general level of fundamental research instead of specification and design for particular applications. This is only normal as such technologies are mainly the offspring of scientific research; but the manifestations of a technology (i. e., concrete products) and their properties and commonalities are studied afterward when the products have matured from "proof of concept" over "bleeding edge" to "state of the art". Only then, the kind of products itself becomes of interest for research and the development of general formal methods. Such formal methods are the basis for the step from mere feasibility to better (robust, reliable) and more convenient (inexpensive) products if they allow us to prove the soundness of system specifications and assertions over system properties and to use formal semantics as the basis for generating correct implementations as far as possible. Perhaps, elaborated techniques and methods with their results can be reused or extended, perhaps so that with a narrowing and specialization of their application area even more or more appropriate results can be achieved due to additional assumptions.[8]

Besides the conceptual clarification of the notion of HCCS, it is therefore desirable to have a formal modeling technique for HCCS, so that we can specify the features of such systems in an appropriate way and are able to simulate, test, and analyze/validate them by using not only the structure but also the formal semantics of the modeling technique. General classic properties of interest for analysis of behavioral systems are related to consistency, safety and security requirements, liveness, termination etc.

[8] Just to provide an example: With the appearance of programmable computers, besides the classic worst-case analysis of an algorithm's complexity in terms of Turing-decidability and feasibility also the average-case complexity became of (maybe even grater) importance for concrete systems.

2.4.1. Focus of Modeling in this Work

Of course, it is a long way from the idea of HCCS to a comprehensive formal toolbox or even framework allowing for human-centric features. Due to the wide-spread and diffuse nature of the notion of HCCS, we surely have to choose and formulate specific points of action to focus on modeling problems in a manageable extent.

For the work at hand as a first endeavor towards HCCS, we agree on the viewpoint of modeling high-level user–system behavior and communication features in order to better approach the general concept of HCCS (cf. Sect. 2.3) and how to represent its most important notions, e. g., actors, channels, and interactions. Data transfer and synchronization between different HCCS respecting different contexts are too advanced topics for this first stage of modeling and rely heavily on a basic understanding of HCCS themselves, which are the reasons why we concentrate on modeling single HCCS for now.

The common Open Systems Interconnection (OSI) model [IOS94], which describes abstraction layers as a way to focus on the granularity of communication systems on different levels, suggests the topmost "Application Layer (7)" as such systems' most abstract interface to the user. In our viewpoint, we identify the "application" as a device or means allowing some human to use the system considered as the HCCS. In terms of the RM-ODP (see Sect. 2.2.1), this is primarily related to the "Enterprise Viewpoint", described in [IOS96] as "A viewpoint on the system and its environment that focuses on the purpose, scope and policies for the system.", because we want to describe the features that a HCCS offers to its users. With this choice, we show interest in the particular question *how* to describe and represent the users and their possible (inter-)actions as essential parts of HCCS rather than to realize the exchange of data on low-level (physical) levels.

We concretize this point in Sect. 3.2.2 with properties of a representative example in mind.

2.4.2. Formal Techniques for Aspects of Human-Centric Communication Spaces

We recall the important aspects of HCCS (see Sect. 2.3) that have to be supported by formal techniques in consideration for modeling here in more technical terms pointing out to areas treated by well-established formal modeling techniques. There are also crossbreeds and extensions of these techniques that integrate different aspects or that are specialized to support particular modeling needs. We discuss some promising variants of these extensions and related work towards modeling of communication systems to find appropriate candidates. Hybrid techniques may cover more than one aspect; we categorize them under the aspect that fits their ancestors' primary intention.

Content and Contextuality In "Human-Centric Communication", machine action is related to humans and their mental and physical conditions. This, among other things, requires processing of data to be based on content and interpretation with regard to contextual constraints and semantically defined goals. Content is the result of interpretation in an environment of action and judgment, whereas the input to interpretation is data in some form of representation. Theories of interpretation are not only found in the fields of logic and abstract data types with their various kinds of model theoretic, denotational and operational semantics, but also in process algebras and in behavioral formalisms such as Petri nets or transaction diagrams. On the other hand, content is being modeled in semantic nets, ontologies, databases and repositories. In the World Wide Web, content is contained in documents and in web sites, and is coded into link structures and services. The general question of how to appropriately modeling content, context and orientation to goals, is the basis of the more recent research areas of knowledge management and the semantic web. Human-Centric Communication puts a particular focus on this modeling question and intends to impose methodology and rigor to the reasoning in communication support.

The available techniques for this aspect might be the less developed compared to the following ones because human-centricity and liberal handling of data, as e. g., in the (overstressed) sense of Web 2.0, are relatively young fields compared to the classic ones of structuring and concurrent behavior of systems, although the necessity of considering individual needs has been discussed for some time [Ric83]. Therefore, it might be advised to focus on the more elaborate ones to find more results to reuse in the first steps towards modeling of HCCS, even if this means that we have to defer the treatment of modeling advanced topics of human-centricity such as adaptivity to physical conditions of the concrete system's environment or of the user. On the other side of the complexity spectrum, as the minimum requirement a formal approach for HCCS at least has to support data values for representing identities of its actors and the data to be transmitted.

Topology In HCCS, we have to deal with highly dynamic structures and behavior; most notably, the number of users known to the system can grow or decrease during runtime. In order to maintain a variable number of users, we also need a possibility to reconfigure the structure. With the term "topology", we subsume the general structure of a HCCS and its dynamics of changing actors and channels. There are several popular techniques for the modeling of system structures:

The Unified Modeling Language (UML) [OMG10] aims at a universal collection of different diagram languages for object-oriented modeling of all relevant aspects of arbitrary systems and has gained substantial acceptance and industrial relevance. Although we simply classify it in this category of topology for this overview, the UML is not limited to structural specifications but is also used for modeling behavioral aspects.

UML models support systems with dynamic structure in the sense that objects can construct other objects at runtime (e. g., in sequence diagrams), but changing behavior specifications in order to support dynamic role management is problematic because the meta model is considered immutable in normal cases. Structural consistency properties for the UML diagrams are expressed with the Object Constraint Language (OCL), but unfortunately the UML itself is very heterogeneous and although a lot of current research (also independent from the official UML contributors) is dedicated to formalizing sublanguages of the UML, up to now the languages' semantics are mostly explained informally in the official UML specification.

Graph transformation techniques [Roz97] support on the one hand graphs as a fundamental structure with different extensions for data and on the other hand the manipulation of them. Further, many transformation approaches are provided with results for analyzing transformation sequences. Because graphs are by their nature only structural objects, they lack a notion of behavior. However, appropriate graph transformation systems may also be used to simulate the semantics of behavioral systems such as Petri nets that we describe in the next item.

Recently, graph transformation has also been used for model transformation, i. e., translating (graph-based) models of one domain into another domain [TEG^{+}05]. Ongoing research in this area deals with correctness and completeness of the transformation systems w. r. t. the domains' syntax and semantics so that graph transformation may be interesting for supporting the ideas of content and translation of such between different Communication Spaces, too. Even aspects like access rights for role-based models [KMPP05] are among the many applications of graph transformations.

Interaction The dynamic aspect of a communication space's content and topology is captured by the term "interaction". Actions performed by the users cause transmission of content and data along channels on the one hand and changes in the topology of the HCCS on the other hand. As the human-centric interpretation of content w. r. t. a user's context and his environmental conditions, the possible interactions in the HCCS and their effects should depend on and regard the individual user's preferences. This may impose dynamics on the interaction processes themselves. We look at some popular formalisms for modeling interaction and system behavior:

Process languages For modeling interaction in concurrent communication systems, many process description languages such as process algebras have been developed. The notion of channels is clearly defined in these languages by the semantics of communication operators between processes, and actors are processes that are built with these operators and composed in parallel to send and receive messages among each other.

The interaction semantics of the Algebra of Communicating Processes (ACP) [BK84] requires that actions for sending and receiving data occur synchronized in the parallel communicating processes, which is helpful for broadcasting messages over the whole system. In addition, the calculus of Communicating Sequential Processes (CSP) [Hoa85] allows to specify the synchronized actions with an explicit interface for the parallel processes. The Calculus of Communicating Systems (CCS) [Mil80] even uses the action names for receiving and sending as channel names to synchronize (unicast) communication. But in all these calculi, process definitions are static and channels are implicit between parallel processes. A better choice for a calculus w. r. t. a dynamic topology would be the extension of CCS named π-calculus [Mil99], which features an operator for spawning new processes and allows to communicate channel names that can be used for responding via a particular channel.

The central notions of HCCS such as channels and communication are in the focus of these formalisms. Special communication needs and modes (such as broadcasting and multicasting, cf. Sect. 3.2.5) are typically modeled as protocols (for the actor processes) with the different process algebra approaches, possibly with coordinating system processes. However, such protocol models are not trivial. Alternatively, it is possible to further adapt the calculi's syntax and semantics, as the Calculus of Broadcasting Systems (CBS) [Pra91] has been set up.

Petri nets [Pet62] with their progeny in form of many more powerful extended variants are another widely used approach for concurrent and distributed behavior systems [Rei85a, RT86, MOM89, MM90, Win87, Bau90] that today offers excellent support modeling, simulation, and formal analysis of such systems.

High-level net classes are obtained by combining Petri nets with an appropriate data type part [JR91, Rei91], which is interesting for the content aspect of HCCS. Most prominent are colored Petri nets [Jen92, Jen94, Jen97], a combination of Petri nets and a high-level programming language, which is an extension of the functional programming language Standard ML. Colored Petri nets offer formal verification methods and an excellent tool support, which has been used in numerous case studies within a large variety of different application areas. Apart from this, there are algebraic high-level (AHL) nets [EPR94, EHP$^+$02], which give rise to a formal and well-defined description due to their integration of classical algebraic specifications [EM85] into Petri nets.

Although the primary intention of Petri nets is to describe processes, they obviously have a graph-like structure, which may be useful regarding the topology aspect of HCCS. There is plenty of related work about Petri nets for communication systems, e. g., in [ERRW03], but with a few exceptions they are not adapted to treat communication nets with dynamic structure. We discuss some particular approaches of Petri nets with dynamic structure in detail allowing the manipulation of running processes:

Self-modifying Nets [Val78] have been the first approach to nets with nonfixed behavior. The main idea behind this approach is that arc weights in self-modifying nets depend on the current marking.

Recursive Nets and Dynamic Transition Refinement follow the idea of flexible transitions. In recursive Nets [HP00, HB08], firing of transitions can spawn new processes in form of recursive nets that may return tokens as result to the spawning ancestor net.

The latter approach of dynamic transition refinement [KR07] is an extension of object nets [Val98] that features nets as tokens. These object nets can be used to refine transitions in the top net containing the tokens.

Mobile Petri Nets [AB09] are inspired by the notion of mobility in process calculi and try to integrate channels and dynamics in colored nets. To realize mobility with channels, in Mobile Petri nets the post domain of transitions may depend on the values of the consumed tokens representing place (and hence channel) names. Furthermore, the firing of a transition can create a new net depending on the previous one. The next firing step will then occur in the produced net, which realizes dynamic net evolution.

Cooperative Nets [SB94] consider an object-oriented approach to dynamic net structures. Nets are treated as objects that can access each other via the values of their tokens that represent names of the net objects. Transitions can be associated with actions on the objects that are references by the token names, i. e., sending token values to or consuming values from places of a referenced net. A transition can also call predefined constructors for new net objects.

These approaches treat changing of behavior in quite different ways but they all have in common that the changing of a Petri net is part of its firing behavior. From this follows that the firing rules for these net variants differ considerably from the common Petri net firing rule for static nets, which can be a problem for analysis as important results either are not proven yet or even are invalid for the particular variant. This is the case in Mobile Petri nets, which even cannot be visualized as a usual Petri net, i. e., as a bipartite graph, because of their marking-depending structure that precludes any static analysis.

Besides this theoretical problem, the concepts of reconfiguration may be inappropriate for our intention of HCCS: If we understand behavior of actors in HCCS as firings of transitions (cf. Sect. 3.2.4), we implicitly identify the actions with transitions. For self-modifying nets, recursive nets, and nets with dynamic transition refinement it is difficult to imagine how they can support a dynamic set of actors because the set of available transitions is either fixed or can merely be refined but it is not possible to create a new structure (for new actors) that can possibly interact with other parts of a net.

Both arguments can be applied to cooperative nets because they are marking-dependent and their set of available net object types has to be predefined in order to provide the necessary constructor operations for new objects. So, although it is possible to create new actor net objects, their behavior cannot be manipulated at runtime, but it depends on the concrete modeling methodology whether this is a serious problem.

In order to have the most flexibility for changes of a Petri net structure and even for the choice of the concrete Petri net formalism and firing rule, another approach seems promising:

Rule-based Reconfiguration of Petri Nets follows the idea to treat a Petri net as a graph-like structure and to formulate transformation rules similar to graph transformation rules for reconfiguration. The advantages are that there are already several rule-based Petri net transformation approaches, e. g., for P/T nets and AHL nets [EEPT06], which indicates that the transformation approach does not necessarily depend on the concrete net variant, and — which follows directly — that we can possibly use any well-elaborated Petri net variant and its analysis results for processes up to reconfiguration. Moreover, the transformation approach itself can provide some useful results for analyzing reconfigurations. In this thesis, we consider a reconfigurable net as a Petri net transformation system consisting of a concrete Petri net, which represents the current configuration, together with a set of transformation rules. [9]

We already mentioned that graph transformation systems can be equipped with rules simulating behavior semantics. But it might be advantageous to distinguish user behavior from reconfiguration and to possibly use standard results for the behavior analysis of Petri nets instead of transferring them tediously to an ad-hoc graph transformation system.

Simple and Intuitive Modeling For HCCS, we do not explicitly regard the aspect of human-centric modeling in the sense that the modeling process itself is optimized for the modeler's physical and mental condition or needs; but in general this is a desirable characteristic, so we note it here as a "soft goal". Diagrammatic specification models such as Petri nets and graphs supporting visual modeling and visual behavior simulation can have advantages for system modeling w. r. t. readability and understandability, although there is no standard measure for these properties [DVLS02]. Visual languages and models that are represented formally and precisely by them can usually also be understood mostly by people who are not familiar with the formalism with less effort than the linear textual ones[10] languages. This is justified by similar statements about visual programming languages [GP96, BWGP01].

2.4.3. An Integrated Approach for Modeling of Communication Spaces

From the previous discussion we can conclude that reconfigurable Petri nets are a promising candidate for a formal technique to describe Communication Spaces from a high-level viewpoint (see Sect. 2.4.1). They describe structures that can flexibly be changed at runtime by rule-based transformation and are equipped with a concurrent behavioral semantics. For the necessary operations on identity and transmitted data, especially the

[9] The project "Formal modeling and analysis of flexible processes in mobile ad-hoc networks" (forMAl-NET) [PEH07] had its focus on developing techniques for reconfigurable nets to model systems with dynamic structures, first only mobile ad-hoc networks and in its continuation also communication-based systems.

[10] Although commonly used so, *textual* is not really the opposite of *visual* but only its special case in one dimension, i. e., linear expressions in contrast to two- or even three-dimensional diagrams that are usually denoted with the term "visual". However, the actual opposite of *visual* languages are *verbal* ones, which we do not consider for formal modeling.

algebraic variant of high-level nets seems to be a good choice because rule-based reconfiguration of this net class has already been developed to some extent. Petri nets and the graph transformations are formally well-founded and provide many analysis results that can possibly be used to prove properties relevant for Communication Spaces. Moreover, Petri nets and rule-based graph-transformation are visual techniques, which may yield models that are easy to read and understand as well as intuitive tool support. Thus, when we mention reconfiguration of Petri nets in the following, we mean the rule-based approach that is similar to algebraic graph transformation with rules.

2.5. Communication Platforms

> Non ho mai capito questi social network,
> per me servono solo a fare i porci a letto.
> Ogni volta che nasce una nuova piattaforma,
> mi fa l'effetto di un libro che ho già letto.
>
> *(Caparezza)*

We narrow the focus to a special kind of systems in this section, namely Communication Platforms (CP), suggesting themselves to be considered under the aspects of Communication Spaces. With this, we can identify concrete requirements and features for the representative platform Skype as a popular example of CP in the next chapter. This allows us to make some assumptions that characterize these systems still under a high-level modeling viewpoint, from which we can pose questions for the representation of system parts and features. Then, we try to answer these questions by formulating appropriate modeling principles for Communication Platforms before we develop a modeling approach based on reconfigurable high-level nets and finally evaluate it on a case study for Skype.

Traditionally, the term "platform" has been used in the broad sense of architecture in hardware platforms (specific processor architectures) and computing platforms (combinations of hardware architecture and software/modeling frameworks), in general as something "one can build upon". There are many examples that propose platforms in combination with or supporting communication, e. g.,

- in [MSM97] a communication platform "enables the management application programs, which reside on the operations platform, to communicate with the network elements and the other operations systems by using management protocols" as an essential part of a general management and operations platform, which is "a set of reusable and basic software packages".
- [BGM$^+$03] treat Modular Communication Platforms as "standards-based communications infrastructure platforms and building blocks designed to deliver reduced expenditure, solution flexibility, TTM, and vendor choice."
- [Sie04] focuses on a platform for wirelessly-communicating smart objects that features a high-level description language for the typical recurrent structure of

context-aware applications that supports automatic generation of code, examples for typical context-aware communication services such sa context-aware device discovery and topology construction, and — most remarkably — grouping of objects by distributed tuplespaces for describing the infrastructure and inter-object collaboration.[11]

All in all, these approaches aim at subsystems or architectures that are meant to realize or support modeling of data transmission in the context of bigger systems that finally may provide services to some end-user.

A slightly different and more general view is given in [LNW03]. Here, "platform-based design" is opposed to model-based design as a "view from below" constraining possible design choices to those that can be easily implemented whereas hiding unnecessary details of the implementation technology.

In contrast to this standard meaning of platform, we take this notion to denote a special kind of communication system where the main aspects of Communication Spaces (content and contextuality, dynamic topology, adaptive and human-centric communication features) and their importance for these systems are almost obvious. We use the term "platform" in the sense of a ground to merely act on rather than to build upon and define a **Communication Platform** as

> a (possibly Internet-based) service with the main or only purpose to allow humans to communicate with each other. Essentially, the users are the only perceivable actors in these systems and thus every reaction of the system is triggered directly by the users' actions. Usually, the users can set preferences regarding their privacy and availability to communication, which are respected by the system.

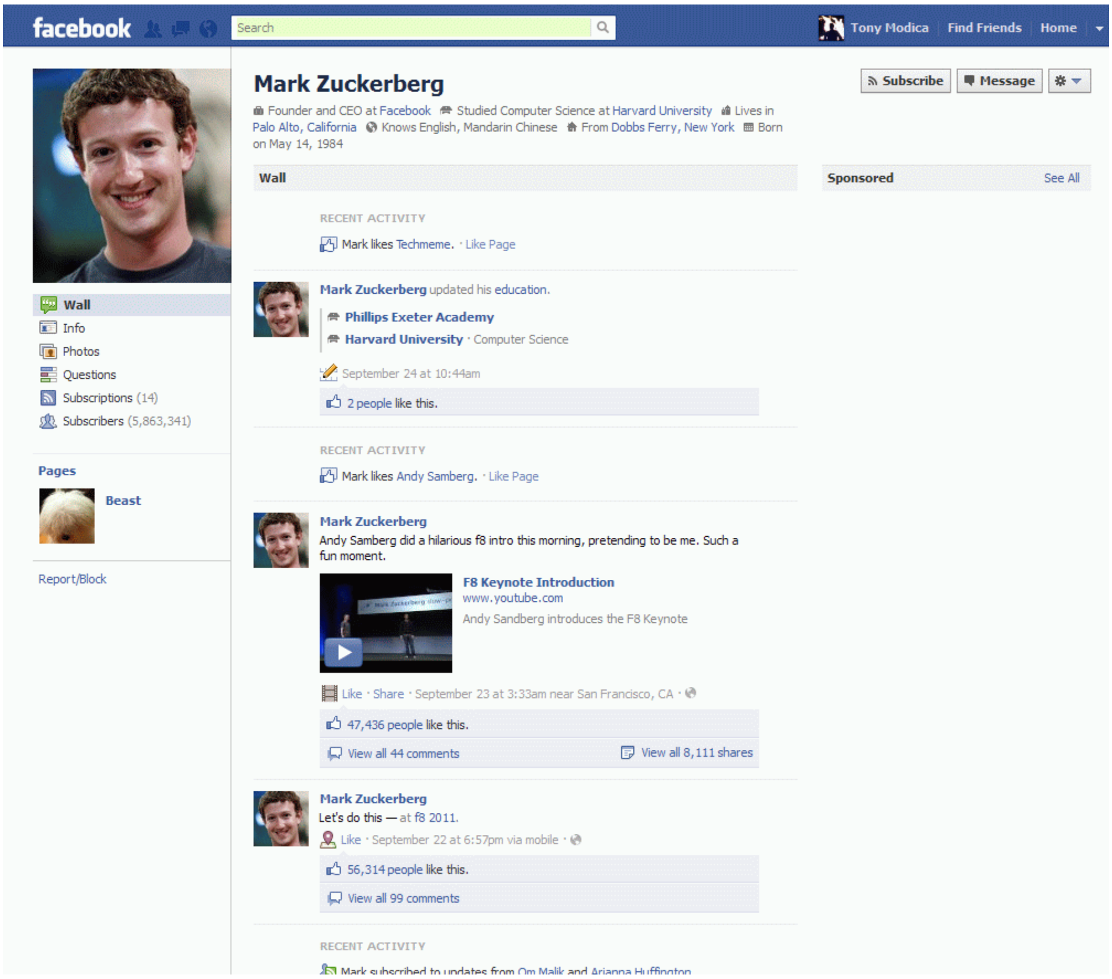
There are many popular applications and services that can be intuitively identified as Communication Platforms:

Facebook is a social network service, where people can connect with their friends.[12] Since its start in 2004, its number of active members has risen to over 800 million, of which more than 50% log on to Facebook everyday, accoding to Facebook's statistics. Mainly, personal profiles are accessed with a web browser via the Facebook website, where one can send and comment messages, announce events, share videos and photos, and let contacts know what he is doing or what he likes (see Fig. 2.3). But more and more members — currently more than 350 million according to Facebook — also use their mobile devices to access Facebook as there are applications for almost every mobile platform. The Facebook API allows to develop games and applications as well as to connect Facebook accounts with other websites, e. g., to provide a tagging-like feature to recommend and mark YouTube videos, news, etc. as "liked". More than 2 billion posts are liked and

[11] Although the complete approach of [Sie04] might be interesting in the bigger context of general Communication Spaces, it is still focused on realizing systems with small independent smart devices by programming them in order to achieve an overall system's functionality.

[12] http://www.facebook.com

commented on per day. Facebook is, after Google, the second most popular (i. e., traffic-generating) website in the United States according to the three-month Alexa traffic rankings.



Mark Zuckerberg

Founder and CEO at Facebook · Studied Computer Science at Harvard University · Lives in Palo Alto, California · Knows English, Mandarin Chinese · From Dobbs Ferry, New York · Born on May 14, 1984

RECENT ACTIVITY

Mark likes Techmeme. · Like Page

Mark Zuckerberg updated his education.

Phillips Exeter Academy

Harvard University · Computer Science

September 24 at 10:44am

2 people like this.

RECENT ACTIVITY

Mark likes Andy Samberg. · Like Page

Mark Zuckerberg

Andy Samberg did a hilarious f8 intro this morning, pretending to be me. Such a fun moment.

F8 Keynote Introduction
www.youtube.com
Andy Sandberg introduces the F8 Keynote

Like · Share · September 23 at 3:33am near San Francisco, CA ·

47,436 people like this.

View all 44 comments · View all 8,111 shares

Mark Zuckerberg

Let's do this — at f8 2011.

Like · September 22 at 6:57pm via mobile ·

56,314 people like this.

View all 99 comments

RECENT ACTIVITY

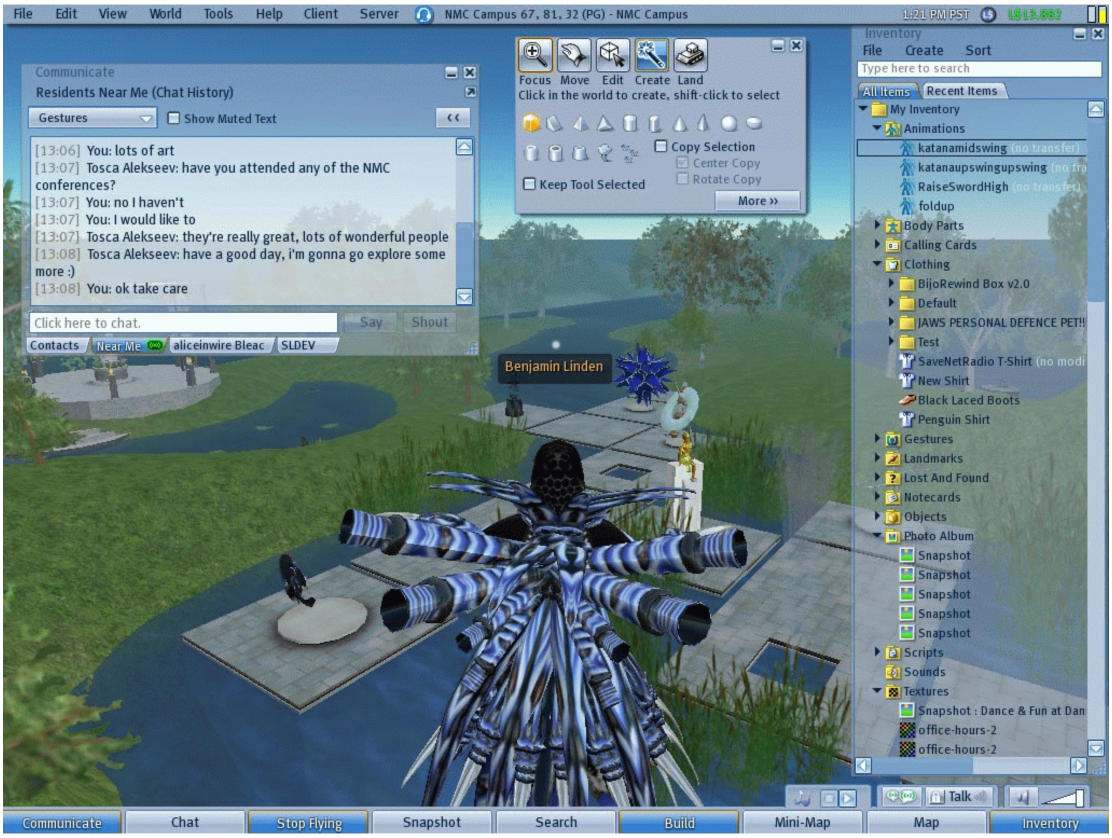
Mark subscribed to updates from Om Malik and Arianna Huffington

Figure 2.3.: Facebook page with messages from its founder Mark Zuckerberg

Second Life is a 3D virtual world founded in 2003 where people can command virtual representations of themselves, called avatars, to interact with other avatars.[13] The rather complex interface to Second Life is similar to third-person shooter games and is provided by a viewer program that has to be installed on the user's computer (see Fig. 2.4). Second Life features its own economy based on the virtual currency "Linden Dollar", which is traded by Second Life's creator Linden Labs for real money. Inside Second Life, people can spend Linden Dollars for virtual goods designed or programmed by other members. These goods may be virtual estate, buildings, clothes and accessories or even behavioral features such as gestures for

[13] http://secondlife.com

the avatars. Many popular companies of various commercial sectors have established stores with virtual counterparts of their products in Second Life such as Adidas, American Apparel, Apple or IBM. According to the last economy reports on the official website, in the second quarter of 2011 Second Life had 1,042,000 average monthly unique logged-in users, of which 44% also participated in the economics.



[13:06] You: lots of art
[13:07] Tosca Alekseev: have you attended any of the NMC conferences?
[13:07] You: no I haven't
[13:07] You: I would like to
[13:07] Tosca Alekseev: they're really great, lots of wonderful people
[13:08] Tosca Alekseev: have a good day, i'm gonna go explore some more :)
[13:08] You: ok take care

Figure 2.4.: The Second Life Viewer client

Skype is currently a widely used Internet telephone (also called VoIP — voice over IP) software and can be obtained and used free of charge since its foundation in 2003.[14] In its basic version it features most of the functions a communication tool is expected to have such as direct (video) calls, chats, multi-user conferences and contact management. Besides the possibility to call other Skype users for free, one may also call mobile and landline phones worldwide with costs, for which credits or (regional) flatrates can be purchased. Furthermore, Skype provides the service to bind a local landline telephone number to a Skype account to allow incoming calls from regular phones. Skype applications are available for some mobile platforms as a cost-saving alternative that exploits Internet flatrates, which

[14] http://www.skype.com

> are often included in mobile contracts. In the first quarter of 2011, Skype was reported by Telecompaper to have 663 million accounts registered, but users are not limited to have one account.[15] Anyway, around 25 million clients can be expected to be online regularly, which is displayed in the client (see Fig. 2.5). Recently in May 2011, Skype has been incorporated as a division of Microsoft, after it was taken over first by eBay in October 2005.

Of course, these examples can also be described as ordinary communication systems, but if we recall the discussion in Chap. 2, we see that this term also applies on more technical systems serving just some computing purpose. The definition of Communication Platforms above allows us to think about the details of modeling by narrowing the wide focus of Communication Spaces to concrete instances such as the given three and pointing out the typical features they have in common and we are interested in. Without this characterization, Skype would rather be categorized as a pure telecommunication application, Facebook as a social network, and SecondLife as a virtual 3D world simulation, without finding a least common denominator other than "communication".

[15] In the white pages function, the Skype client currently claims to search for matches within one billion registered users.

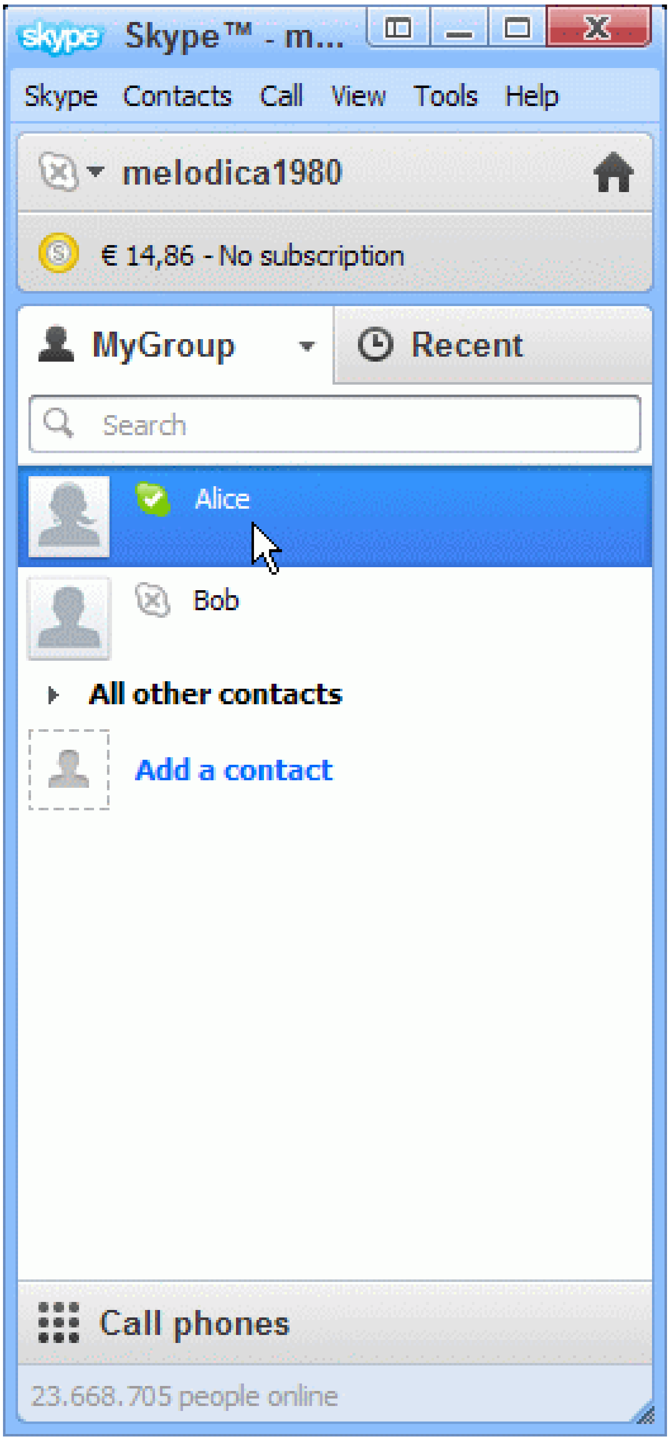


Figure 2.5.: The main window of the Skype client

3. Modeling Requirements of Communication Platforms and Skype

> The works and customs of mankind do not seem to be very suitable material to which to apply scientific induction. A very large part of space-time must be investigated, if reliable results are to be obtained. Otherwise we may (as most English children do) decide that everybody speaks English, and that it is silly to learn French.
>
> *(Alan Turing)*

In the previous chapter, we have outlined the vision of Communication Spaces and Platforms and their main aspects with a preliminary discussion about candidates for formal techniques. This leads to the general plan to use (and possibly adapt) high-level Petri nets with rule-based reconfiguration for modeling of Communication Spaces from a high-level abstract viewpoint. As a concrete instance and representative for Communication Platforms that provides the most common and important features of such systems, we present Skype in detail. In the end, this yields modeling principles and particular challenges for a realization with the technique of reconfigurable Petri nets, which we approach in the next chapter.

3.1. Skype in Detail

> Ich kenne einen, der bei Facebook arbeitet — und keine Freunde hat.
>
> *(Marc-Uwe Kling)*

As a representative concrete Communication Platform, we consider Skype (see [AAD07] for a comprehensive introduction) as a reasonable object of a modeling case study because it offers the common basic interesting communication features of most Communication Platforms, but still its simple interface helps to concentrate on modeling the user behavior and on the aspects of Communication Spaces and Platforms.

Anyway, Skype has many features that might be too ambitious to be approached in a first discussion about how to represent and describe its vital parts in terms of Communication Spaces in the modeling focus we decided on in Sect. 2.4.1 and with using a technique based on reconfigurable Petri nets (cf. Sect. 2.4.3). With this in mind, we agree on the following restrictions to extract a reasonable core functionality of Skype to be treated in the modeling case study:

- According to our interest in high-level modeling, we consider communication features and user activities only and refrain from modeling the user interface or general software usage. We do not distinguish the platform (hardware, operating system) on which the Skype client application runs.

- The details of data transmission on lower levels, i. e., beyond the actual user action of sending and receiving and their effects on the actors, is out of scope as well as the handling of concrete networking issues and protocols.[1]

- To have a definitive boundary of the Communication Space, we consider internal communication to other users who run the Skype client, only. We ignore external phone calls (SkypeIn, SkypeOut) and Short Message Service (SMS), as well as all other features that are not free to use inside the Skype net.

- We do not distinguish video communication from phone calls as they are basically identical up to the data that is transmitted.[2]

- Additional plug-ins may provide additional functionality such as automatic answering of calls, which we do not consider.

- Former versions of the Skype clients had some additional features that are not longer supported in the current version 5, and which we ignore therefore:[3]

  SkypeMe was a special status users could set in their clients. This resulted in temporary overriding of privacy settings so that other users could call without explicit permission directly from the search tool.

  SkypeCasts were public conferences that had to be announced on the Skype website and were available only during the announced time slot. The creator of a SkypeCast was the only person able to talk in it, all other participants were initially only allowed to listen. The creator could grant participants temporarily the right to talk. Another discontinued feature similar to SkypeCasts were Public Chats.

  Shared groups were a special kind of contact groups with several effects: First, a user set up a shared group resulting in invitations sent to each group member. When an invited user accepted, he was added to the shared group and could access it via his own contact list. Every member of a shared group could invite further users to it so that these users appeared in each contact list of the other group members as well. Users could only delete themselves from the shared group by deleting the whole shared group.

[1] Anyway, Skype's communication protocol and source code are undisclosed and proprietary without available documentation, although there have been many attempts to analyze and decipher it, e. g., [DK06, ZZZY10]. But it is common knowledge that it uses a peer-to-peer structure with super-node clients [BS04].

[2] Although, the feature of group video calls with more than two users was beta in version 5.0 of the Skype client. In the current version 5.1, this feature that has to be paid for by subscribing for a Skype Premium account or by using the Business version.

[3] Actually, on the Skype website besides the following items the former Skype version 4.2 itself is listed as a "Discontinued feature".

> With SkypeCasts and shared groups, two quite unique features were abolished, but they may have not been crucial or even important to most of the Skype users, unlike newer features like video conferences.

This reduces the features of our interest to the following that we describe in detail:

3.1.1. Contact Management

Typically, users want to control privacy settings in communication applications like Skype. Other users should call or chat only if they have been allowed to do so by the one that they want to contact. Commonly, this is solved with contact lists (see Fig. 2.5) in combination with specific privacy settings (see Fig. 3.3). Additionally, users can explicitly block contacts (from the contact list or deleted contacts) so that both cannot further call or send messages to each other. Usually, blocked contacts are automatically deleted from the contact list and put again on the contact list if unblocked later. But still, the history of communications with blocked contacts is accessible. Actually, when a user deletes one of his contacts, he has to ask again for permission if he wants to add the deleted again to his contact list. But, strangely, the Skype client also allows users to block already deleted contacts so that after unblocking the deleted contact again, it will automatically be restored in the contact list.[4] Regarding that Skype allows this circumvention, we may assume that users do not have to ask for permission to restore deleted contacts at all. Anyway, a user can still call or send messages to other users by accessing them via his history log or the white pages even if they are not present in his contact list (by a window as shown in Fig. 3.1). The presence of a contact on the contact list only affects possible communication in combination with the privacy settings shown in Fig. 3.3.

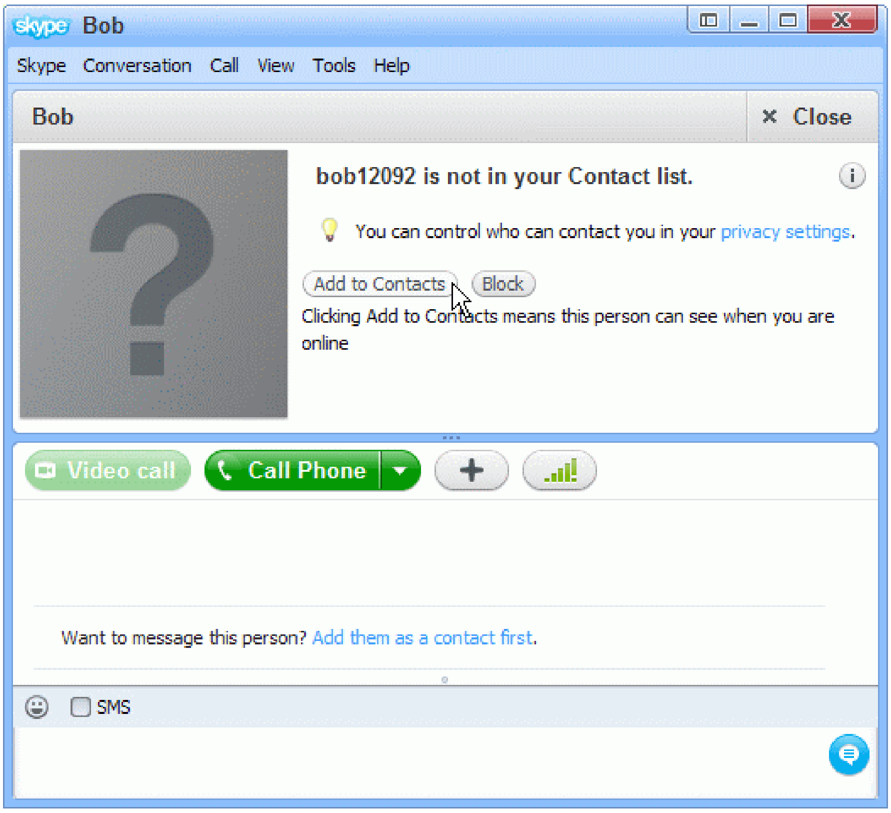
Skype offers a function to search for new contacts in a white pages directory of all registered Skype users. From the search results, users can be asked if they accept to be added to the requester's contact list, which of course can be denied (see Fig. 3.2). If the options are set accordingly under the "Notification" tab (below the privacy settings), Skype notifies a user when any of its contacts changes its online status.

Contacts can be organized in categories, which does not affect any other action. Further, after having participated in a group conversation (chats or conferences), on can access it through the log under the "Recent" tab in the main window or even store it as shortcut references in the contact list (see Fig. 2.5).

3.1.2. Calls

A user can directly call a user on his contact list as in a telephone call as long as the callee has not put the caller on the black list. For performing calls and other communication, Skype opens a communication window for the selected contact with buttons for the appropriate actions (see Fig. 3.4). For contacts, who have not given the authorization to be put on the contact list, the interface windows look rather as in Fig. 3.1. The callee

[4] There seem to be some inconsistent or unintuitive possibilities of acting in the Skype client, for which we should just make a choice for a uniform and consistent behavior. See also the following paragraph about handling of groups.



bob12092 is not in your Contact list.

You can control who can contact you in your privacy settings.

Clicking Add to Contacts means this person can see when you are online

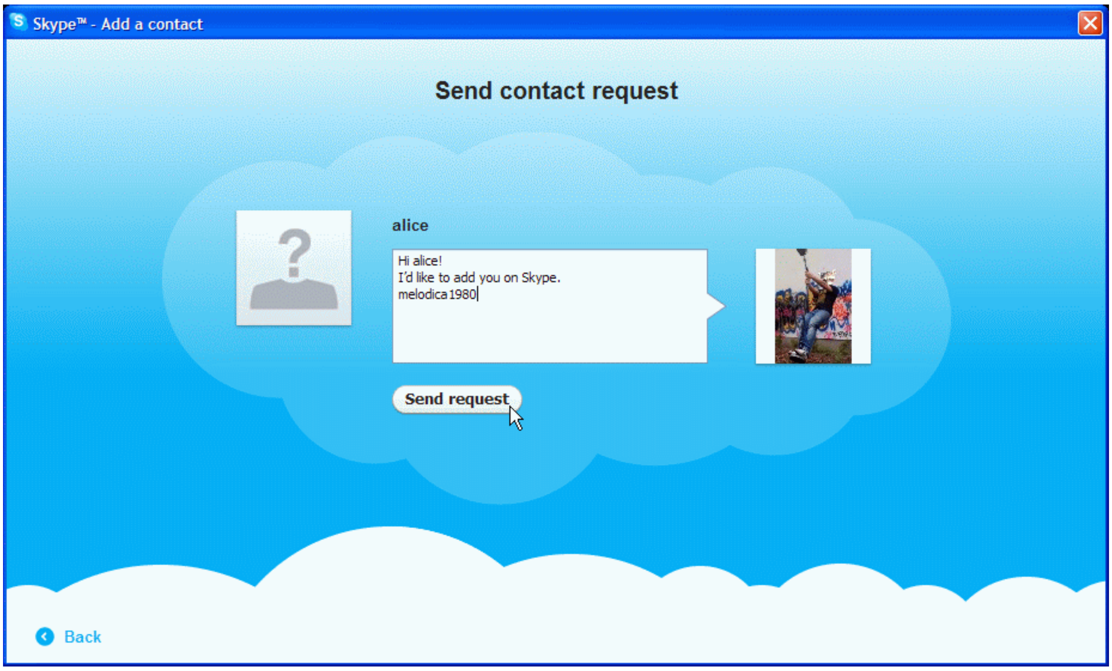
Want to message this person? Add them as a contact first.

Figure 3.1.: Interface for an unauthorized contact

gets notified by its Skype client (depending on its status by ringing or just quietly) and he may then accept the call unless he is already in another call or conference. In this case he has to end the current call or put it on hold before accepting the incoming one. This means that calls and conferences are exclusive communications and a user can only participate in one at a time.

3.1.3. Conferences

A conference is a generalized form of direct call where several participants (up to a technically bounded number) talk to all other participants. A direct call can be considered as special case of a conference with two participants (see Fig. 3.5). In Skype, a conference is always held in the context of a group that either has been created before or that is created implicitly when a conference is started directly from the contact list (not within a group). The "first" user of a conference is designated as its host who is equipped with special rights in this conference. We examine these rights in detail in the following section about chats because chats and conferences are both handled by managing groups in the Skype client. As the groups can be managed during a conference, other users can be added and kicked depending on the corresponding rights (see Figs. 3.6 and 3.7). If the host quits a conference, it is terminated, i. e., the other participants cannot continue



Hi alice!
I'd like to add you on Skype.
melodica1980

Figure 3.2.: Request for adding a contact

this conference.

3.1.4. Call Forwarding

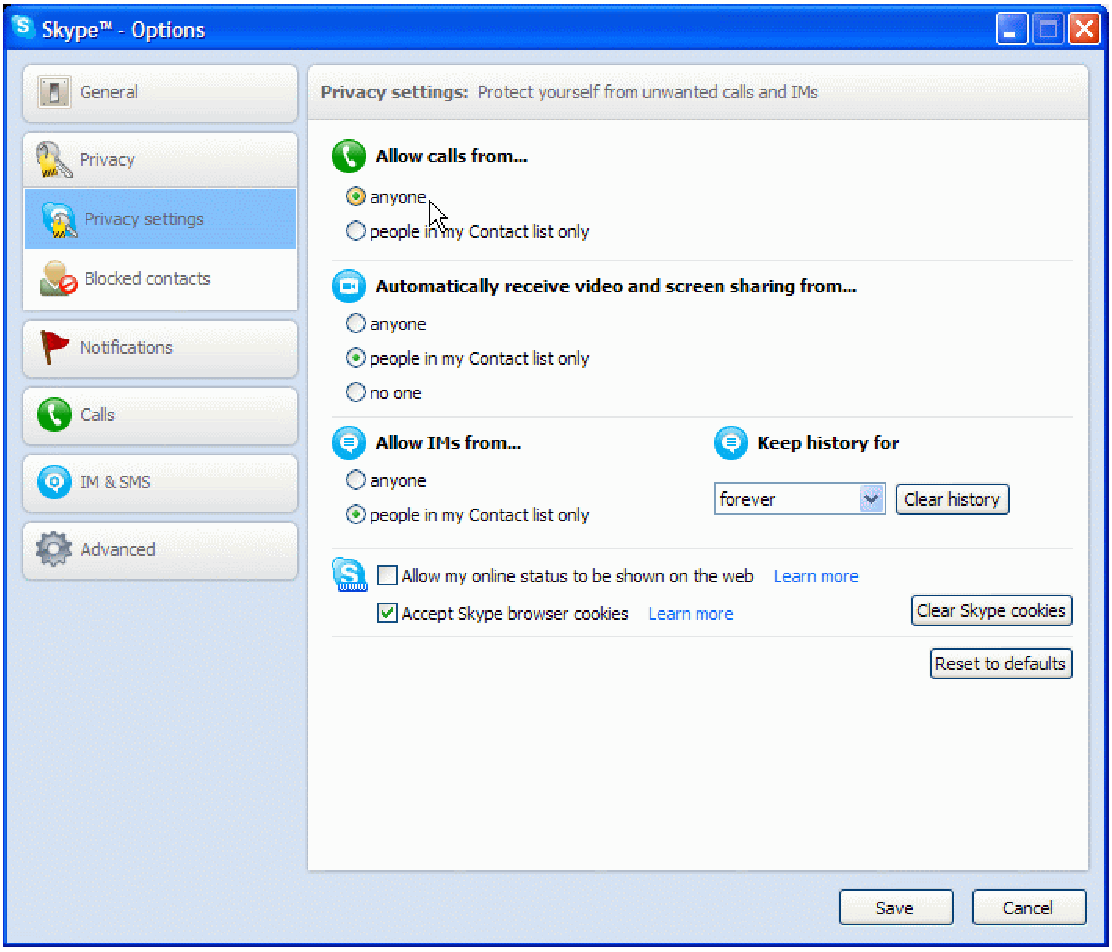
Known from service hotlines in the field of commercial communications, call forwarding means that the callee transfers an incoming call to another contact in his list as if the caller would have called the contact he is being forwarded to directly, even if the caller does not know this contact. In Skype, call forwarding (also known as transferring) only works for direct calls and is no longer available once a direct call is expanded to a conference of at least three people, even if later on the group is reduced to two members again.

Similar to call forwarding is the feature that allows a user to add an incoming call to a call or to a conference he is already on.

3.1.5. Chats

Chats are similar to conferences but in contrast they based on sending instant messages that do not have to be perceived immediately as acoustic messages in calls, wherefore we classify chats as nonexclusive communication. In Skype, a chat always happens in a group so that we can treat the notions of chats and groups as synonyms in the Skype world. The receiving Skype client notifies its user on new messages.[5] A Skype user can

[5] In fact, the Skype network employs support servers allowing to send messages to other clients even if they are offline so that they receive the messages when they activate their client. Recently, this feature caused a major blackout that shut down most of the Skype network. See http://blogs.



Privacy settings: Protect yourself from unwanted calls and IMs

Allow calls from...

● anyone
○ people in my Contact list only

Automatically receive video and screen sharing from...

○ anyone
● people in my Contact list only
○ no one

Allow IMs from...

○ anyone
● people in my Contact list only

Keep history for forever

□ Allow my online status to be shown on the web
☑ Accept Skype browser cookies

Figure 3.3.: Privacy settings in Skype

participate in as many chats as he likes in parallel, add references to group chats (as in Fig. 3.8 in his contact lists, and look through each chat's history when its group window is opened. It is possible to delete the history of a chat in the Skype client. Note that a user always receives all messages sent by any other group member as long as he is still a member of the same group; Skype ignores the privacy settings and whether users have blocked each other in this case.

The host's and participants' rights in chats and conferences The possible actions of a user in a conference or a group chat depends on his role in the group and in the conference. As said above, we call the initiator of a conference the *host*, but this user is not necessarily the owner of the group in which the conference takes place, who for example is the user that selected a set of his contacts and founded the group by sending an instant message as in Fig. 3.6. The user who founded the group is always the group's owner and is the only one who can remove users from the group!

skype.com/en/2010/12/cio_update.html

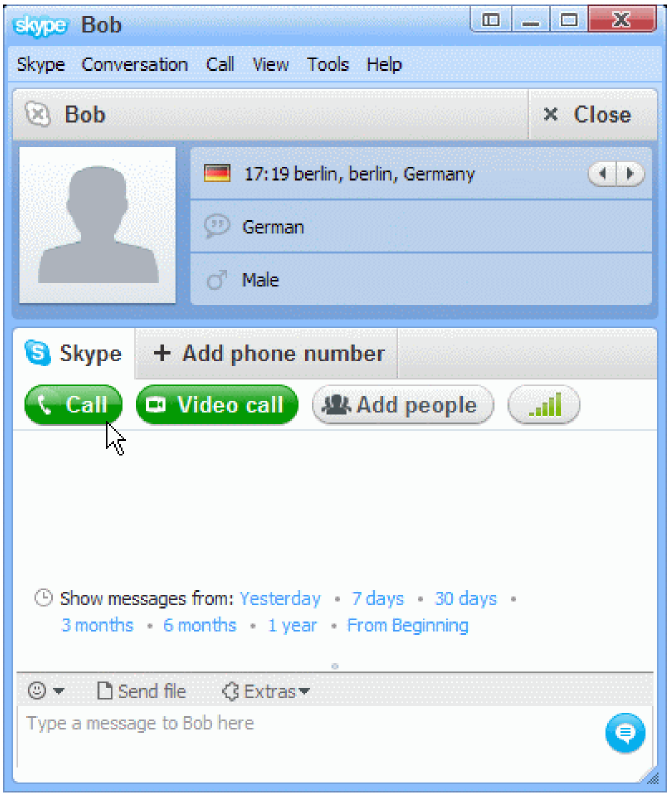
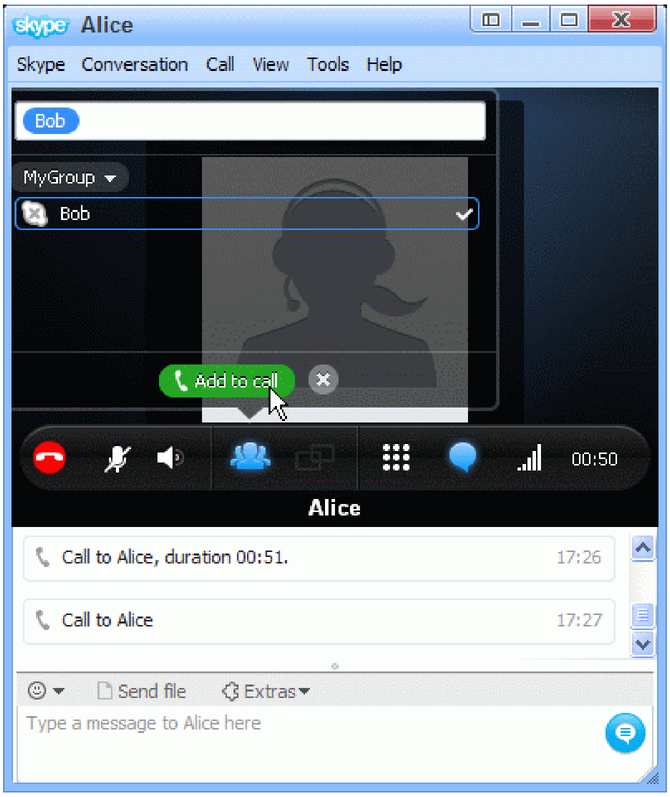

Bob
Skype Conversation Call View Tools Help
Bob
× Close
17:19 berlin, berlin, Germany
German
Male
Skype
+ Add phone number
Call
Video call
Add people
Show messages from: Yesterday • 7 days • 30 days • 3 months • 6 months • 1 year • From Beginning
Send file
Extras
Type a message to Bob here

Figure 3.4.: Initiating a direct call to Bob

The context menu action “New Group Conversation” just does the same as “Send IM”, which is opening a group window as e. g., in Fig. 3.8.[6] Starting a conference always implies creating a group in which the conference takes place, so that it can be considered as a shortcut to creating the group and pushing the “Call group” button with the only difference that unavailable contacts are also removed automatically from the group. Possible actions in groups and running conferences are the following ones: adding new users from the own contact list to a group (if neither has blocked the other), removing users from a group, and leaving a group (i. e., removing oneself). In a running conference, there are further the possibilities to hang up oneself or another user in a conference without removing him from the group and to rejoin a possibly hung-up group member to the running conference. Some actions imply others, for example if someone leaves a group he will also hang up automatically if he is in a running conference, but this is not necessarily true vice versa.

Another possibility to create a group or a conference indirectly is to start a direct call and then add other users to it from the own contact list (even blocked contacts) as in Fig. 3.5. Note that in this case Skype also distinguishes the first callee (the one the

[6] Skype does not treat direct calls or chats, i. e., with only two people, as groups but once a group is created by having a group chat or conference it will remain a group even if reduced to less than three members. In principle it is possible to reduce groups to one member or even to empty groups.


skype Alice
Skype Conversation Call View Tools Help
Bob
MyGroup
Bob
Add to call
00:50
Alice
Send file
Extras
Type a message to Alice here

Call to Alice, duration 00:51. 17:26

Call to Alice 17:27

Figure 3.5.: A running direct call with Alice before adding Bob

host called) from the other group members that were added later and grants him in the conference (extended from a call) almost the rights of the call host (who in this case is also the group owner). Table 3.1 gives a comprehensive overview over the complex rights of the conference host, the group owner, the first direct callee, and the other group members. We have to distinguish the rights of the group owner from the rights of the call host in a conference. In the last row, the owner's or the other members' rights may apply for the call's host, and in the other rows, the host's or the other members' rights may apply for the owner.

Even if the owner or the direct callee is removed from a group and readded and rejoined later, Skype will remember his role and grant him extra rights. This table is the result of experiments that we performed on the Skype clients and, to our knowledge, is not described in this detail in any other documentation of Skype. Thus, we cannot guarantee that these rules cannot be bypassed or overridden by unusual extensive group management or other operations in the clients. Moreover, the rule that all group members are allowed to remove other users as soon as the owner has left the group (and not been readded) seems to be a random choice and it is unlikely that users will ever encounter or even notice it in everyday use. This behavior might even change with future versions.

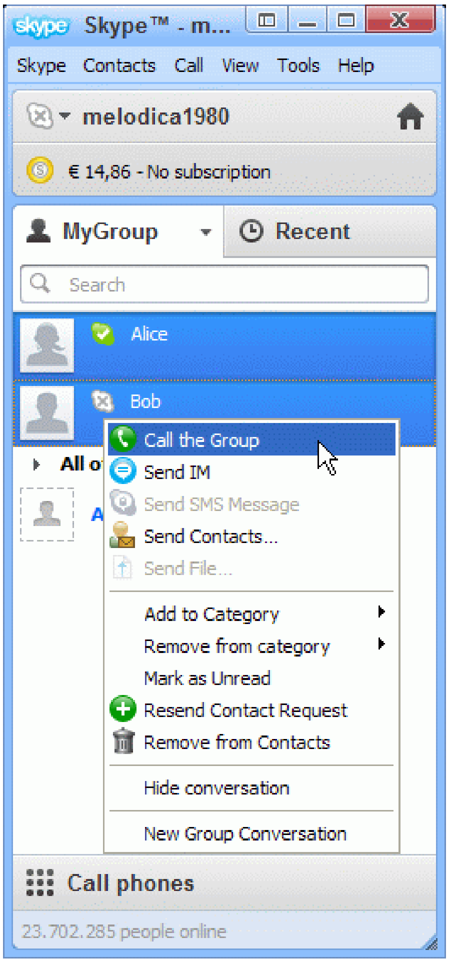


Figure 3.6.: Initiating a conference to Alice and Bob

Handling of groups However, management of groups in the Skype client is not clearly explained in any documentation and to some extent unintuitive: One may select a set of contacts and start a chat and invite and kick contacts while chatting as in conferences. If one tries to create a new group chat for a set of contacts and he already has created a reference to the group chat of the same set of users in the same order, Skype automatically opens the existing reference. This might seem sensible, but as a curious "feature", one may store an additional reference for a different group of the same set of members (resulting from inviting/kicking participants) as well in the contact list and have two parallel group chats with the exact same set of contacts (see Fig. 3.9). It is confusing (or at least unexpected) that Skype always opens the first — i. e., the oldest — group in its "Recent" history that fits with the selection, even if the user has explicitly left this group and is unable to participate anymore or even to rejoin without getting added by the other participants! We do not regard this as a special feature, but more as peculiarity of the client's user interface and rather assume for the modeling that groups can be created, manipulated, and accessed freely.

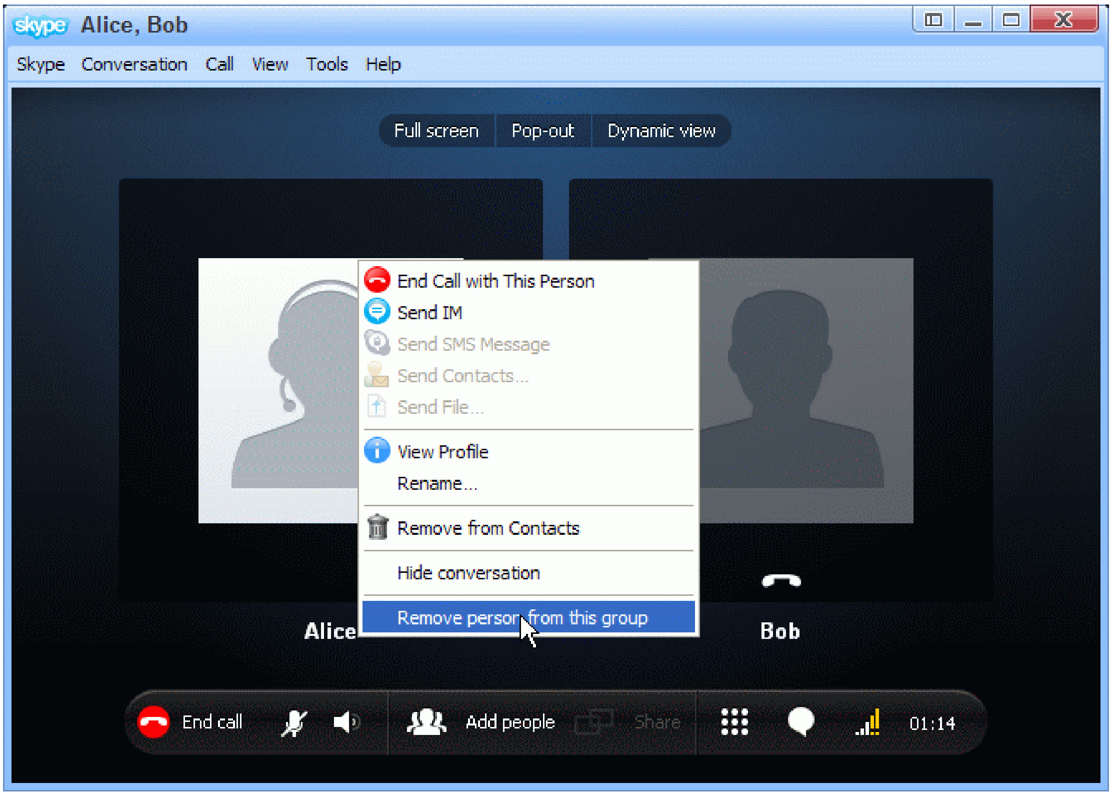


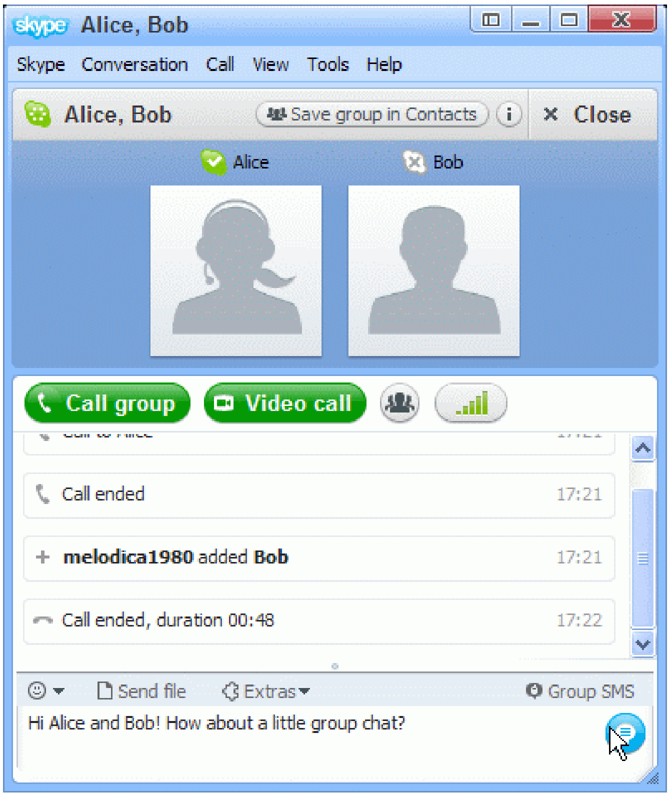
Figure 3.7.: Removing Alice from a conference

3.1.6. Use Case Scenarios

The following describes a typical sequence of actions between some Skype users.

First, three new users Alice, Bob, and Carol, who know each other in real life, register with Skype and activate their account. They choose the options for the privacy settings of their clients that every other Skype user may call but that only known contacts may send messages. Alice looks up Bob in the Skype white pages directory and sends a request for exchanging their contacts (see Fig. 3.2). Bob accepts this offer so that both Alice and Bob now have the others contact on their contact list. After that, Bob requests Carol's contact, which he accepts to exchange likewise. Now, Bob knows Alice's and Carol's contact, but the women each only have Bob as contact. Alice calls Bob and ends this direct call after some pleasant talking.

Bob selects the contacts of Alice and Carol from his contact list and founds a new group of three users, of which he is the owner (see Sect. 3.1.5). In the group, Bob announces via a group message (as in Fig. 3.8) that he is going to start a conference with Alice and Carol. Alice and Carol both agree to the proposed time for a conference by replying messages. Note that Alice gets Carol's group message and vice versa although their privacy settings are set to "only messages from know contacts" because group messages are transmitted even if the contacts do not know each other. At the appointed time, Bob starts the conference and Alice and Carol join in (see Sect. 3.1.3). Then, Bob hangs



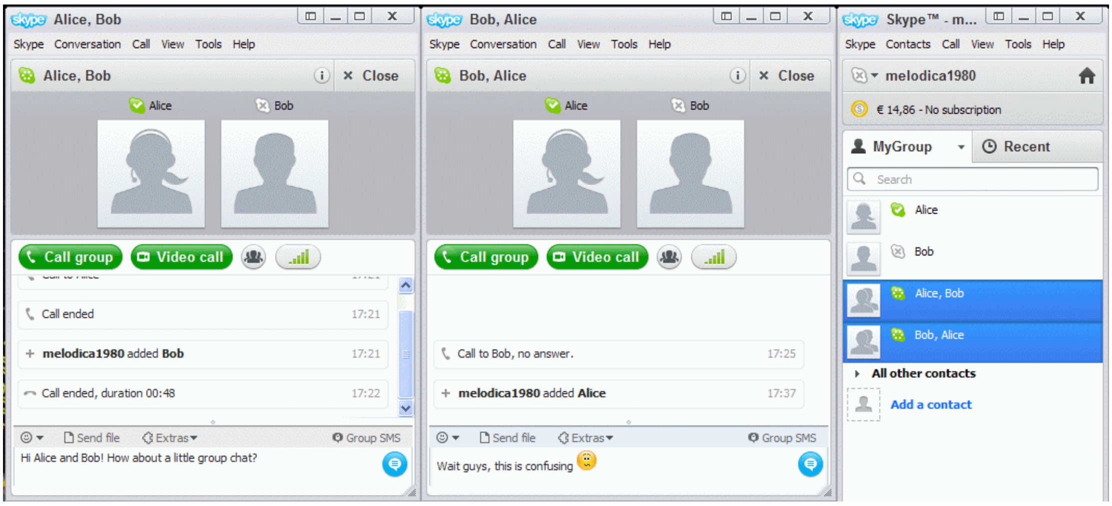
Call ended 17:21

melodica1980 added Bob 17:21

Call ended, duration 00:48 17:22

Hi Alice and Bob! How about a little group chat?

Figure 3.8.: Example for group chat



Call ended 17:21

melodica1980 added Bob 17:21

Call ended, duration 00:48 17:22

Hi Alice and Bob! How about a little group chat?

Call to Bob, no answer. 17:25

melodica1980 added Alice 17:37

Wait guys, this is confusing

melodica1980

€ 14,86 - No subscription

Alice

Bob

Alice, Bob

Bob, Alice

All other contacts

Add a contact

Figure 3.9.: Example for confusing group management

up, which quits the whole conference.

After a while, Alice starts a new conference in the group created by Bob, but Bob is offline at this time and only Carol joins in. Alice and Carol have a quarrel and Carol

conference:	**conf. host (of initial call)**	**first callee (of initial call)**	**other group member**
hang up self/for others	✓ in extended call only own joined users; quits whole conference if hangs up himself	✓ like host	× only self
(re)join self	× (hang-up quits whole conference)	× like host	✓
(re)join others	✓ if not blocked (in either direction); in extended call only own added users	✓ like host	×
group:			
leave group	✓ implies hang-up and ends a running conference	✓ like host	✓ implies hang-up if joined in conference
add others to group	✓ if not blocked by self (in either direction); others join conference automatically	✓ like host	✓ if not blocked by self (in either direction); added users wait to be joined by host
	group owner		
remove others from group	✓ implies hang-up if other joined in conference	✓ but not the group owner (the initial call host); implies hang-up for other	× only leave self; may remove all other users when group owner leaves (but still not the original owner if he gets readded)

Table 3.1.: Action rights of users in groups and conferences

blocks Alice in her client because she does not want to be called by Alice again in the group nor receive messages from her (see Sect. 3.1.1).

Later, Alice tries to start a conference again, in which only Bob joins, who meanwhile has gone online again. Carol has blocked Alice so that Carol's client does not ring on attempt of Alice to start a conference in the group. Alice convinces Bob (being the group owner) to remove Carol from his group. In her place, Alice invites her new friend Dave to join the group, whose contact she has got recently. As Dave joins the group, he automatically also joins the running conference, because Alice is the host of this conference (see Table 3.1).

3.1.7. Interesting Properties for Models of Skype

As a preparation for further discussion of model properties and how to formulate, ensure, or validate them properly, we gather some noteworthy characteristics of Skype. In a formal model of Skype, these should be reflected in some verifiable properties of the model.

User names It is evident that each user in Skype must have a user name in order to be addressable by other users. Such a user name may be an internal qualifier (like for objects in object-oriented programming languages) that differs from the display name, but the names must be unique to avoid confusion and misrouted data.

Exchanged contacts If a user A has another user B on his contact list, this must be the result of a contact exchange, i. e., the other user B should have user A on his contact list as well or at least have deleted A (with the option to restore this contact back to the contact list).

Special roles There are situations in Skype in which the number of certain roles for the affected users is constrained. For example,

- in each group there must be exactly one owner (cf. Sect. 3.1.5), i. e., one distinct user who can remove every other user from his group,
- in each conference there must be exactly one host, i. e., one distinct participant who quits the conference if he hangs up, and
- in a direct call only the callee can forward the call to another user selected from his contacts.

Exclusion of actions — mutual and state-related Some actions and model states explicitly exclude other actions. For example, there are many situations in which a user cannot start a call: He already may be in a another call, which he has to put on hold to start another call, or he may simply be offline. There are also actions such as the contact exchange that are mostly independent of other actions. A user can ask as many users for contact exchange as he likes at the same time and in almost any situation; the only situations preventing a request for a contact exchange are that there is already a pending contact request or that the addressed user is already on the contact list. Moreover, a user can be participating in a call or in a conference (actively or by holding it) if he is not offline, only.

3.2. Principles and Challenges for Modeling Communication Platforms

> I believe my objection to the notion of "problem" is due to my deep conviction that the moment one labels something as a "problem", that's when the real problem starts.
>
> *(Raymond Smullyan)*

With Skype as a typical and representative concrete system for our narrowed focus of Communication Platforms as a kind of Communication Spaces, we can start to ask questions that help us to develop ideas of how to get to a formal modeling approach with reconfigurable Petri nets. The first questions are of course about the characteristics of Skype that are interesting for Communication Platforms in general and which we want to be able to validate with our formal approach as properties of a formal model. With these properties in mind, we further concretize our intention of "high-level modeling of user–system behavior and communication features" that we set as a first focus for modeling Communication Spaces in the work at hand (cf. Sect. 2.4.1). Then, several general fundamental questions arise about how features of Skype and Communication Platforms are identified with parts of this general concept, in particular the transmission of data in a net. In the end, the answers that we decide to follow lead to modeling principles and design decisions for modeling Communication Platforms and which problems and challenges we have to face for reconfigurable Petri nets for properly modeling the aspects of Communication Spaces. It may turn out that there are some fundamental needs that we have to approach by extending the formalism of reconfigurable Petri nets just in order to even be able to represent basic behavior and dynamic structures before we go on to more elaborate problems as for example environment- and context-dependent data.

3.2.1. Interesting Model Properties from the View of Communication Platforms

When we attempt to model a Communication Platform like Skype with a formal approach, we also want to validate properties for this model that can be expressed in terms of Communication Platforms. We can abstract some characteristics of the real Skype system in Sect. 3.1.7 to desirable concepts and properties of models of Communication Platforms in general.

Model integrity For most Communication Platforms and their models, one has an intuitive idea of several conditions for structural and behavioral integrity. We discuss some examples of such conditions.

A central property of all systems for communications is that the communicating parties be addressable unambiguously. In order to achieve this goal in models of Communication Platforms, where everything is focused on the users, we have to equip the users' representation with unique identifiers. Although the representation of a user's name may differ from the actual identifier in concrete system, we simply denote the distinguishing attribute of users with the common term "user name". It is obviously reasonable to demand for the integrity of a Communication Platform model that all users have different user names at any time.

In the context of a communication situation, every affected user has a special role, which enables the user to perform certain actions. Note that a user's role in a context can be characterized by a thorough description of the possible actions of this user and their effects that affects other users in this context. In Sect. 3.1.7, we enumerate some conditions for a model of the concrete Communication Platform Skype. Note that constraints on roles are highly dependent on the concrete system to model, in contrast to unique user names, which is a model-independent property for the abstract notion of Communication Platforms in general. Model-dependent properties formulated for a concrete Communication Platform may not be transferable to or not even be formally well-defined for other Communication Platform models. But it is desirable for general properties of Communication Platforms to find a possibility to formulate them in a way that they can be checked easily in arbitrary platform models, without necessarily having to adapt them to the concrete model.

Because the state of a Petri net-based platform model is represented in the structure and the marking of the platform net, we regard state-related dependencies of actions as issues of model-dependent integrity and expect that the formal integrity validation techniques we develop in the following cover this as well.

Action independence From the discussion about dependence of actions to other actions (rather than to states represented in he model structure) in Sect. 3.1.7, we can generalize to the common question of independence of actions in behavioral systems. For testing a model and validating correct behavior, it is an important aspect to be able to check whether two possible actions in an actual platform configuration are independent in the sense that they can occur in any order and lead to the same result configuration. If the effects of actions in the model are quite complex, it is even more valuable for the platform designer if the conditions for independence help to locate the reasons for unexpected errors.

It is important to keep in mind that the dependence of a concrete action and its result to other preceding or concurrent actions (and their results) is a different issue than the question whether even performing an action is possible when the model is in a particular state. For example, the requirement of being able to request a contact exchange in Skype at any time without affecting the result of other actions is a concern of mutual independence of actions — whereas the requirement that there can only exist one pending request from one user to another one is state-based (where we assume that the possible actions of accepting and denying the request are represented by the Petri net structure). For defective models, combining the validation techniques we develop in the following can be even more informative: If by validating we detect an inconsistent model state with two concurring contact requests after a contact exchange has been requested twice, we possibly can examine the dependencies of these two actions to find out the error in the model and to correct it so that the second action is discarded appropriately, leading to a consistent state.

3.2.2. High-level Modeling of Communication Platforms

The properties from Sect. 3.2.1 focus on the users as the most important parts in a platform model and the actions they can possibly perform. This suggest to consider the possible actions of users and their effects in the platform as the most important factors for modeling and validation. In order to model a concrete Communication Platform according to the description in Sect. 2.5 and still to be able to formulate and validate these properties, the modeling approach and method for Communication Platforms should concentrate on this particular level of abstraction.

The authors of [RSV97]

> believe that it is possible to clearly identify what role communication plays at all levels of the design and keep it separate from component behavior. It is our intention to show how the design of an electronic system can be carried out into two almost independent steps: the design and selection of the functionality of the components of the design and the way in which these components interact through a communication mechanism. In particular, we believe that a specific methodology can be used to carry out the design of communications in a top-down, constraint-driven fashion.

Their basic idea of "interface-based design" is to develop a methodology to uncouple communication (via interfaces) from the other features of a system with the goal of orthogonalizing electronic design. However, the focus lies on modeling of Asynchronous Transfer Mode (ATM) networks as large electric systems and on refining models of communication — in this context they mean mere data transmission — to implementations as circuits. Instead, we are interested in the other side and regard communication as acting of users that has the effect of exchange of data. The most important aspect of this viewpoint is to possibly distinguish user actions (as triggering events) from their effects. So the concept of uncoupling has a lot in common with what we denote as our intention of "high-level behavior modeling" (cf. Sect. 2.4.1), for which we want to concentrate on the modeling of possible user behavior independently from some underlying data transmission mechanism, which we just presume to be available. Also, we are not interested how users can actually access data like histories via the user interfaces — e. g., in order to read them — as long as we can assume that this data is stored in places that can be distinguished clearly to belong to a user; and we do not regard the mere look-up and visual output of data as actions we want to model because they have no noticeable effect on the system.

In short, our model must allow us to model the users' actions and their effects in the system such as data transmission or structure manipulation, both clearly distinguishable from each other. We want to describe the effects in a way that corresponds on the abstraction level approximately to an interface of an underlying reactive system as realization of this merely executive system behavior. This is justified by our assumption for Communication Platforms that all occurring effects are caused by actions of the users.

3.2.3. Related Work to Modeling of Skype and Communication Platforms

In Sect. 2.4.2, we discuss approaches that might be appropriate for a first step towards the modeling of Communication Spaces, especially some extensions to Petri nets supporting dynamic structures. Here, we look at some related work that (partially) covers our intention by modeling communications in dynamic systems or from a high-level view on user interaction or that deal directly with our concrete choice of Skype as a representative.

High-level communications The users in our high-level models are close to the concept of "peers" in Peer-to-Peer networks (P2P), where a peer is informally defined as an entity with capabilities similar to other entities in the system. According to [MKL$^+$02], the concept of P2P has as one of its main goals "dynamism" in form of ad-hoc connectivity. Further, they state that "when an application is intended to support a highly dynamic environment, the P2P approach is a natural fit" and the notion of P2P can be seen as a mind set that supports direct interaction among its users, which is convenient for Communication Platforms. All these points suggest P2P as an interesting notion related to Communication Platforms, but P2P is too specific for our needs because it assumes that the clients are themselves the nodes of the communication network, which we do not have in the other (server-based) examples of Facebook and Second Life. Hence, we cannot solely rely on P2P as a fundamental idea for Communication Platforms.

The approach of scenario-based programming with life sequence charts (LSC) of [HM03] fits very well to the idea of specifying a system by describing the possible user actions in an interface and their resulting perceivable effects. The LSC describe the possible behavior and interaction of objects as instances of classes. A great advantage of this approach is that it comes with an implementation called the "Play engine" that allows the designer to specify scenarios as behavior by "playing in" events for (graphical) interface objects effects related to classes of a class model and then to "play out" this specification on concrete object instance models. Unfortunately, up to now the Play engine is not (yet) capable to handle dynamic creation and deletion of objects, although LSCs support binding of instances with object properties. So, the number of objects is fixed per specification, which is too restrictive for our needs.

[HK99, Nar02] both are about "modeling user behavior", although in a different sense than ours. They are mainly interested in quantitative system properties that emerge from observable action occurrences such as workload or dynamics of user groups. Here, modeling of user behavior does not mean to have a possibility to design and specify user behavior (as a model in the sense of an instance of some meta model) but to just assume behavior to happen somehow so that it should be examined in order to get to a descriptive model of it.

Skype-related work A great portion of the scientific literature concerning Skype is about analyzing its communication protocols and encryptions, such as e. g., [BS04, DK06, ZZZY10]. Skype's communication protocol and source code are undisclosed and proprietary without available — at least official — documentation, which makes Skype an interesting object for research in this fields, because they seem to work properly.

One of the slightly more abstract works is [KHTR10], which models the P2P network of Skype nodes with attributed typed graphs. New Skype nodes can be created, connected, and loaded with traffic in form of a numerical value by applying transformation rules. Although this work considers dynamic structures and model simulation via application of transformation rules (which works very similar for reconfigurable Petri nets), the main focus is still on quantitative network properties and not on modeling communication behavior for Skype users as we intend to do.

3.2.4. Towards Modeling Communication Platforms with Reconfigurable Petri Nets

Regarding our choice of reconfigurable Petri nets from Sect. 2.4.3 as a formal technique for modeling Communication Spaces — and Platforms — together with the viewpoint of high-level modeling, we now identify and characterize the main parts of a Communication Space (cf. Sect. 2.3) in Skype and how they are related to Petri nets. For this discussion, we need an understanding of behavior of Petri nets and their reconfiguration. There might be some special requirements that we have to approach later by developing a new variant of Petri nets and going into the formal details, but for now only the informal basics are necessary, which apply to every model that is considered as a Petri net.

Reconfigurable Petri nets in a nutshell A Petri net is a graph-like structure with two kinds of nodes: The *places* of a net can carry *tokens* and all tokens together are called the *marking* of the net. The marking represents a state of the net and we say that a place p contains a number x of tokens in a certain marking. The behavior of a net is described with the other kind of nodes, *transitions* that have incoming and outgoing *arcs* to places.[7] A *firing* of a transition stands for some action that occurs and its effect is specified by the arcs describing the change of the nets marking (i. e., its state) when firing this transition. A transition can only fire if the tokens on the places of the incoming arcs satisfy a firing condition related to labels on the arcs. For example, in simple Petri nets the arcs are labeled with numbers that state how many tokens have to be present on the places connected to the incoming arcs. A firing transition then *consumes* this number of tokens and *produces* a set of tokens that result from the labels on the outgoing arcs on the corresponding places. In extended Petri net variants such as algebraic high-level nets, the tokens can stand for an algebraic value and the arcs of the transition can be equipped with terms and variables that — besides the number of consumed and produced tokens — only allow specific token values on the places to enable a transition for firing.[8]

We call the manipulation of a net's structure — i. e., the set of its places, transitions and connecting arcs — and possibly of its marking as reconfiguration. Our favored formalism for specifying reconfigurations are transformation rules for high-level structures as graphs or Petri nets, which are similar to the rules of Chomsky grammars to define

[7] The nature of a net's nodes are commonly reflected by their names, i. e., countable nouns or predicates for places and verbs for transitions.

[8] Examples for firings steps of place–transition nets and algebraic high-level nets with individual tokens can be found in Example 2 and Example 7.

formal (textual) languages: They have a left-hand side (LHS) for matching a subpart of a graph, a net, or a word and right-hand side (RHS) that replaces the occurrence in the target structure matched by the LHS. The details of rule application, i. e., the possible matching and the replacement of elements within the correct context, depends on the concrete transformation approach.[9] For the following discussion, it is sufficient to understand the general concept of rule-based reconfiguration.

We now bring the concepts of reconfigurable Petri nets together with Communication Platforms represented by the concrete example Skype:

Actors are all the registered users in Skype, whether they are active or disconnected. Note that we restricted by assumption the behavior of a user to actions he can perform in its client and distinguish them from the possible effects if needed. Therefore, in the following we can simply identify a Skype client with the user who operates it. It is a natural choice to regard the possible behavior of an actor/client/user as firing of transitions in a Petri net, so that the actor is represented in a Petri net by a set of transitions that implicitly are assigned to it. As we assume to use high-level Petri nets with data values as tokens representing communication and other data, firing of transitions can manipulate these values on the places adjacent to the transitions and causing immediate effects.

In Skype, we can observe that — besides communication in form of data transmission — the effects of clients' (inter)actions are limited to their realms, i. e., a client cannot directly manipulate data in another client but only animate him to do so, e. g., by inviting him to accept a contact exchange request (cf. "Contact Management", Sect. 3.1).

Channels represent means of communication between actors, i. e., allowing users to perform actions that cause the transmission of data. A simple example is the exchange of identities for contact lists, where the asked user has the two options of accepting or denying the request for contact exchange of which the first would result in adding entries to both users' contact lists.

From our assumption of transitions as action representations and the fact that in conferences and group chats a user typically sends a message only once and assumes that all participants receive it, we have to consider channels not only as one-to-one connections but rather as variable structures with multiple actors connected to it, similar to hyperedges in hypergraphs.

Context of a channels and the data sent through them in Skype can be understood as the channel's state together with its intention or function. A state of a channel as a Petri net structure whose transitions provide actions to the participants can represented in two complementary ways: Implicitly, the intention of a channel — being a conference or a contact exchange — is already represented by the structure

[9] For graphs, several rule-based transformation approaches have been developed [Roz97], of which especially the abstract categorical approaches seem interesting. In Chap. 4, we instantiate such an abstract transformation approach with a suitable category of Petri nets in order to benefit from the properties that are already proven for the abstract framework.

and the possible actions for the participants. In addition, we need some explicit context representation, e. g., for the history of the sent messages in a chat or in order to distinguish active participants in a conference from users that are invited but have not yet joined and still cannot "hear" what the active participants are saying. For the explicit part, we can make use of the algebraic data types and store it in data tokens on dedicated data places.

This idea of context suggests that channels are not only structural features to transport data but also are useful as representation of idle communication that has a traceable history and can be continued later. At first sight, Fig. 2.2 might give a different intuitive idea about channels, but we do not assume all communication between two actors necessarily being handled or represented by just one channel. This figure merely shows a snapshot where a particular set of data is currently being transfered. Even the arrows on the channel arcs should not let us blindly assume that channels have a designated direction or even are strictly unidirectional. Unidirectional channels only transmit data in one direction, so that true conversational communication would require two such channels in each direction between each participant, which makes modeling and handling of group-like communication much and unnecessarily harder.

Content spaces, access rights, and roles and their representation in Petri nets mostly follow from the above. The *actual* content space of a user is the entirety of token data on places assigned to it such as user names, communication histories, contact and black lists etc. We can also identify a *potential* content space of the users that is defined by the algebra domains of the token sorts from which users may generate data tokens by appropriate transition firing. Access rights and roles are modeled implicitly by the structure of the client nets and the preconditions and effects of their transitions.

Dynamics of actor sets and channels — as an important point in Communication Spaces and Platforms — can obviously be realized by application of reconfiguration rules; but we also have to take into consideration that we assume the actors' actions to be the only cause of effects in Communication Platforms. This means for our Petri nets that transition firings on the one hand have to be a necessary precondition for possible rule applications and on the other hand should also be sufficient in the sense that they cause a reconfiguration by rule application to happen as soon as possible.

It harmonizes with our aim of high-level modeling that we strictly distinguish the acting of users in their clients as transition firing from reconfiguration realizing necessary dynamic effects. Note that action occurrences are just expressing a user's intentions to achieve an effect and should be handled as such in some way by the system, either in simple cases as direct effects of transition firings or by triggered rule-based reconfiguration resulting in more complex and dynamic effects as channel creation or extension.

3.2.5. Data Transmission in Petri Nets

In communication networks like the Internet, there are several common modes of send one–receive many network transmission mechanisms (cf. [Los03]). We shortly discuss them and their relevance for modeling of Communication Platforms and transmission of data with reconfigurable Petri nets following the principles given in Sect. 3.2.4.

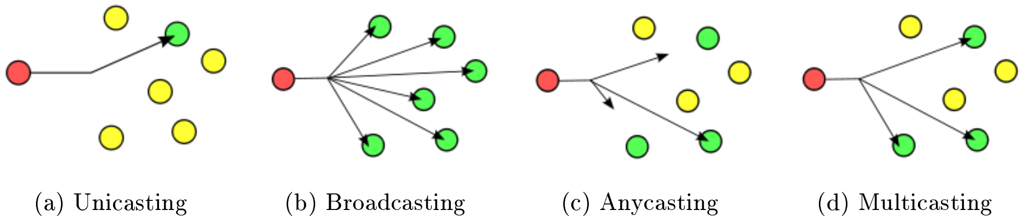

(a) Unicasting (b) Broadcasting (c) Anycasting (d) Multicasting

Figure 3.10.: Schemes for data distribution in networks

Unicasting is the transmission of data to one defined receiver (Fig. 3.10a). In Skype and Communication Platforms, this mode is essential because every direct communication to another user such as plain calls or text messages corresponds to this principle. In Petri nets, this kind of transmission can be made possible by reconfiguring the platform net such that for each known contact of a client a transition is created that allows this client to send data to its contacts (e. g., to some input buffer place of the contact's client part) by firing it.

Broadcasting of a message let all other entities in the network receive it (Fig. 3.10b). This data transmission mode is rarely found in Communication Platforms because it overrides any possibility of a user to decide whether he wants to receive data from some other users at all, which does not fit our idea of Communication Platforms. In fact, the concrete typical examples for Communication Platforms we consider in Sect. 2.5 do not feature message broadcasting for regular users. Unless there is some super user such as the administrator of a web forum who wants to distribute global announcements to all forum users, we can ignore this transmission mode. It is more likely that a user wants to perform a "personal broadcast", i. e., sending data to all of his known contacts instead of all users that in the whole platform, for which the multicasting mode described below is more appropriate.

Anycasting describes the method of dispatching a message to the system such that any other user — or more precisely, exactly one other user, usually the "nearest" in some sense — receives it (Fig. 3.10c). This mode is even more inappropriate for Communication Platforms because typically a user chooses the recipient(s) of sent data before sending it so that we do not regard this mode further.

Multicasting could be characterized as a multiple unicast or selective broadcast, i. e., a defined set of receivers get the transmitted data (Fig. 3.10d). In addition to unicasting, multicasting is the other main mode of data transmission in Communication Platforms and is needed to realize important features such as conferences or multiuser chats in groups.

We see that for modeling Communication Platforms, we can concentrate on uni- and multicasting of data. Whereas unicasting seems to be quite natural to be modeled in reconfigurable Petri nets by using rules to create transitions whose firings represent the transmission of data along some existing channel, multicasting is not that intuitive to achieve.

3.2.5.1. Modeling Multicasting in Petri Nets

In the discussion in Sect. 3.2.4 about representing actors of Communication Platforms in reconfigurable Petri nets, we choose to represent the actors' actions exclusively by transitions and to reconfigure the platform net — i. e., deleting or creating transitions — if some user suddenly has more possible actions available as a consequence to a previous action. A central characteristic of Communication Platforms is that the set of actors is dynamic, which we can approach by using reconfigurations as well to create new client components (consisting of transitions for the new user's actions) in the platform net. It is an important question how to model the effect of data multicasting in such a highly dynamic Petri net, probably by reconfiguration, too.

Related work on modeling multicasting There exist several publications that are about modeling multicasting to some extent with or for Petri nets. In [PT07], an approach for modeling multicasting with high-level Petri nets is presented. Unfortunately, the proposed approach does not fit to our intention of high-level modeling and multicasting is achieved by a complex structure of routing transitions. In our models of Communication Platforms, we want to abstract from such details and simply model the outcome of a multicasting action that is triggered by a single transitions firing by the user/client who sends the data with this firing. Therefore, routing of multicasted data with several necessary firing steps that are not related to a certain user action is not an adequate representation. Moreover, in [PT07] the multicasting nets are assumed to be immutable and because of the complexity of the routing nets is not obvious how to reconfigure them for e. g., introducing additional users into the systems and allowing them to perform multicast operations. For similar reasons, also the Petri net based model of a multicasting protocol in [LL95] is not adequate for our intentions. Other publications that are interested in other network types rather than Petri nets cannot help us either because they are mainly focused on other levels of abstraction such as [JGJ+00] that aims at improving the bandwidth of data distribution by path optimization with the concept of an overlay network.

Even if we decide to model multicasting in some way by using rule-based reconfiguration there are some pitfalls we have to avoid. Intuitively, we could simply try to add transitions to the platform net by reconfigurations as for unicasting. Following this idea, we would need for every possible multicasting a single transition. In a platform with n users this would result in $2^n - 1$ transitions for each user, one for every subset of all possible users that can receive a message via multicasting. Although this construction already leads to very complex nets, the complexity of the reconfigurations needed for adding further users properly to this heavily connected structure is even higher, and we have not even considered the cases that some recipients may be temporarily unavailable (such as *offline* in Skype) to receive messages and how to model this appropriately. In this case, the net must provide a transition for each possible combination of actively participating users that actually can receive something in a conference and this complex structure would need to be reconfigured excessively on the slightest manipulation of the channel.

To avoid this complexity explosion, the next idea would be to simulate multicasting by multiple unicasting, i. e., to split the transmission to a set of actors into single transmissions to each receiving actor. This makes it necessary to code the status of the multicast data for each actor into the net (cf. [LL95]). Still, each actor must be connected to every other actor in order to be able to perform multicasting but the main problem is that this approach is not compatible to our idea of a transition representing a user's action. If we modeled a conference of arbitrary size with multiple unicasting, a user who wants to be heard by all participants would need to fire the corresponding transition — i. e., repeat to say his message — for every other user in the conference. This method lacks the atomicity we assume a multicasting action to have.

From this discussion, we see that modeling the communication behavior of multicasting is a challenge for — even reconfigurable — Petri net-based modeling techniques if the number of actors is not known a priori. The main reason for the discussed problems comes from the assumption of transitions whose firing have the final effect of multicasting some data tokens. Instead, we can make use of the separation of user actions and system reactions and consider the user action as just requesting the transmission of data in the current context of the channel that determines the active participants. Now, the problem of appropriately distributing the data is shifted to modeling the effects of the system reaction to the user request, and we have to find a way to specify such effects by rule-based reconfiguration. In [BEE$^+$09], we propose another modeling approach that uses transitions to create tokens announcing a request for transmission of data to a selected set of other users. In other words, a single transition models the action of dispatching some data and the application of a rule copies and moves the data to the recipients that are identified by some net structure (possibly connection of an actor component to a conference) or identifier tokens (on a place representing a conference, cf. [BEE$^+$09]) in the platform net. With this technique, we can describe multicasting that is first triggered by firing of a transition and then executed by application of some rule moving the data accordingly.

3.2.6. Modeling Principles

Based on the previous considerations, we can summarize some concrete modeling principles and reasonable guide lines for Communication Platforms as reconfigurable high-level Petri nets with the example Skype in mind:

1. The structure of the platform is modeled by a single net and the users/clients are

represented by components of this net, consisting of the transitions for the user's actions. There is no reason for keeping the client components apart in separate nets; moreover, it is more convenient to have all places in one net in order to connect the clients via channels in form of net structures.

2. User identities are represented by unique data values of a specific user name domain on a dedicated place in the user client.

3. All direct user actions are represented by firings of transitions. Whether an action can be performed by a client in its current state is reflected by the preconditions of the corresponding transition. As such actions, we regard everything a user can possibly do via the user interface of the platform that affects the platform.

   There is an exception for this principle: We are only interested in actions actually having some effect on other actions and not just on the user interface. A prominent example for such a “indifferent” action is to set a state such as “away” or “do not disturb” that merely affects the way that incoming calls or messages are announced in the interface, e. g., by simply suppressing the ring tone. Obviously, if we modeled such transitions, their firing possibly would just change some markings that we would not use anywhere else in the model because following the idea of high-level modeling from Sect. 3.2.2, we aim for abstracting from the concrete interface.

4. Every transition has to be implicitly assigned to the one user who may trigger the represented action. In most cases, an action depends on its invoking user's data, i. e., data tokens on adjacent places. By this principle, we get that the transitions in the net can be assigned implicitly to the users that are supposed to own — or in other words to trigger — them if we can pinpoint a core part of a user's client component with the unique data value representing the user identity. But a transition may also depend on data of multiple users. Therefore, we demand structural features in the model such as special arc inscriptions that allow us to clearly infer the owning user.

5. All system effects have their prime cause in transition firings, either if they are achieved directly by firing transitions such as the manipulation of data tokens or are realized by the system by applying reconfiguring transformation rules (as a “black box”). In the latter case, the transition firing can be interpreted as a request for a system reaction. Typical examples are cases where reconfiguration is necessary, i. e., actions that extend or abridge the possible behavior of a user such as creation of conferences.

6. It is advisable to keep in mind the context of an action and provide transitions accordingly. For example, there is no need for a single transition “leave group” in the client and it is a better choice to provide such a transition for each group with the corresponding effects.

7. For the sake of simplicity, we can further assume that actions in the graphical user interface (e. g., of Skype) sometimes are shortcuts for sequences of other own dedicated actions the model provides. For example, calling a set of contacts is the

same as calling one person and immediately adding other users to the conversation to form a group. We may refrain from modeling the complex compound actions explicitly if we can realize them by combining simpler ones that we model anyway; but we must avoid inconsistent model configurations (cf. Sect. 7.1.4)! Also note that the composed actions nevertheless have to belong to the executing user!

8. After actions have become "dead", i. e., their owning user will never perform them because they are not longer available in the user interface, the corresponding transitions should be deleted appropriately. The typical example is exchange of contact data, where affirmation and denying of the asked user can be interpreted as a request to the system for removing the two transitions representing these actions, which are not needed anymore.

The case study in Chap. 7 is based on these principles, refines and concretizes them, and demonstrates how to model specific features with transformation system of reconfigurable high-level Petri nets that we define in the following chapter.

3.2.7. Particular Resulting Challenges for a Concrete Approach of Reconfigurable High-Level Petri Nets

A a bridge to the next chapters, we point out and summarize some main issues to focus on for the development of a concrete formal approach of reconfigurable high-level Petri nets that conforms to the considerations of this chapter.

Structure-dependent interaction In Sect. 3.2.5.1, we discuss the unmanageable complexity of modeling multicasting in communication systems by reconfigurable Petri nets. We can make use of the separation of user actions and system reactions and consider the user action as just requesting the transmission of data in the current context of the channel that determines the active participants. Now, the problem of appropriately distributing the data is shifted to modeling the effects of the system reaction to the user request, for which we use the method based on the application of special rules that we have outlined in [BEE^{+}09]. We approach this problem with an adequate definition of Communciation Platform models and user actions in Sect. 5.3.

Reconfiguration of markings Channels are highly dynamic and therefore we have to expect frequent reconfiguration of the Petri net structure of a channel representing a conference or a chat as users are added or leave. We have to face even more complicated details as users can be in different states when participating in a conference, e. g., waiting to join the conversation or having hung up already.

From this and the previous point about data transmission by reconfiguration, we can see the necessity for a formal approach of reconfiguration rules that can manipulate the markings of a net independently from the structure. This might seem obvious but Chap. 4 brings out that previous approaches for Petri net transformation may not allow us to formulate reconfiguration as freely as we need it, so we state this here explicitly.

In addition to the power of expressing arbitrary marking changes by rules, we also can close the conceptual gap between transition firing and reconfiguration with the result of simulating firing steps with reconfiguration (see Sect. 4.2.3), which is another point in favor of this requirement especially with regard to validation of model properties.

Formulation and validation of model properties In Sect. 3.1.7 and Sect. 3.2.1, we describe some properties of models of Communication Platforms we are interested in, namely questions about model integrity and independence of actions.

For the integrity, we develop in Sect. 6.1 means to express the conditions, e. g., for unique user names or correct role constellations in certain communication contexts w. r. t. the concrete platform to be modeled. This representation of the conditions has to be checked against the actual platform model configuration in order to validate the properties after every configuration change.

For the independence of actions, we can rely on existing formal notions of independence for both firings in Petri nets and rule-based reconfigurations. But if we want to relate firing steps with reconfigurations in models for Communication Platforms, we may also need a formal notion of dependency between firing steps and reconfigurations. These independence notions combined in Sect. 6.3 allow to thoroughly check the independence of all possible configuration changes in models of our integrated modeling approach for Communication Platforms.

Control of rule applications It may be useful for formal analysis and model validation to simulate firing steps with rule applications, but in particular for the modeling of communication features, we want to relate certain firing steps representing the user actions with the system reactions and effects in form of application of corresponding rules.

For this purpose we define a (also Petri net-based) control structure mediating between Petri net behavior and reconfiguration in Sect. 5.4. Otherwise, we would simply have to assume that a Communication Platform reacts at some unspecified time after a user action without being able to force any reaction ever to happen.

One may argue that this problem comes with our choice of rule-based reconfiguration of Petri nets because other approaches for mutable Petri nets (cf. Sect. 2.4.2) bind their reconfiguration directly to firing steps. But these other approaches' strict dependency of reconfiguration and firing either restricts reconfiguration to the environment of a transition or alters the firing behavior considerably. The first restriction demands a locality of reconfigurations that is not adequate for our purposes of modeling and the second one makes it even harder to analyze or validate interesting properties for Communication Platforms. Therefore, it seems to be advisable not to alter the semantics of firing steps and reconfigurations so that they depend on each other too strictly in order to possibly be able to use techniques for validation in an adequate way on the one hand. On the other hand, we need some formalism to express our intention of relating firing steps to triggered reconfigurations and to have a suitable semantics for this formalism that complies to our modeling principles.

4. Transformation of Petri Nets with Individual Tokens

> I hope that it sums up the way that I feel to your satisfaction [...]
> Believe me when I say that I cannot apologize enough
> When all you ever wanted from me was a token of my love
>
> *(Lily Allen)*

In Sect. 2.4.3, we decide to consider a kind of reconfigurable Petri nets with high-level algebraic tokens as characteristics of a formal approach for modeling of Communication Platforms according to the principles we have developed in Sect. 3.2.6. A theory of rule-based transformation based on double pushout (DPO) graph transformation [EEPT06] is already available for place/transition (P/T) systems, i. e., P/T nets with an initial marking, and also for algebraic high-level (AHL) systems that have data elements of an algebra [EM85] as tokens rather than low-level "black tokens" [PEL08]. This exploits the graph-like structure of Petri nets and allows us to formulate rules to change the structure of a Petri net. Because we use the formal categorical framework of *weak adhesive high-level replacement categories* as a fundamental basis, we assume the reader to be familiar with the concepts used in [EEPT06] like basic category theory, double pushout transformations, signatures and algebras etc., in order to be able to follow the formal parts in the rest of this thesis.

P/T and AHL systems have been shown to form a weak adhesive HLR category with a class of marking-strict morphisms [EHP^{+}07, PEHP08, Pra08]. This allows us to apply all the results for $\mathcal{M}$-adhesive transformation systems concerning the local Church-Rosser property, parallelism, and concurrency of transformations as shown in [EEPT06] also to transformation systems of Petri systems, which are preferable classic properties for transformation systems. In the following, we use the notion of $\mathcal{M}$-adhesive categories, which supersedes the former notion of "weak adhesive HLR category".

Unfortunately, by using just marking-strict morphisms, we cannot formulate transformation rules for Petri systems that change markings without some restrictions, which is inconvenient in terms of usability and intuitiveness of the transformation approach.[1] Marking-changing rules are essential for modeling communication systems and platforms with Petri nets, especially for realizing multicasting of data tokens in high-level nets, as we discuss in detail in Sect. 3.2.5.1. The main technical reason for this problem is that the pushout of elements from a monoid is not their sum but their supremum.

[1] To be precise, one can formulate such rules, but the results of their applications may not be unique. In order to change the marking of a place properly, this place has to be deleted and recreated by the rule, which may bring up problems caused by adjacent transitions in the unmatched context.

To overcome this restriction, we present a new Petri net formalism based on P/T nets and AHL nets that we call “Petri nets with individual tokens”, together with a rule-based transformation approach that is based on the “collective” one (with monoid markings) in [PEHP08]. The pivotal difference concerns the representation of net markings: for the new individual approach, we propose a net’s marking as a set of individuals instead of a (collective) sum of a monoid. Nevertheless, the formal definition of the net’s structure remains the same so that nets with individual tokens still follow the concept “Petri nets are monoids” from [MM90]. We further present $\mathcal{M}$-adhesive system for Petri nets with individual tokens in Sect. 4.3.

In Sect. 4.1, we begin with the definition of P/T nets with individual tokens (PTI nets) and define rule-based PTI transformation as we presented it (with finite markings) in [MGH11]. As a main result, we show the equivalence of transition firing steps and the application of firing-simulating rules. With this result, we have an answer to the challenge of reconfiguration of markings for our modeling approach that we summarized in Sect. 3.2.7. In Sect. 4.2, we lift the definitions and results for PTI nets to AHL nets with individual tokens (AHLI nets). In Sect. 4.3, we instantiate the categorical high-level transformation framework of $\mathcal{M}$-adhesive transformation systems to PTI and AHLI nets. Finally, we relate both kinds of individual Petri systems with their collective counterparts and with each other by a notion of behavioral firing equivalence in Sect. 4.4 and by functors in Sect. 4.5.

On the Notion of Individual Tokens

The term of *individual tokens* has been used in two senses for Petri nets so far:

1. It was mentioned first in [Rei85b], where it was used to describe “tokens that can be identified as individual objects”. The main contribution of this article was to define markings as multisets of distinguished elements rather than amounts of indistinguishable black tokens. In the end, *individual tokens* in this context is a synonym for what by now is called *high-level tokens*. This is not our intended meaning of individuality, moreover we will consider both low-level and high-level Petri nets with individual tokens.

2. The other notion of token individuality has been coined in [GP95] [2] as “individual token interpretation” of firing steps, which entitles the definition of processes in [GR83].

   In [Gla05], the author investigates the collective-individual dichotomy of firing steps, where under the individual approach firing sequences consider not only the number and value of tokens (as in the collective approach) but also the history — i. e., the origin transition — of tokens. [BMMS99] introduces a category of Petri nets to be interpreted with a functorial individual semantics.

The following approach of Petri nets with individual tokens is related to 2., but in contrast, we will already introduce individual tokens in the (syntactical) definition of the

[2] Meanwhile, there is an updated version of this article: [GP09].

Petri nets (and their category) themselves, so that we can exploit the individuality of tokens in the transformation approach, independently of the firing rule. The individual firing rule of [Gla05] is very close to the one presented in the next sections, but it is still an interpretation of firing steps of a classic representation of Petri nets.

4.1. Place–Transition Nets with Individual Tokens

In this section we treat P/T nets and equip them with markings of individual tokens. After describing their firing behavior for individual markings and introducing rules for the transformation of marked nets we show how special transformations correspond to transition firing steps.

4.1.1. Definition and Firing Behavior

With the definition of nets with individual tokens, we follow the concept "Petri nets are monoids" from [MM90]:

Definition 4.1.1 (Place–Transition Nets with Individual Tokens)
We define a marked P/T net with individual tokens, short PTI net, as

$$NI = (P, T, pre, post, I, m), \text{ where}$$

- $N = (P, T,\ pre, post\colon T \to P^{\oplus})$ is a classical P/T net, where $P^{\oplus}$ is the free commutative monoid over P,
- I is the (possibly infinite) set of individual tokens of NI, and
- $m\colon I \to P$ is the marking function, assigning the individual tokens to the places.

Further, we denote the environment set of a transition $t \in T$ as $ENV(t) = PRE(t) \cup POST(t) \subseteq P$ with

$$\begin{aligned} PRE(t) &= \{p \in P \mid pre(t)(p) \neq 0\}, \\ POST(t) &= \{p \in P \mid post(t)(p) \neq 0\} \end{aligned} \qquad \triangle$$

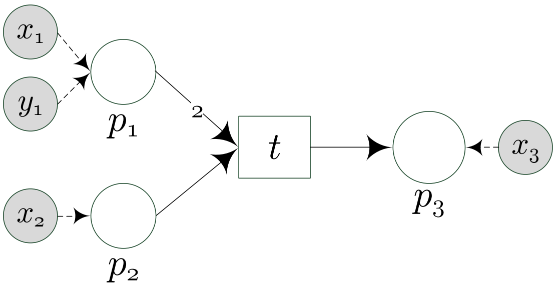


Figure 4.1.: Example PTI net

Example. Fig. 4.1 shows the graphical representation of a PTI net with individual marking $I = \{x_1, y_1, x_2, x_3\}$, $m(x_1) = m(y_1) = p_1$, $m(x_2) = p_2$, $m(x_3) = p_3$.

Now, we define the behavior of the PTI nets as firing steps. Because of the individual tokens, we have to consider a possible firing step in the context of a selection of tokens.

Definition 4.1.2 (Firing of PTI Nets)
A transition $t \in T$ in a PTI net

$$NI = (P, T, pre, post, I, m)$$

is *enabled* under a *token selection* (M, m, N, n), where

- $M \subseteq I$,
- m is the token mapping of NI,
- N is a set with $(I \setminus M) \cap N = \emptyset$,
- $n \colon N \to P$ is a function,

if it meets the following *token selection condition*:

$$\left[\sum_{i \in M} m(i) = pre(t)\right] \wedge \left[\sum_{i \in N} n(i) = post(t)\right]$$

If an enabled t fires, the follower marking (I', m') is given by

$$I' = (I \setminus M) \cup N, \quad m' \colon I' \to P \text{ with } m'(x) = \begin{cases} m(x), & x \in I \setminus M \\ n(x), & x \in N \end{cases}$$

leading to $NI' = (P, T, pre, post, I', m')$ as the new PTI net in the *firing step* $NI \overset{t}{\rightarrowtail} NI'$ via (M, m, N, n). △

Example. We demonstrate a firing step in a net that models a minimalistic communication client in Fig. 4.2, which can be in the states online/offline and idle/busy represented by the corresponding places. The client can be activated and deactivated, start a call, and hang up an ongoing call by firing the corresponding transitions. With the marking $I = \{i_1, i_2, i_3\}$, $m(i_1) = m(i_2) =$ `Online`, $m(i_2) =$ `Idle`, we consider the client to be online but not in a call.[3]

Firing of PTI nets is similar to firing of collective P/T nets with the difference that in PTI nets, we have to select the concrete individual tokens to be consumed and produced by the firing step. If the client net had a collective marking, obviously the transition *call* could fire because there are sufficient tokens on its predomain places. Place `Online` even carries one token more than *call* demands for firing so that we expect to find more than one selection for firing *call* with this individual marking.

[3] Note that we have two tokens on place `Online` and transition *call* consumes a token from this place. The transition *call* already depends on place `Idle` to be marked, but we accept this redundancy for the sake of demonstrating the selection of tokens.

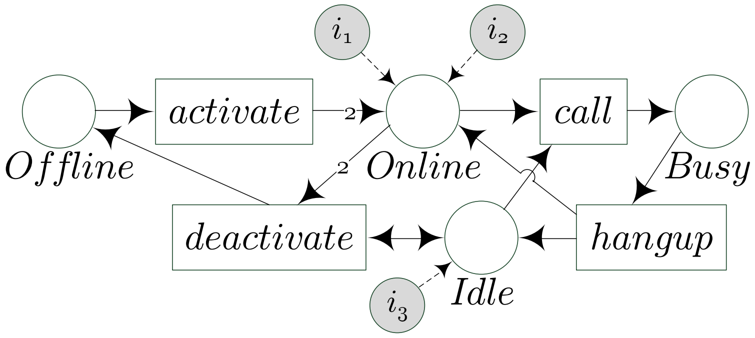


Figure 4.2.: Example client PTI net

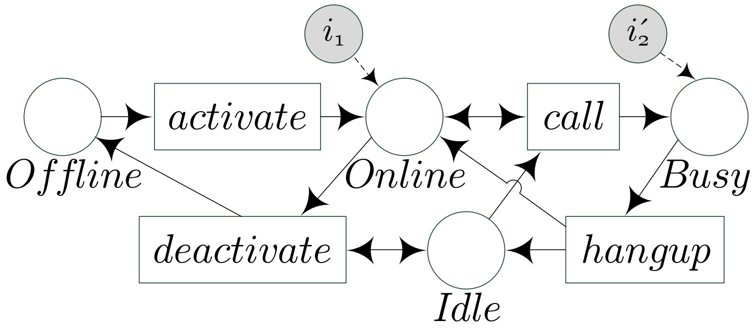


Figure 4.3.: Example client net after firing transition *call*

Consider the token selection $S = (M, m, N, n)$ with $M = \{i_2, i_3\}$, m as the current client net marking, $N = \{i_2'\}$, and $n(i_2') = \texttt{Busy}$. Clearly, S is a valid token selection because $M \subseteq I$ and the sets $I \setminus M$ and N are disjoint. The transition *call* is activated under S because the token selection condition is satisfied:

$$\sum_{i \in M} m(i) = m(i_2) \oplus m(i_3) = \texttt{Online} \oplus \texttt{Idle} = \mathit{pre}(\mathit{call})$$

$$\sum_{i \in N} n(i) = n(i_2') = \texttt{Busy} = \mathit{post}(\mathit{call})$$

The firing step of *call* via S leads to the follower marking $I' = \{i_1, i_2'\}$, $m'(i_1) = m(i_1) = \texttt{Online}$, $m'(i_2') = n(i_2') = \texttt{Busy}$ as depicted in Fig. 4.3.

Note that the transition *call* is activated also under the selection $\hat{S} = (\hat{M}, m, N, n)$ where $\hat{M} = \{i_1\}$. Alternatively, we could also fire *call* under a selection with different set of produced individual tokens, e. g., $\hat{N} = \{i_{42}\}$ and corresponding $\hat{n}$, as long as it is disjoint to the set of preserved tokens $I \setminus M$. This indicates the possible choice of available tokens to be consumed and produced that is reflected in the concrete firing step by the token selection.

Remark (Token Selection). The purpose of the token selection is to specify exactly which tokens should be consumed and produced in the firing step. Thus, $M \subseteq I$ selects the individual tokens to be consumed, and N contains the set of individual tokens to be produced. Clearly, $(I \setminus M) \cap N = \emptyset$ must hold because it is impossible to add an individual token to a net that already contains this token. The functions m and n relate the tokens to their carrying places. It would be sufficient to consider only the restriction $m_{|M}$ here or to infer it from the net, but as a compromise on symmetry and readability, we denote m in the token selection.

Note that because transition environments are finite, every transition can only be activated under finite token selections. Otherwise, if M or N is infinite, the sums in the token selection condition over the places are not even well-defined. But in the following, we consider token selections always in the context of an activated transition, so we can safely formulate well-defined sums over all elements of M or N of a valid token selection.

For the next subsection about PTI net transformation, we define a category of PTI nets with suitable morphisms and show constructions for pushouts and pullbacks in this category.

Definition 4.1.3 (PTI Net Morphisms and Category PTINets)
Given two PTI nets $NI_i = (P_i, T_i, pre_i, post_i, I_i, m_i)$, $i \in \{1, 2\}$, a PTI net morphism is a triple of functions

$$f = (f_P \colon P_1 \to P_2, f_T \colon T_1 \to T_2, f_I \colon I_1 \to I_2) \colon NI_1 \to NI_2$$

such that the following diagrams commute (componentwise for pre_i and $post_i$):

$$\begin{tikzcd}
T_1 \arrow[r, shift left, "pre_1"] \arrow[r, shift right, "post_1"'] \arrow[d, "f_T"'] \arrow[dr, phantom, "="] & P_1^{\oplus} \arrow[d, "f_P^{\oplus}"] \\
T_2 \arrow[r, shift left, "pre_2"] \arrow[r, shift right, "post_2"'] & P_2^{\oplus}
\end{tikzcd}
\qquad
\begin{tikzcd}
I_1 \arrow[r, "m_1"] \arrow[d, "f_I"'] \arrow[dr, phantom, "="] & P_1 \arrow[d, "f_P"] \\
I_2 \arrow[r, "m_2"] & P_2
\end{tikzcd}$$

or, explicitly, that $f_P^{\oplus} \circ pre_1 = pre_2 \circ f_T$, $f_P^{\oplus} \circ post_1 = post_2 \circ f_T$, $f_P \circ m_1 = m_2 \circ f_I$.

The category **PTINets** consists of all PTI nets as objects with all PTI net morphisms. △

Remark. For a general PTI net morphism f we do not require that its token component f_I be injective in order to have pushouts, pullbacks, and an $\mathcal{M}$-adhesive category. But later we may require injectivity of f_I for rule and match morphisms and to have morphisms preserving firing behavior.

Fact 4.1.4 (Construction of Pushouts in PTINets)
Pushouts in **PTINets** are constructed componentwise in **PTNets** and **Sets**. So, (1) is a PO in **PTINets** iff (2) is a PO in **PTNets** and (3) is a PO in **Sets** with the components of the **PTINets** morphisms, where $m_3 \colon I_3 \to P_3$ is induced by PO object I_3 in the commuting cube below (whose front is the PO of place sets).

$$\begin{tikzcd}
NI_0 \arrow[r, "f_1"] \arrow[d, "f_2"'] \arrow[dr, phantom, "(1)"] & NI_1 \arrow[d, "g_1"] \\
NI_2 \arrow[r, "g_2"'] & NI_3
\end{tikzcd}$$

$$\begin{tikzcd}
N_0 \arrow[r, "f'_1"] \arrow[d, "f'_2"'] \arrow[dr, phantom, "(2)"] & N_1 \arrow[d, "g'_1"] & I_0 \arrow[r, "f_{1I}"] \arrow[d, "f_{2I}"'] \arrow[dr, phantom, "(3)"] & I_1 \arrow[d, "g_{1I}"] \\
N_2 \arrow[r, "g'_2"'] & N_3 & I_2 \arrow[r, "g_{2I}"'] & I_3
\end{tikzcd}$$

$$\begin{tikzcd}
I_0 \arrow[rr, "f_{1I}"] \arrow[dr, "m_0"] \arrow[dd, "f_{2I}"'] & & I_1 \arrow[dr, "m_1"] \arrow[dd, "g_{1I}" near end] & \\
& P_0 \arrow[rr, "f_{1P}" near start] \arrow[dd, "f_{2P}" near start] & & P_1 \arrow[dd, "g_{1P}"] \\
I_2 \arrow[rr, "g_{2I}" near end] \arrow[dr, "m_2"'] & & I_3 \arrow[dr, dashed, "m_3"] & \\
& P_2 \arrow[rr, "g_{2P}"'] & & P_3
\end{tikzcd}$$

If f_{1X}, for $X \in \{P, T, I\}$, is injective then g_{2X} is as well; analogously for components of f_2 and g_1.

Proof. See appendix A.3.1.1 □

Example. Fig. 4.4 shows two pushouts in **PTINets**.

Fact 4.1.5 (Construction of Pullbacks in PTINets)
Pullbacks in **PTINets** along injective **PTINets** morphisms[4] are constructed componentwise in **PTNets** and **Sets**.

Consider a commutative square (1) in **PTINets** with g_1 having an injective net component g'_1. (1) is a PB in **PTINets** iff (2) is a PB in **PTNets** and (3) is a PB in **Sets** with the components of the **PTINets** morphisms, where $m_0 \colon I_0 \to P_0$ is induced by PB object P_0 in the commutative cube below (whose front is the PB of place sets).

$$\begin{tikzcd}
NI_0 \arrow[r, "f_1"] \arrow[d, "f_2"'] \arrow[dr, phantom, "(1)"] & NI_1 \arrow[d, "g_1"] \\
NI_2 \arrow[r, "g_2"'] & NI_3
\end{tikzcd}$$

$$\begin{tikzcd}
N_0 \arrow[r, "f'_1"] \arrow[d, "f'_2"'] \arrow[dr, phantom, "(2)"] & N_1 \arrow[d, "g'_1"] & I_0 \arrow[r, "f_{1I}"] \arrow[d, "f_{2I}"'] \arrow[dr, phantom, "(3)"] & I_1 \arrow[d, "g_{1I}"] \\
N_2 \arrow[r, "g'_2"'] & N_3 & I_2 \arrow[r, "g_{2I}"'] & I_3
\end{tikzcd}$$

$$\begin{tikzcd}
I_0 \arrow[rr, "f_{1I}"] \arrow[dr, dashed, "m_0"] \arrow[dd, "f_{2I}"'] & & I_1 \arrow[dr, "m_1"] \arrow[dd, "g_{1I}" near end] & \\
& P_0 \arrow[rr, "f_{1P}" near start] \arrow[dd, "f_{2P}" near start] & & P_1 \arrow[dd, "g_{1P}"] \\
I_2 \arrow[rr, "g_{2I}" near end] \arrow[dr, "m_2"'] & & I_3 \arrow[dr, "m_3"] & \\
& P_2 \arrow[rr, "g_{2P}"'] & & P_3
\end{tikzcd}$$

Proof. See appendix A.3.1.1 □

Example. The pushouts in Fig. 4.4 are pullbacks, too.

[4] We require morphisms that are injective in order to obtain componentwise pullbacks in **PTNets**. See [EEPT06] for details.

4.1.2. Transformation of PTI Nets

This section is about rule-based transformation of PTI nets. We use the double pushout (DPO) approach, which has also been used in [EHP+08] for P/T systems with collective markings and which stems from [EEPT06]. We characterize the applicability of a rule at some match by initial pushouts.

Definition 4.1.6 (PTI Transformation Rules)
A PTI transformation rule is a span of injective **PTINets** morphisms

$$\varrho = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R).$$

We call L the left-hand side (LHS), K the interface, and R the right-hand side (RHS) of ϱ. △

Remark. In contrast to [EHP+08], rule morphisms for PTI rules are not required to be marking-strict in order to obtain an $\mathcal{M}$-adhesive category (see Sect. 4.3). This allows us to arbitrarily change the marking of a PTI net by applying rules with corresponding places and individual tokens in L and R.

Definition 4.1.7 (PTI Transformation)
Given a PTI transformation rule $\varrho = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ and a PTI net NI_1 with a PTI net morphism $f\colon L \to NI_1$, called the match, a direct PTI net transformation $NI_1 \overset{\varrho,f}{\Longrightarrow} NI_2$ from NI_1 to the PTI net NI_2 is given by the following DPO diagram in the category **PTINets**:

$$\begin{tikzcd}
L \arrow[d, "f"'] \arrow[dr, phantom, "(\mathbf{PO_1})"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d] \arrow[dr, phantom, "(\mathbf{PO_2})"] & R \arrow[d, "f^*"] \\
NI_1 & NI_0 \arrow[l] \arrow[r] & NI_2
\end{tikzcd}$$

To be able to decide whether a rule is applicable at a certain match, we formulate a gluing condition for PTI nets, such that there exists a pushout complement of the left rule morphisms and the match if (and only if) they fulfill the gluing condition. The correctness of the gluing condition is shown via a proof on initial pushouts over matches, according to [EEPT06].

Definition 4.1.8 (Gluing Condition in PTINets)
Given PTI nets K, L, and NI and **PTINets** morphisms $l\colon K \to L$ and $f\colon L \to NI$. We define the set of identification points[5]

$$IP = IP_P \cup IP_T \cup IP_I \text{ with}$$

- $IP_P = \{x \in P_L \mid \exists x' \neq x\colon f_P(x) = f_P(x')\}$,

[5] That is, all elements in L that are mapped noninjectively by f.

- $IP_T = \{x \in T_L \mid \exists x' \neq x \colon f_T(x) = f_T(x')\}$,
- $IP_I = \{x \in I_L \mid \exists x' \neq x \colon f_I(x) = f_I(x')\}$,

the set of dangling points[6]

$$DP = DP_T \cup DP_I \text{ with}$$

- $DP_T = \{p \in P_L \mid \exists t \in T_{NI} \setminus f_T(T_L) \colon f_P(p) \in ENV(t)\}$,
- $DP_I = \{p \in P_L \mid \exists i \in I_{NI} \setminus f_I(I_L) \colon f_P(p) = m_{NI}(i)\}$, $\triangle$

and the set of gluing points[7]

$$GP = l_P(P_K) \cup l_T(T_K) \cup l_I(I_K)$$

We say that l and f satisfy the gluing condition if $IP \cup DP \subseteq GP$. Given a PTI rule $\varrho = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$, we say ϱ and f satisfy the gluing condition iff l and f satisfy the gluing condition.

$$\begin{tikzcd}
L \arrow[d, "f"'] & K \arrow[l, "l"'] \arrow[r, "r"] & R \\
NI & &
\end{tikzcd}$$

The following fact shows the correspondence between the gluing condition in **PTINets** and the categorical gluing condition (see Def. A.2.2), which is a necessary and sufficient condition for the existence of unique pushout complements (up to isomorphism) in all $\mathcal{M}$-adhesive categories (cf. Sect. 4.3).

Theorem 4.1.9 (Gluing Condition for PTI Transformation)
Given a PTI rule $\varrho = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ of an $\mathcal{M}$-adhesive system and a match $f \colon L \rightarrow NI$ into a PTI net $NI = (N, I, m \colon I \rightarrow P_{NI})$. The rule ϱ is applicable at the match f, i. e., there exists a (unique) pushout complement NI_0 in the diagram below, iff ϱ and f satisfy the gluing condition in **PTINets***.*

$$\begin{tikzcd}
L \arrow[d, "f"'] \arrow[dr, phantom, "(PO)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, dashed, "f'"] & R \\
NI & NI_0 \arrow[l, dashed] &
\end{tikzcd}$$

Proof. By Fact A.2.16 ϱ and f satisfy the gluing condition in **PTINets** if and only if ϱ and f satisfy the categorical gluing condition, which by Fact A.2.3 is a necessary and sufficient condition for the (unique) existence of the pushout complement NI_0. □

[6] That is, all places in L that are mapped to a place in NI such that the mapped place would leave a dangling arc if it was removed from NI.

[7] That is, all elements in L that have a preimage in K.

4.1.3. Correspondence of Transition Firing and Rules

An interesting aspect of the possibility to formulate marking-changing rules in **PTINets** is that rules can simulate firing steps of transitions. We give a construction for transition rules that simulate a firing step of some transition under a specific token selection and show that firing of a transition corresponds to an application of a transition rule and vice versa. With this correspondence we can easily show that token-injective **PTINets** morphisms preserve firing steps.

Definition 4.1.10 (PTI Transition Rules)
We define the *transition rule* for a transition $t \in T$, being enabled under a token selection $S = (M, m, N, n)$ in a PTI net $NI = (P, T, pre, post, I, m)$, as the rule

$$\varrho(t, S) = (L_t \overset{l}{\leftarrow} K_t \overset{r}{\rightarrow} R_t), \text{ with}$$

- the common fixed net structure $PN_t = (P_t, \{t\}, pre_t, post_t)$, where $P_t = ENV(t)$, $pre_t(t) = pre(t)$ and $post_t(t) = post(t)$,
- $L_t = (PN_t, M, m_t \colon M \to P_t)$, with $m_t(x) = m(x)$,
- $K_t = (PN_t, \emptyset,\ \emptyset \colon \emptyset \to P_t)$,
- $R_t = (PN_t, N, n_t \colon N \to P_t)$, with $n_t(x) = n(x)$,
- l, r being the obvious inclusions on the rule nets.

m_t and n_t are well-defined because t is enabled under S: The token selection condition implies that $\forall x \in M \colon m(x) \in PRE(t)$ and due to the construction of PN_t we have $PRE(t) \subseteq ENV(t) = P_t$. The argument for n_t works analogously.

Note that t is enabled under S in L_t. △

Example. The diagram in Fig. 4.4 shows the two pushouts in **PTINets** resulting from applying the PTI transition rule $\varrho(t, S) = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ to the net NI with identical token morphism component.

Theorem 4.1.11 (Correspondence of PTI Firings and Transformations)

1. *Each firing step $NI \overset{t,S}{\rightarrowtail} NI'$ via token selection $S = (M, m, N, n)$ corresponds to an induced direct transformation $NI \overset{\varrho(t,S),f}{\Longrightarrow} NI'$ via the transition rule $\varrho(t, S)$, where the match $f \colon L_{\varrho(t,S)} \to NI$ is an inclusion.*

2. *Each direct transformation $NI \overset{\varrho(t,S),f}{\Longrightarrow} NI_1$ via some transition rule $\varrho(t, S)$ with $t \in T_{NI}$, token selection $S = (M, m, N, n)$, and token-injective match $f \colon L_{\varrho(t,S)} \to NI$, implies that the transition $f_T(t)$ is enabled in NI under some token selection $\bar{S}$ with firing step $NI \overset{f_T(t),\bar{S}}{\rightarrowtail} NI^*$ such that $NI^* \cong NI_1$.*

Proof. See appendix A.3.2. □

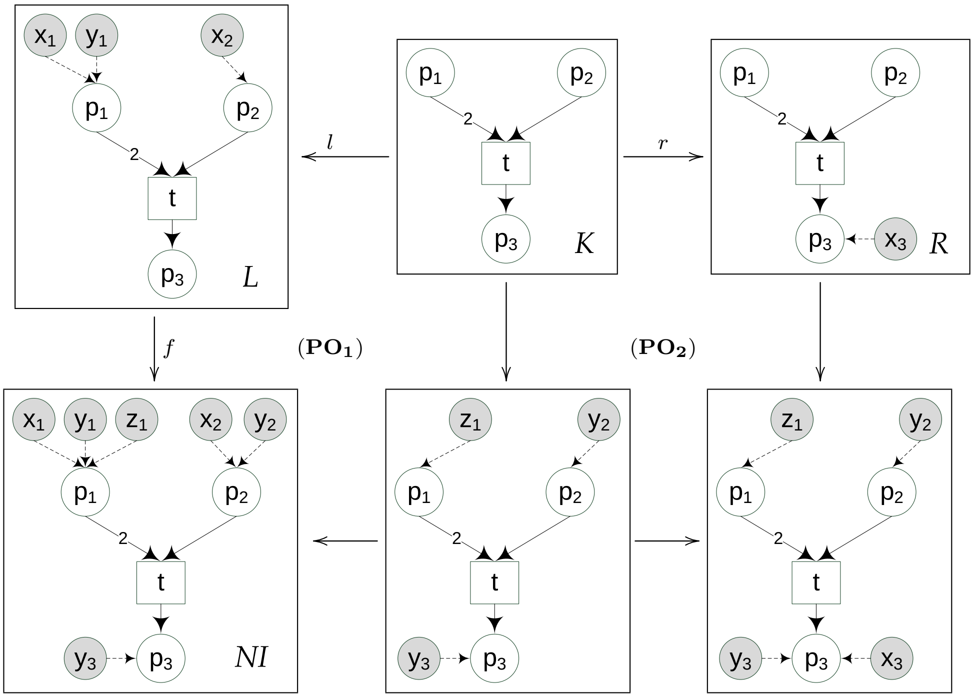


Figure 4.4.: Direct DPO transformation in **PTINets** simulating a firing step

Remark (Related approaches). The encoding of PTI transition rules and the correspondence between the firing of PTI nets and the application of transition rules stated in the theorem above are very similar to those presented in [Kre81]. A difference, however, of our encoding is the fact that the transition rules are encoded directly as PTI transformation rules rather than as graph transformation rules like in [Kre81].

A generalization of Petri nets to graph grammars is presented in [CM95] such that transitions correspond to rules and firing steps to rule applications. This approach uses an encoding of transitions as graph rules similar to Def. 4.1.10 and the transition productions of [Kre81] with the difference that they contain only the individual tokens as typed graph nodes where the types represent the places marked by the tokens. The authors of [CM95] mention a subtle mismatch of the encoding as a conceptual problem, i. e., that the indistinguishable tokens of the multiset marking in the actual net are more abstract than the tokens with distinguishable individuals in the graph representation. Because of several possible matches for the individuals, there are different transformations representing the same unique firing step of a transition and there are many grammars representing the same net.

Although both constructions have inspired the transition rules for PTI nets, it is not our ambition to formalize a strict simulation relation between a PTI net's behavior and an — in some sense — equivalent (net) grammar. We rather lift the result of Theorem 4.1.11 to AHLI nets in Theorem 4.2.12 to be able to relate arbitrary net transformation steps with firing steps in Sect. 6.3.2.

Theorem 4.1.12 (Token-injective PTI Net Morphisms preserve Firings) *Given a PTI net morphism $f\colon NI_1 \to NI_2$ with injective f_I component, then for each firing step $NI_1 \overset{t,S}{\rightarrowtail} NI_1'$ there exists a firing step $NI_2 \overset{f_T(t),S'}{\rightarrowtail} NI_2'$ and a PTI net morphism $f'\colon NI_1' \to NI_2'$ (depicted in the diagram below).*

$$\begin{tikzcd}
NI_1 \arrow[r, tail, "t{,}S"] \arrow[d, "f"'] & NI_1' \arrow[d, "f'"] \\
NI_2 \arrow[r, tail, "f_T(t){,}S'"'] & NI_2'
\end{tikzcd}$$

Proof. Given $f\colon NI_1 \to NI_2$ and $NI_1 \overset{t,S}{\rightarrowtail} NI_1'$ via some token selection S as above, we have by the first part of Theorem 4.1.11 the direct transformation given by the pushouts (1) and (2) with $\varrho(t,S) = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ on the inclusion match $inc_L\colon L \to NI_1$ in Fig. 4.5.

$$\begin{tikzcd}
L \arrow[d, "inc_L"'] \arrow[dr, phantom, "(1)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d] \arrow[dr, phantom, "(2)"] & R \arrow[d, "inc_R"] \\
NI_1 \arrow[d, "f"'] \arrow[dr, phantom, "(3)"] & NI_{D1} \arrow[l, "l'"'] \arrow[r, "r'"] \arrow[d] \arrow[dr, phantom, "(4)"] & NI_1' \arrow[d, "f'"] \\
NI_2 & NI_{D2} \arrow[l] \arrow[r] & NI_2'
\end{tikzcd}$$

Figure 4.5.: DPO diagrams for a direct transformation of NI_1 and NI_2 with $\varrho(t,S)$

Note that $f\colon NI_1 \to NI_2$ satisfies the gluing condition w. r. t. this transition rule and f, because $l'_P = id_{P_1}, l'_T = id_{T_1}$[8] and none of the individuals of NI_1 is an identification point ($IP_I = \emptyset$) due to injectivity of f_I. This allows to construct the transformation with extended rule $NI_1 \overset{l'}{\leftarrow} NI_{D1} \overset{r'}{\rightarrow} NI_1'$ along pushouts (3) and (4). Hence, also $(1+3)$ and $(2+4)$ are pushouts of a direct transformation via $\varrho(t,S)$. Because $f \circ inc_L$ is clearly token-injective, the second part of Theorem 4.1.11 implies $NI_2 \overset{f_T(t),S'}{\rightarrowtail} NI_2'$. □

Remark (Firing-preserving diagrams correspond to extension diagrams). The diagram in Fig. 4.5 corresponds to an extension diagram for rule $\varrho(t,S)$ and f (as depicted below) because (1) – (4) are pushouts.

$$\begin{tikzcd}
NI_1 \arrow[r, Rightarrow, "\varrho(t{,}S){,}inc"] \arrow[d, "f"'] & NI_1' \arrow[d, "f'"] \\
NI_2 \arrow[r, Rightarrow, "\varrho(t{,}S){,}f\circ inc"] & NI_2'
\end{tikzcd}$$

[8] This means that all places and transitions of NI_1 are gluing points.

4.2. Algebraic High-Level Nets with Individual Tokens

In this section, we lift the results of the previous section to high-level nets whose tokens represent values of an algebra to a signature [EM85] as we have done in [MGE+10]. The rule-based transformation of *collective* algebraic high-level nets from [PER95] is the foundation for the approach presented in the following. The first approach of inscripting Petri nets with algebraic terms was developed in [Rei91], however, we use the approach of [EPR94], which allows us to use an arbitrary algebra, as long as it complies to the given signature.[9]

4.2.1. Definition and Firing Behavior

We lift the definitions of PTI nets to algebraic high-level nets with individual tokens:

Definition 4.2.1 (Algebraic High-Level Nets with Individual Tokens)
We define a marked AHL net with individual tokens, short AHLI net, as

$$ANI = (\Sigma, P, T, pre, post, cond, type, A, I, m) \text{ where}$$

- $AN = (\Sigma, P, T, pre, post, cond, type, A)$ is a classical AHL net with
  - signature $\Sigma = (S, OP; X)$ of sorts S, operation symbols OP and an S-indexed family of variable sets $X = (X_s)_{s \in S}$,
  - sets of places P and transitions T,
  - $pre, post\colon T \to (T_\Sigma(X) \otimes P)^{\oplus}$[10], defining the transitions' pre- and postdomains,
  - $cond\colon T \to \mathcal{P}_{fin}(Eqns(S, OP, X))$ assigning a finite set of Σ-equations (L, R, X) as firing conditions to each transition,
  - $type\colon P \to S$ typing the places of the signature's sorts,
  - a Σ-algebra A,
- I is the (possibly infinite) set of individual tokens of ANI, and
- $m\colon I \to A \otimes P$ is the marking function, assigning the individual tokens to the data elements on the places. $m(I)$ defines the actual set of data elements on the places of ANI. m does not have to be injective.

Further, we introduce some additional notations:

- $Var(t) \subseteq X$ is the set of variables occurring in equations and on the environment arcs of t,
- $CP = (A \otimes P) = \{(a, p) \in A \times P \mid a \in A_{type(p)}\}$ as the set of consistent value–place pairs,

[9] Both articles consider algebraic specifications with equations instead of signatures (which can be understood as specifications without equations). In this article we take into account signatures, only.

[10] $T_\Sigma(X) \otimes P = \{(t, p) \in T_\Sigma(X) \times P \mid t \in T_{\Sigma,type(p)}(X)\}$, i. e., the pairs where term t is of sort $type(p)$.

- the set of consistent transition assignments

  $$CT = \{(t, asg) \in T \times (Var(t) \to A) \mid \forall (L, R, X) \in cond(t)\colon \overline{asg}(L) = \overline{asg}(R)\}$$

  i. e., all firing conditions of t are valid when evaluated with the variable assignment asg[11],

- $ENV(t) = PRE(t) \cup POST(t) \subseteq (T_\Sigma(X) \otimes P)$ as the environment of a transition $t \in T$ where

  $$\begin{aligned} PRE(t) &= \{(term, p) \in (T_\Sigma(X) \otimes P) \mid pre(t)(term, p) \neq 0\} \\ POST(t) &= \{(term, p) \in (T_\Sigma(X) \otimes P) \mid post(t)(term, p) \neq 0\} \end{aligned}$$

- $ENV_P(t) = \pi_P(ENV(t)) \subseteq P$ the place environment of t,[12]

- $pre_A, post_A\colon CT \to CP^{\oplus}$, defined by

  $$\begin{aligned} pre_A(t, asg) &= (\overline{asg} \otimes id_P)^{\oplus}(pre(t)), \\ post_A(t, asg) &= (\overline{asg} \otimes id_P)^{\oplus}(post(t)) \end{aligned}$$

  Similarly, we define the sets[13]

  $$\begin{aligned} PRE_A(t, asg) &= \{(a, p) \in (A \otimes P) \mid pre_A(t, asg)(a, p) \neq 0\} \\ POST_A(t, asg) &= \{(a, p) \in (A \otimes P) \mid post_A(t, asg)(a, p) \neq 0\} \end{aligned}$$

  We can express the concrete required *number* of a token (a, p) for t to fire under assignment asg with $pre_A(t, asg)(a, p)$ by interpreting the monoid $pre_A(t, asg)$ as a function $CP \to \mathbb{N}$. Similarly, we get the produced *number* of (a, p) with $post_A(t, asg)(a, p)$. △

Example. Fig. 4.6 shows the graphical representation of an AHLI net with

- signature $\Sigma = (\{s_1, s_2, s_3\},\ \{t_{11}\colon\ \to s_1, t_{12}\colon\ \to s_1, t_2\colon\ \to s_2, t_3\colon\ \to s_3\};\ \emptyset)$,
- algebra carrier sets $A_{s_1} = \{a_1, b_1, c_1\}, A_{s_2} = \{a_2, b_2\}, A_{s_3} = \{a_3, b_3, c_1\}$
- $pre(t) = (t_{11}, p_1) \oplus (t_{12}, p_1) \oplus (t_2, p_2), post(t) = (t_3, p_3)$,
- $type(p_1) = s_1, type(p_2) = s_2, type(p_3) = s_3$,
- $cond(t) = \emptyset$,
- $I = \{x_1, y_1, x_2, x_3\}$, $m(x_1) = m(y_1) = (a_1, p_1)$, $m(x_2) = (a_2, p_2)$, $m(x_3) = (a_3, p_3)$,

[11] where $\overline{asg}\colon T_\Sigma(X) \to A$ is the evaluation of Σ-terms over variables in X to values in A. Technically, $\overline{asg} = xeval(asg)_A$ results from a free construction over asg.

[12] where π_P is the projection on the place component of the environment.

[13] Obviously, these sets are the same as $(\overline{asg} \otimes id_P) \circ PRE(t)$ and $(\overline{asg} \otimes id_P) \circ POST(t)$, respectively.

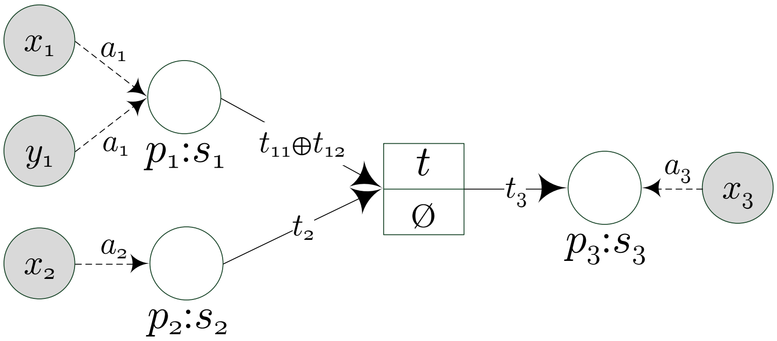


Figure 4.6.: Example AHLI net

Note that the algebraic value of an individual token is given next to the dashed arc to its carrying place. In the following, if a transition has an empty set of conditions we just denote the transition name without an explicit $\emptyset$ below.

Remark (Individual vs. collective algebraic data tokens). Each AHLI net with finite individual token marking (I, m) can be interpreted as an AHL net with collective marking

$$M = \sum_{(a,p)\in A\otimes P} |m^{-1}(a,p)|(a,p) = \sum_{i\in I} m(i)$$

where $|m^{-1}(a,p)|$ denotes the cardinality of individual tokens in I that m maps to (a,p) (cf. Sect. 4.4).

In AHL nets with collective markings, the tokens are value–place pairs (a,p) so that the tokens can be distinguished by their data values. The main difference of AHLI nets to AHL nets with collective markings is that we even distinguish tokens of the same algebraic value on the same place. Moreover, the individuals equip data tokens with identities.

The visualization of individual tokens in Fig. 4.6 is close to the formal definition as it shows the individual tokens as nodes and indicates the target place by a dashed arc and the data value by an inscription on this arc. In some cases, we may not be interested in the actual individuals so that we can use an alternative notation as in Fig. 4.7, where the individual is pointing to a token with an algebraic value on the target place. This notation is closer to marked AHL nets, and we can simply omit the nodes indicating the individuals if appropriate but we are still able to decide whether a net can fire.

Similar as for low-level PTI nets, we now define firing steps for a transition and a token selection. In addition, we have to take into account assignments evaluating the variables on the arcs and in the transition conditions to algebra values.

Definition 4.2.2 (Firing of AHLI nets)
A consistent transition assignment $(t, asg) \in CT$ for an AHLI net

$$ANI = (\Sigma, P, T, pre, post, cond, type, A, I, m)$$

is *enabled* under a *token selection* (M, m, N, n), where

- $M \subseteq I$,
- m is the token mapping of ANI,
- N is a set with $(I \setminus M) \cap N = \emptyset$,
- $n\colon N \to A \otimes P$ is a function,

if it meets the following *token selection condition*:

$$\left[\sum_{i \in M} m(i) = pre_A(t, asg)\right] \wedge \left[\sum_{i \in N} n(i) = post_A(t, asg)\right]$$

If such an asg-enabled t fires, the follower marking (I', m') is given by

$$I' = (I \setminus M) \cup N, \quad m'\colon I' \to A \otimes P \text{ with } m'(x) = \begin{cases} m(x), & x \in I \setminus M \\ n(x), & x \in N \end{cases}$$

leading to $ANI' = (AN, I', m')$ as the new AHLI net in the *firing step* $ANI \xrightarrow{t,asg} ANI'$ via (M, m, N, n). △

Remark (Token Selection). The purpose of the token selection is to specify exactly which tokens should be consumed and produced in the firing step. Thus, $M \subseteq I$ selects the individual tokens to be consumed, and N contains the set of individual tokens to be produced. Clearly, $(I \setminus M) \cap N = \emptyset$ must hold because it is impossible to add an individual token to a net that already contains this token. m and n relate the tokens to their place–value pairs. It would be sufficient to consider only the restriction $m_{|M}$ here or to infer it from the net but as a compromise on symmetry and readability we denote m in the token selection.

Note that (similar to PTI nets) because transition environments are finite, every transition can only be activated under finite token selections. Otherwise, if M or N is infinite, the sums in the token selection condition over the places are not even well-defined.

sorts:	*State*, *Bool*
opns:	$F\colon \to Bool$
	$Online, Offline, DND\colon \to State$
	$eq\colon State\ State \to Bool$
vars:	$s\colon \to State$

Table 4.1.: Signature Σ for the example AHLI net in Fig. 4.7

Example. We demonstrate firing of AHLI nets with an example that is similar to the one for PTI nets for Example 2. The net shown in Fig. 4.7 models a simple client that can be in three states *online*, *offline*, and *do not disturb*. This net has the signature Σ given in Table 4.1 and its Σ-algebra A consists of the carrier sets $A_{State} =$

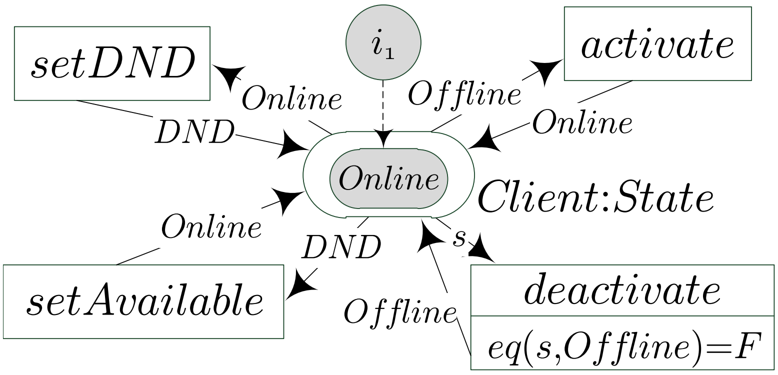


Figure 4.7.: Example client AHLI net

$\{Online, Offline, DND\}$, $A_{Bool} = \{F, T\}$ and the equality function eq_A on the elements of A_{State} with the obvious correspondence to the signature elements.

The net is currently marked with $I = \{i_1\}$, $m(i_1) = (Online, \texttt{Client})$, which indicates that the client is online. Note the notation variation with the token value depicted as a token ellipse on the place instead of an inscription on the individual token's arc to the place. The net's transitions model the change of the client's state. Intuitively, a client can be *activated* if it is online, be set to *do not disturb* if it is online, be set back to online status (which implies availability) if "do not disturb" is set, and finally be *deactivated* if it is in any state but offline. Note the firing condition of the transition *deactivate*, which demands the consumed state token (assigned to variable s) to be different from *Offline*.

For this marking, the pair $(deactivate,\ asg\colon\ Var(deactivate) \to A)$ with the variable assignment $asg(s) = Online$ is consistent, because the evaluated firing condition of *deactivate* is valid:

$$\overline{asg}(eq(s, Offline)) = eq_A(Online, Offline) = F = \overline{asg}(F)$$

Consider the token selection $S = (M, m, N, n)$ with $M = \{i_1\}$, m as the current client net marking, $N = \{i'_1\}$, and $n(i'_1) = (Offline, \texttt{Client})$. Clearly, S is a valid token selection because $M \subseteq I$ and the sets $I \setminus M$ and N are disjoint. The consistent transition assignment $(deactivate, asg)$ is activated under S because the token selection condition is satisfied:

$$\sum_{i \in M} m(i) = m(i_1) = (Online, \texttt{Client}) = (asg(s), \texttt{Client}) = pre_A(deactivate, asg)$$

$$\sum_{i \in N} n(i) = n(i'_1) = (Offline, \texttt{Client}) = post_A(deactivate, asg)$$

The firing step of $(deactivate, asg)$ via S leads to the follower marking $I' = \{i'_1\}$, $m'(i'_1) = n(i'_1) = (Offline, \texttt{Client})$ as depicted in Fig. 4.8.

For the next subsection about AHLI net transformation, we define a category of AHLI nets with suitable morphisms and show constructions for pushouts and pullbacks in this category.

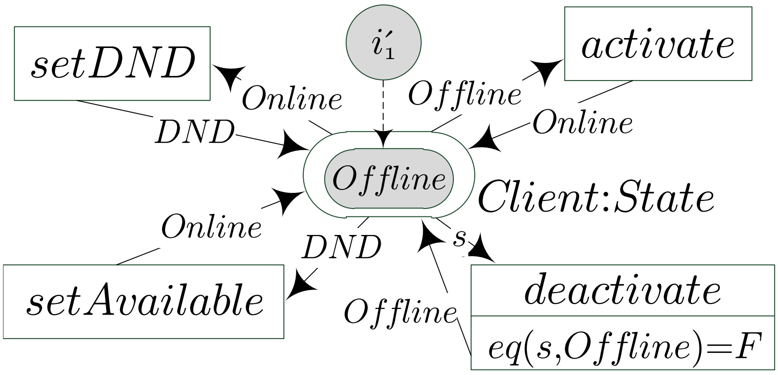


Figure 4.8.: Example client AHLI net after firing transition *deactivate*

Definition 4.2.3 (AHLI Net Morphisms and Category AHLINets) Given two AHLI nets $ANI_i = (\Sigma_i, P_i, T_i, pre_i, post_i, cond_i, type_i, A_i, I_i, m_i)$, $i \in \{1, 2\}$, an AHLI net morphism $f\colon ANI_1 \to ANI_2$ is a pentuple

$$f = (f_\Sigma\colon \Sigma_1 \to \Sigma_2, f_P\colon P_1 \to P_2, f_T\colon T_1 \to T_2, f_A\colon A_1 \to V_{f_\Sigma}(A_2), f_I\colon I_1 \to I_2)$$

such that the following diagrams commute (componentwise for pre_i and $post_i$)[14]:

$$\begin{tikzcd}
\mathcal{P}_{fin}(Eqns(\Sigma_1)) \arrow[d, "\mathcal{P}_{fin}(f_\Sigma^\#)"'] & T_1 \arrow[l, "cond_1"'] \arrow[r, shift left, "pre_1"] \arrow[r, shift right, "post_1"'] \arrow[d, "f_T"'] & (T_{\Sigma_1}(X_1) \otimes P_1)^\oplus \arrow[d, "(f_\Sigma^\# \otimes f_P)^\oplus"] \\
\mathcal{P}_{fin}(Eqns(\Sigma_2)) & T_2 \arrow[l, "cond_2"] \arrow[r, shift left, "pre_2"] \arrow[r, shift right, "post_2"'] & (T_{\Sigma_2}(X_2) \otimes P_2)^\oplus
\end{tikzcd}$$

$$\begin{tikzcd}
P_1 \arrow[r, "type_1"] \arrow[d, "f_P"'] & S_1 \arrow[d, "f_\Sigma"'] & I_1 \arrow[r, "m_1"] \arrow[d, "f_I"'] & A_1 \otimes P_1 \arrow[d, "f_A \otimes f_P"] \\
P_2 \arrow[r, "type_2"'] & S_2 & I_2 \arrow[r, "m_2"] & A_2 \otimes P_2
\end{tikzcd}$$

The category **AHLINets** consists of all AHLI nets as objects and all AHLI net morphisms. △

Remark (Generalized algebra morphisms). Because $V_{f_\Sigma}(A_2)$ just forgets some carrier sets of A_2 if we consider them as a family of sets, we may use f_A also with A_2 as codomain omitting the postponed family of identities $(\iota_s\colon (V_{f_\Sigma}(A_2))_s \to A_{2,f_\Sigma(s)})_{s \in S}$. Moreover, in the following constructions the algebra parts of the pushout cospan (and the pullback span resp.) result from general constructions in the Grothendieck construct with objects $(\Sigma, A \in \mathbf{Alg}(\boldsymbol{\Sigma}))$ and generalized algebra morphisms. See [EBO92, TBG91] for amalgamation of algebras and limits/colimits in Grothendieck constructs and also [EM85, EOP06] for details on the usage of free functors.

[14] V_{f_Σ} is the forgetful functor induced by signature homomorphism f_Σ, such that $f_A\colon A_1 \to V_{f_\Sigma}(A_2)$ is a generalized Σ_1-homomorphism. $f_\Sigma^\#$ is the extension of f_Σ to terms and equations.

Definition 4.2.4 (Translated Variable Assignments)
For a transition assignment $(t, asg\colon Var(t) \to A_1)$ and an AHLI net morphism $f\colon ANI_1 \to ANI_1$ with f_Σ being bijective on the variables in $Var(t)$, we denote with $asg_f = f_A \circ asg \circ f^{-1}_{\Sigma|Var(t)}\colon Var(f_T(t)) \to A_2$ the *translation* of asg along f.

$$\begin{tikzcd}
Var(t) \arrow[r, "asg"] \arrow[dr, phantom, "="] & A_1 \arrow[d, "f_A"] \\
Var(f_T(t)) \arrow[u, "f^{-1}_{\Sigma|Var(t)}"] \arrow[r, "asg_f"'] & A_2
\end{tikzcd}$$

Actually, for every transition $t \in T_1$, it follows directly from the AHLI net morphism properties of f that $f_{\Sigma|Var(t)}\colon Var(t) \to Var(f_T(t))$ is surjective. It is sufficient to demand injectivity of $f_{\Sigma|Var(t)}$ for asg_f being well-defined. △

Fact 4.2.5 (Construction of Pushouts in AHLINets)
Pushouts in **AHLINets** are constructed componentwise in **AHLNets** and **Sets**. So, (1) is a PO in **AHLINets** iff (2) is a PO in **AHLNets** and (3) is a PO in **Sets** with the components of the **AHLINets** morphisms, where $m_3\colon I_3 \to A_3 \otimes P_3$ is induced by PO object I_3 in the commutative cube below (whose front's place components let the front commutate because of the PO in the net structure).

$$\begin{tikzcd}
ANI_0 \arrow[r, "f_1"] \arrow[d, "f_2"'] \arrow[dr, phantom, "(1)"] & ANI_1 \arrow[d, "g_1"] \\
ANI_2 \arrow[r, "g_2"'] & ANI_3
\end{tikzcd}$$

$$\begin{tikzcd}
AN_0 \arrow[r, "f_1'"] \arrow[d, "f_2'"'] \arrow[dr, phantom, "(2)"] & AN_1 \arrow[d, "g_1'"] \\
AN_2 \arrow[r, "g_2'"'] & AN_3
\end{tikzcd}
\qquad
\begin{tikzcd}
I_0 \arrow[r, "f_{1I}"] \arrow[d, "f_{2I}"'] \arrow[dr, phantom, "(3)"] & I_1 \arrow[d, "g_{1I}"] \\
I_2 \arrow[r, "g_{2I}"'] & I_3
\end{tikzcd}$$

$$\begin{tikzcd}
I_0 \arrow[rr, "f_{1I}"] \arrow[dr, "m_0"] \arrow[dd, "f_{2I}"'] & & I_1 \arrow[dr, "m_1"] \arrow[dd, "g_{1I}"] & \\
& A_0 \otimes P_0 \arrow[rr, "f_{1A} \otimes f_{1P}"] \arrow[dd, "f_{2A} \otimes f_{2P}"'] & & A_1 \otimes P_1 \arrow[dd, "g_{1A} \otimes g_{1P}"] \\
I_2 \arrow[rr, "g_{2I}"] \arrow[dr, "m_2"'] & & I_3 \arrow[dr, dashed, "m_3"] & \\
& A_2 \otimes P_2 \arrow[rr, "g_{2A} \otimes g_{2P}"'] & & A_3 \otimes P_3
\end{tikzcd}$$

If f_{1X}, for $X \in \{P, T, I\}$, is injective then g_{2X} is as well and similar for components of f_2 and g_1 and the other diagrams.

For the construction of pushouts in **AHLNets** we refer to [PER95].

Proof. See appendix A.3.1.2 □

Example. Fig. 4.9 shows two pushouts in **AHLINets**.

Fact 4.2.6 (Construction of Pullbacks in AHLINets)
Pullbacks in **AHLINets** along injective **AHLNets** morphisms[15] are constructed componentwise in **AHLNets** and **Sets**.

Consider a commutative square (1) in **AHLINets** with g_1' being injective. (1) is a PB in **AHLINets** iff (2) is a PB in **AHLNets** and (3) is a PB in **Sets** with the components of the **AHLINets** morphisms, where $m_0\colon I_0 \to A \otimes P_0$ is induced by PB object P_0 in the commutative cube below (whose front's place components let the front commutate because of the PB in the net structure).

$$\begin{tikzcd}
ANI_0 \arrow[r, "f_1"] \arrow[d, "f_2"'] \arrow[dr, phantom, "(1)"] & ANI_1 \arrow[d, "g_1"] \\
ANI_2 \arrow[r, "g_2"'] & ANI_3
\end{tikzcd}$$

$$\begin{tikzcd}
AN_0 \arrow[r, "f_1'"] \arrow[d, "f_2'"'] \arrow[dr, phantom, "(2)"] & AN_1 \arrow[d, "g_1'"] & I_0 \arrow[r, "f_{1I}"] \arrow[d, "f_{2I}"'] \arrow[dr, phantom, "(3)"] & I_1 \arrow[d, "g_{1I}"] \\
AN_2 \arrow[r, "g_2'"'] & AN_3 & I_2 \arrow[r, "g_{2I}"'] & I_3
\end{tikzcd}$$

$$\begin{tikzcd}
I_0 \arrow[rr, "f_{1I}"] \arrow[dr, dashed, "m_0"] \arrow[dd, "f_{2I}"'] & & I_1 \arrow[dr, "m_1"] \arrow[dd, "g_{1I}"] & \\
& A_0 \otimes P_0 \arrow[rr, "f_{1A} \otimes f_{1P}"] \arrow[dd, "f_{2A} \otimes f_{2P}"] & & A_1 \otimes P_1 \arrow[dd, "g_{1A} \otimes g_{1P}"] \\
I_2 \arrow[rr, "g_{2I}"] \arrow[dr, "m_2"'] & & I_3 \arrow[dr, "m_3"] & \\
& A_2 \otimes P_2 \arrow[rr, "g_{2A} \otimes g_{2P}"'] & & A_3 \otimes P_3
\end{tikzcd}$$

Proof. See appendix A.3.1.2 □

Example. The pushouts in Fig. 4.9 are pullbacks, too.

4.2.2. Transformation of AHLI Nets

This section is about rule-based transformation of AHLI nets. For this, we use the double pushout (DPO) approach, which has also been used for collective AHL nets in [PER95] and which has been investigated in the context of $\mathcal{M}$-adhesive systems in [EEPT06]. We characterize the applicability of a rule at some match by initial pushouts.

Definition 4.2.7 (AHLI Transformation Rules)
An AHLI transformation rule is a span of injective **AHLINets** morphisms

$$\varrho = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R).$$

We call L the left-hand side (LHS), K the interface, and R the right-hand side (RHS) of ϱ. △

Remark. AHLI Rule morphisms are not required to be marking-strict in order to obtain an $\mathcal{M}$-adhesive category (see Sect. 4.3). This allows us to arbitrarily change the marking of an AHLI net by applying rules with corresponding places and individual tokens in L and R.

[15] We require morphisms that are injective on the net structure in order to obtain componentwise pullbacks in **AHLNets**. See [EEPT06] for details.

Definition 4.2.8 (AHLI Transformation)
Given an AHLI transformation rule $\varrho = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ and an AHLI net ANI_1 with an AHLI net morphism $m\colon L \to ANI_1$, called the match, a direct AHLI net transformation $ANI_1 \overset{\varrho,m}{\Longrightarrow} ANI_2$ from ANI_1 to the AHLI net ANI_2 is given by the following DPO diagram in the category **AHLINets**:

$$\begin{tikzcd}
L \arrow[d, "m"'] \arrow[dr, phantom, "(\mathbf{PO_1})"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d] \arrow[dr, phantom, "(\mathbf{PO_2})"] & R \arrow[d, "m^*"] \\
ANI_1 & ANI_0 \arrow[l] \arrow[r] & ANI_2
\end{tikzcd}$$

To be able to decide whether a rule is applicable at a certain match, we formulate a gluing condition for AHLI nets, such that there exists a pushout complement of the left rule morphisms and the match if (and only if) they fulfill the gluing condition. The correctness of the gluing condition is shown via a proof on initial pushouts over matches, according to [EEPT06].

Definition 4.2.9 (Gluing Condition in AHLINets)
Given AHLI nets K, L, and ANI and AHLI net morphisms $l\colon K \to L$ and $f\colon L \to ANI$. We define the set of identification points[16]

$$IP = IP_P \cup IP_T \cup IP_I \text{ with}$$

- $IP_P = \{x \in P_L \mid \exists x' \neq x\colon f_P(x) = f_P(x')\}$,
- $IP_T = \{x \in T_L \mid \exists x' \neq x\colon f_T(x) = f_T(x')\}$,
- $IP_I = \{x \in I_L \mid \exists x' \neq x\colon f_I(x) = f_I(x')\}$,

the set of dangling points[17]

$$DP = DP_T \cup DP_I \text{ with}$$

- $DP_T = \{p \in P_L \mid \exists t \in T_{ANI} \setminus f_T(T_L)\colon f_P(p) \in ENV_P(t)\}$,
- $DP_I = \{p \in P_L \mid \exists i \in I_{ANI} \setminus f_I(I_L)\colon f_P(p) = \pi_P(m_{ANI}(i))\}$,

and the set of gluing points[18]

$$GP = l_P(P_K) \cup l_T(T_K) \cup l_I(I_K) \qquad \triangle$$

We say that l and f satisfy the gluing condition if $IP \cup DP \subseteq GP$. Given an AHLI rule $\varrho = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$, we say that ϱ and f satisfy the gluing condition iff l and f satisfy the gluing condition.

[16] That is, all elements in L that are mapped noninjectively by f.

[17] That is, all places in L that are mapped to a place in ANI such that the mapped place would leave a dangling arc or token if it was removed from ANI.

[18] That is, all elements in L that have a preimage in K.

$$\begin{tikzcd}
L \arrow[d, "f"'] & K \arrow[l, "l"'] \arrow[r, "r"] & R \\
ANI & &
\end{tikzcd}$$

The following fact shows the correspondence between the gluing condition in **AHLINets** and the categorical gluing condition (see Def. A.2.2), which is a necessary and sufficient condition for the existence of unique pushout complements (up to isomorphism) in $\mathcal{M}$-adhesive categories (cf. Sect. 4.3). It holds in all $\mathcal{M}$-adhesive categories $(\mathbf{AHLINets}, \mathcal{M})$ whose morphism class $\mathcal{M}$ of monomorphisms contains at least inclusions with identities for signature and algebra parts (for concrete instantiations see Sect. 4.3).

Theorem 4.2.10 (Gluing Condition for AHLI Transformation)
Given an AHLI rule $\varrho = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ of an $\mathcal{M}$-adhesive system and a match $f\colon L \to ANI$ into an AHLI net $ANI = (AN, I, m\colon I \to P_{AN})$. The rule ϱ is applicable on match f, i. e., there exists a (unique) pushout complement ANI_0 in the diagram below, iff ϱ and f satisfy the gluing condition in **AHLINets**.

$$\begin{tikzcd}
L \arrow[d, "f"'] \arrow[dr, phantom, "(PO)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, dashed, "f'"] & R \\
ANI & ANI_0 \arrow[l, dashed] &
\end{tikzcd}$$

Proof. By Fact A.2.19 the rule ϱ and match f satisfy the gluing condition in **AHLINets** if and only if ϱ and f satisfy the categorical gluing condition, which by Fact A.2.3 is a sufficient and necessary condition for the existence of a (unique) pushout complement ANI_0. □

4.2.3. Correspondence of Transition Firing and Rules

As for PTI nets in Sect. 4.1.3, we give a construction for transition rules of AHLI nets that simulate a firing step of some transition under a specific token selection and show that firing of a transition corresponds to an application of a transition rule and vice versa. With this correspondence we can easily show that token-injective AHLI net morphisms preserve firing steps.

Definition 4.2.11 (AHLI Transition Rules)
Given an AHLI net

$$ANI = (\Sigma, P, T, pre, post, cond, type, A, I, m)$$

we define the *transition rule* for a consistent transition assignment $(t, asg) \in CT_{ANI}$, being enabled under the token selection $S = (M, m, N, n)$, as the rule

$$\varrho(t, asg, S) = (L_t \overset{l}{\leftarrow} K_t \overset{r}{\rightarrow} R_t) \text{ with}$$

- the common fixed AHL net part $AN_t = (\Sigma, P_t, \{t\}, pre_t, post_t, cond_t, type_t, A)$, where $P_t = ENV_P(t), pre_t(t) = pre_(t), post_t(t) = post(t), cond_t(t) = cond(t)$, and $\forall p \in P_t \colon type_t(p) = type(p)$,
- $L_t = (AN_t, M, m_t \colon M \to A \otimes P_t)$, with $m_t(x) = m(x)$,
- $K_t = (AN_t, \emptyset,\ \emptyset \colon \emptyset \to (A \otimes P_t))$,
- $R_t = (AN_t, N, n_t \colon N \to A \otimes P_t)$, with $n_t(x) = n(x)$,
- l, r being the obvious inclusions on the rule nets.

m_t and n_t are well-defined because t is enabled under S: The token selection condition implies that $\forall x \in M \colon m(x) \in PRE_A(t, asg)$ and due to the construction of AN_t we have $PRE_A(t, asg) \subseteq (A \otimes ENV_P(t)) = (A \otimes P_t)$. The argument for n_t works analogously.

Note that (t, asg) is enabled under S in L_t. △

Remark. The structure of a transition rule depends only on the transition and the token selection, for which there may exist several enabled transition assignments. Therefore different consistent transition assignments may have the same correspondent transition rule. Nevertheless, we denote an AHLI transition rule as $\varrho(t, asg, S)$ rather than $\varrho(t, S)$ to remember the concrete assignment this rule is intended to simulate.

Example. Fig. 4.9 shows an AHLI transition rule $\varrho(t, asg, S) = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$.

Example. The following diagram shows the two pushouts in **AHLINets** resulting from applying the AHL transition rule $\varrho(t, asg, S) = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ for

$$asg = \{t_{11} \mapsto a_1, t_{12} \mapsto b_1, t_2 \mapsto a_2, t_3 \mapsto a_3\}$$

to the net ANI. All nets in this diagram have the same signature and algebra as the example net in Fig. 4.6.

Theorem 4.2.12 (Correspondence of AHLI Firings and Transformations)

1. Each firing step $ANI \xrightarrow{t,asg,S} ANI'$ via token selection $S = (M, m, N, n)$ corresponds to an induced direct transformation $ANI \overset{\varrho(t,asg,S),f}{\Longrightarrow} ANI'$ via the transition rule $\varrho(t, asg, S)$, where the match $f \colon L \to ANI$ is an inclusion.

2. Each direct transformation $ANI \overset{\varrho(t,asg,S),f}{\Longrightarrow} ANI_1$, via some transition rule $\varrho(t, asg, S)$ with $t \in T_{ANI}$, token selection $S = (M, m, N, n)$, and token-injective match $f \colon L_{\varrho(t,asg,S)} \to ANI$ with f_Σ injective on $Var(t)$, implies a consistent transition assignment $(f_T(t), asg_f)$ being enabled in ANI under some token selection $\bar{S}$ with firing step $ANI \xrightarrow{f_T(t),asg_f,\bar{S}} ANI^*$ such that $ANI^* \cong ANI_1$.

Proof. See appendix A.3.3. □

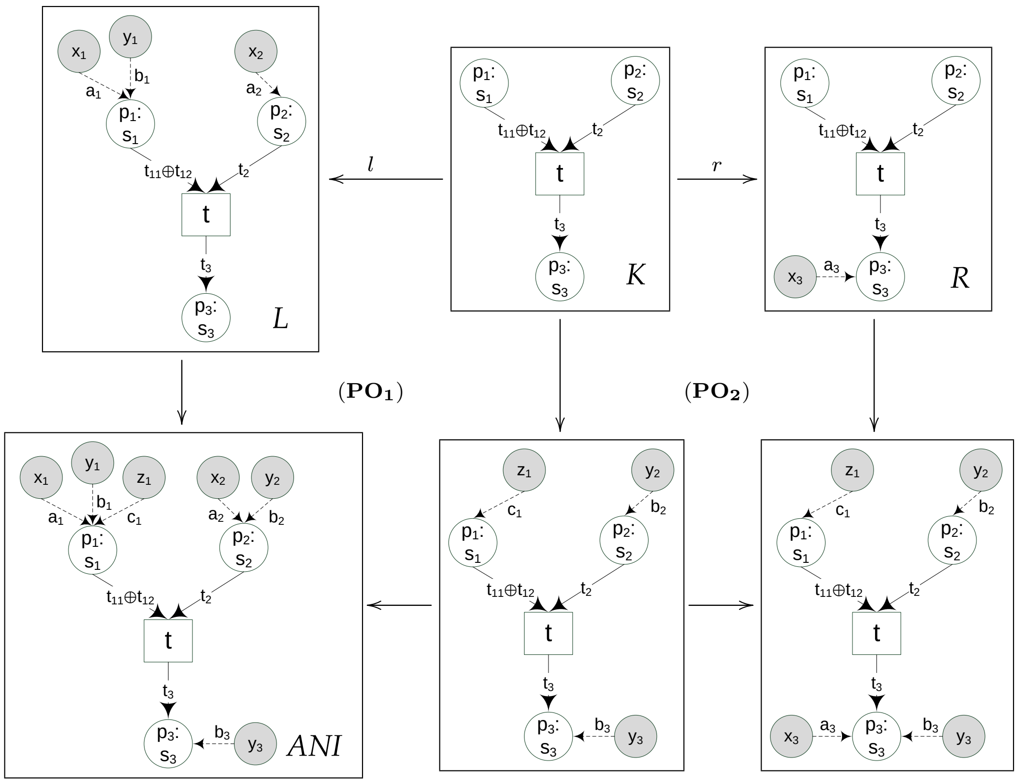


Figure 4.9.: Direct DPO transformation in **AHLINets** simulating a firing step

Remark. For related approaches to the construction of firing rules, consider the remark and the references we give after Theorem 4.1.11.

Theorem 4.2.13 (Token-injective AHLI Net Morphisms preserve Firings) *Given an AHLI net morphism $f\colon ANI_1 \to ANI_2$ such that f_I is injective and f_Σ is injective on all sets $Var(t)$ for every $t \in T_1$*[19]*, then for each firing step $ANI_1 \xrightarrow{t,asg,S} ANI'_1$ there exists a firing step $ANI_2 \xrightarrow{f_T(t),asg_f,S'} ANI'_2$ and an AHLI net morphism $f'\colon ANI'_1 \to ANI'_2$ (depicted in the diagram below).*

$$\begin{tikzcd}
ANI_1 \arrow[r, tail, "{t,asg,S}"] \arrow[d, "f"'] & ANI'_1 \arrow[d, "f'"] \\
ANI_2 \arrow[r, tail, "{f_T(t),asg_f,S'}"'] & ANI'_2
\end{tikzcd}$$

Proof. Given $f\colon ANI_1 \to ANI_2$ and $ANI_1 \xrightarrow{t,asg,S} ANI'_1$ via some token selection S as above, we have by the first part of Theorem 4.2.12 a direct transformation with $\varrho(t,asg,S) = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ given by the pushouts (1) and (2) in Fig. 4.10.

[19] See the “Context Condition” in [EP97].

$$\begin{tikzcd}
L \arrow[d, "inc_L"'] & & K \arrow[ll, "l"'] \arrow[rr, "r"] \arrow[d] \arrow[ll, phantom, "(\mathbf{1})", shift right=20] & & R \arrow[d, "inc_R"] \arrow[ll, phantom, "(\mathbf{2})", shift left=20] \\
ANI_1 \arrow[d, "f"'] & & ANI_{D1} \arrow[ll, "l'"'] \arrow[rr, "r'"] \arrow[d] \arrow[ll, phantom, "(\mathbf{3})", shift right=20] & & ANI'_1 \arrow[d, "f'"] \arrow[ll, phantom, "(\mathbf{4})", shift left=20] \\
ANI_2 & & ANI_{D2} \arrow[ll] \arrow[rr] & & ANI'_2
\end{tikzcd}$$

Figure 4.10.: DPO diagram for a direct transformation of ANI_1 and ANI_2 with $\varrho(t, asg, S)$

Note that $f\colon ANI_1 \to ANI_2$ satisfies the gluing condition w.r.t. l', because $l'_P = id_{P_1}, l'_T = id_{T_1}$ and $IP_I = \emptyset$ due to injectivity of f_I. This allows to construct the direct transformation with extended rule $ANI_1 \overset{l'}{\leftarrow} ANI_{D1} \overset{r'}{\rightarrow} ANI'_1$ along pushouts (3) and (4). Hence, also $(1 + 3)$ and $(2 + 4)$ are pushouts, defining a direct transformation via $\varrho(t, asg, S)$. Because $f \circ inc_L$ is token-injective and injective on the variables $Var(t)$, the second part of Theorem 4.2.12 implies $ANI_2 \overset{f_T(t), asg_f, S'}{\rightarrowtail\!\!\longrightarrow} ANI'_2$. □

Remark (Firing-preserving diagrams correspond to extension diagrams). The diagram in Fig. 4.10 corresponds to an extension diagram for rule $\varrho(t, asg, S)$ and f (as depicted below) because (1) – (4) are pushouts.

$$\begin{tikzcd}
ANI_1 \arrow[r, Rightarrow, "{\varrho(t,asg,S),inc}"] \arrow[d, "f"'] & ANI'_1 \arrow[d, "f'"] \\
ANI_2 \arrow[r, Rightarrow, "{\varrho(t,asg,S),f\circ inc}"] & ANI'_2
\end{tikzcd}$$

4.3. $\mathcal{M}$-adhesive Categories for Individual Petri Systems

We use the notion of $\mathcal{M}$-adhesive category [EGH10], which is short for vertical weak adhesive HLR category. In $\mathcal{M}$-adhesive categories Van Kampen (VK) squares only need to satisfy the vertical VK-property, i. e., the VK-property has to hold for cubes where the vertical morphisms are in $\mathcal{M}$. In contrast, for a weak adhesive HLR categories it is required that the VK-property also holds for cubes where the horizontal morphisms are in $\mathcal{M}$. However, as shown in [EGH10] all the main results of [EEPT06] are still valid for $\mathcal{M}$-adhesive categories.

[Pra08] shows how to construct a marking category for *collective* Petri nets, which can be used to prove the properties of $\mathcal{M}$-adhesive categories — i. e., weak adhesive high-level replacement systems in the sense of [EEPT06] — for marked Petri nets. In this section we present a similar construction for individual markings and show some instantiations of $\mathcal{M}$-adhesive categories for PTI and AHLI nets.

Definition 4.3.1 (Individual Petri Systems)
Given a category **Nets** of nets, an *individual system* $IS = (N, I, m)$ is given by a net $N \in \mathbf{Nets}$, a set of individuals I, and a function $m\colon I \to M(N)$, where $M\colon \mathbf{Nets} \to \mathbf{Sets}$ is a functor assigning a marking set to each net N.

For systems $IS_1 = (N_1, I_1, m_1)$ and $IS_2 = (N_2, I_2, m_2)$, an individual system morphism $f_{IS} = (f_N, f_I)\colon IS_1 \to IS_2$ consists of a net morphisms $f_N\colon N_1 \to N_2$ and a function on the individuals $f_I\colon I_1 \to I_2$ such that $M(f_N) \circ m_1 = m_2 \circ f_I$ as depicted below.

$$\begin{tikzcd}
I_1 \arrow[r, "m_1"] \arrow[d, "f_I"'] \arrow[dr, phantom, "="] & M(N_1) \arrow[d, "M(f_N)"] \\
I_2 \arrow[r, "m_2"] & M(N_2)
\end{tikzcd}$$

All individual systems together with all individual system morphisms constitute the category $\mathbf{ISystems}(\mathbf{Nets}, M)$. △

Theorem 4.3.2 (ISystems(Nets, M) is $\mathcal{M}$-adhesive)
Given an $\mathcal{M}$-adhesive category $(\mathbf{Nets}, \mathcal{M})$ of nets with a marking set functor $M\colon \mathbf{Nets} \to \mathbf{Sets}$ that preserves pullbacks along $\mathcal{M}$-morphisms, then the category

$$(\mathbf{ISystems}(\mathbf{Nets}, M), \mathcal{M} \times \mathcal{M}_{inj})$$

of individual systems over these nets is $\mathcal{M}$-adhesive as well, where $\mathcal{M}_{inj}$ is the class of all injective functions in **Sets**.

Proof. Consider the comma category $\mathbf{C} = ComCat(ID_{\mathbf{Sets}}, M, \{m\})$, with objects $(I, N, op_m\colon I \to M(N))$ and morphisms $f_{\mathbf{C}} = (f_I, f_N) \in Mor_{\mathbf{Sets}} \times Mor_{\mathbf{Nets}}$. Obviously, $\mathbf{C}$ is isomorphic to $\mathbf{ISystems}(\mathbf{Nets}, M)$.

By applying Thm. 1.4 (Construction of [Weak] Adhesive HLR Categories) from [PEL08], we can conclude that $(\mathbf{C}, \mathcal{M} \times \mathcal{M}_{inj})$ is $\mathcal{M}$-adhesive because

- $(\mathbf{Nets}, \mathcal{M})$ and $(\mathbf{Sets}, \mathcal{M}_{inj})$ are $\mathcal{M}$-adhesive categories,
- the functor $ID_{\mathbf{Sets}}$ preserves pushouts along injective functions, and
- the functor $M\colon \mathbf{Nets} \to \mathbf{Sets}$ preserves pullbacks along $\mathcal{M}$-morphisms.

Hence, $(\mathbf{ISystems}(\mathbf{Nets}, M), \mathcal{M} \times \mathcal{M}_{inj})$ is $\mathcal{M}$-adhesive as well. □

Theorem 4.3.3 (PTINets is $\mathcal{M}$-adhesive)
The category $(\mathbf{PTINets}, \mathcal{M}_{inj})$ is an $\mathcal{M}$-adhesive category where

$$\mathcal{M}_{inj} = \{f \in Mor_{\mathbf{PTINets}} \mid f_P, f_T, f_I \textit{ injective}\}$$

Proof. We already know from [Pra08] that $(\mathbf{PTNets}, \mathcal{M}')$ is $\mathcal{M}$-adhesive with $\mathcal{M}'$ being the class of all injective Petri net morphisms. Given the functor $M_{PT}\colon \mathbf{PTNets} \to \mathbf{Sets}$ with $M_{PT}(P, T, pre, post) = P$, we have $\mathbf{PTINets} \cong \mathbf{ISystems}(\mathbf{PTNets}, M_{PT})$.

M_{PT} preserves pullbacks along $\mathcal{M}'$-morphisms because pullbacks along injective morphisms are constructed componentwise in **PTNets**, hence Theorem 4.3.2 states that $(\mathbf{PTINets}, \mathcal{M}_{inj})$ is $\mathcal{M}$-adhesive. □

For PTI net transformation systems, we require rules with rule morphisms in $\mathcal{M}$ and one of the following alternatives for matches:

1. general matches by **PTINets** morphisms,
2. injective matches by $\mathcal{M}$,
3. token-injective matches by $\mathcal{M}' = \{f \in \mathbf{PTINets} \mid f_I \text{ inj.}\}$

For all three cases we have a suitable theory by DPO transformations with general, $\mathcal{M}$-matching resp. $\mathcal{M}'$-matching in $\mathcal{M}$-adhesive categories.

Theorem 4.3.4 (AHLINets(Σ) is $\mathcal{M}$-adhesive)
For a fixed signature Σ the category $(\mathbf{AHLINets}(\mathbf{\Sigma}), \mathcal{M}_I)$ *is an $\mathcal{M}$-adhesive category where*

- $\mathbf{AHLINets}(\mathbf{\Sigma})$ *is the full subcategory of* **AHLINets** *containing all AHLI nets with the signature* Σ,
- $\mathcal{M}_I = \{f \in Mor_{\mathbf{AHLINets}(\mathbf{\Sigma})} \mid f_\Sigma = id_\Sigma, f_A \textit{ isom., and } f_P, f_T, f_I \textit{ injective}\}$

Proof. As shown in [Pra08], we already know that the category $(\mathbf{AHLNets}(SP), \mathcal{M}')$ of AHL nets over a specification SP is $\mathcal{M}$-adhesive with

$$\mathcal{M}' = \{f \in Mor_{\mathbf{AHLNets}(SP)} \mid f_A \text{ isomorphic and } f_P, f_T \text{ injective}\}.$$

Given the functor $M_{AHL,\Sigma}\colon \mathbf{AHLNets}(\mathbf{\Sigma}, \emptyset) \to \mathbf{Sets}$ with

$$M_{AHL,\Sigma}(P, T, pre, post, cond, type, A) = A \otimes P,$$

we have $\mathbf{AHLINets}(\mathbf{\Sigma}) \cong \mathbf{ISystems}(\mathbf{AHLNets}(\mathbf{\Sigma}, \emptyset), M_{AHL,\Sigma})$. $M_{AHL,\Sigma}$ preserves pullbacks along $\mathcal{M}'$-morphisms because pullbacks along injective morphisms are constructed componentwise in $\mathbf{AHLNets}(\mathbf{\Sigma}, \emptyset)$ and we have only algebra isomorphisms in $\mathcal{M}'$. Hence, Theorem 4.3.2 states that $(\mathbf{AHLINets}(\mathbf{\Sigma}), \mathcal{M}_I)$ is $\mathcal{M}$-adhesive. □

Theorem 4.3.5 (AHLINets is $\mathcal{M}$-adhesive)
The category $(\mathbf{AHLINets}, \mathcal{M}_I)$ *is an $\mathcal{M}$-adhesive category where*

$$\mathcal{M}_I = \{f \in Mor_{\mathbf{AHLINets}} \mid f_\Sigma \textit{ injective }, f_A \textit{ isomorphic, and } f_P, f_T, f_I \textit{ injective}\}$$

Proof. As shown in [Pra08], we already know that the category of generalized AHL nets $(\mathbf{AHLNets}, \mathcal{M}')$ is $\mathcal{M}$-adhesive with

$$\mathcal{M}' = \{f \in Mor_{\mathbf{AHLNets}} \mid f_{SP} \text{ strict injective}, f_A \text{ isom., and } f_P, f_T \text{ injective}\}.$$

We consider its full subcategory $(\mathbf{AHLNets}_\emptyset, \mathcal{M}'_{|\mathbf{AHLNets}_\emptyset})$ of generalized AHL nets with empty sets of specification equations, which by Thm. 2.3(i) from [Pra08] is itself $\mathcal{M}$-adhesive. Note that all injective morphisms between specifications without equations are strict.

Given the functor $M_{AHL} \colon \mathbf{AHLNets}_\emptyset \to \mathbf{Sets}$ with

$$M_{AHL}(SP, P, T, pre, post, cond, type, A) = A \otimes P,$$

we have $\mathbf{AHLINets} \cong \mathbf{ISystems}(\mathbf{AHLNets}_\emptyset, M_{AHL})$. M_{AHL} preserves pullbacks along $\mathcal{M}'_{|\mathbf{AHLNets}_\emptyset}$-morphisms because pullbacks along injective morphisms are constructed componentwise in $\mathbf{AHLNets}_\emptyset$ and we have only algebra isomorphisms in $\mathcal{M}'$. Hence, Theorem 4.3.2 states that $(\mathbf{AHLINets}, \mathcal{M}_I)$ is $\mathcal{M}$-adhesive. □

4.4. Equivalence of Firing Behavior in Nets with Individual Tokens

We demonstrate that the concept of nets with individual tokens is compatible with the concept of Petri systems with collective markings using a construction. For this purpose we define a behavioral equivalence relation for every class of nets with individual tokens — introduced as individual systems in Def. 4.3.1 — such that the equivalence classes are in one-to-one correspondence to the nets with collective markings. For the cases of PTI and AHLI nets, we prove that this relation characterizes equivalent firing behavior. For this, we show and use the result that the concrete collection constructions for PTI and AHLI nets preserve and reflect the firing behavior.

The following construction flattens a net with individual tokens to a net with collective marking and forgets the individual token elements. Because the sums of collective markings are elements of the *free* commutative monoid over the set of places, a collective marking consist of a finite number of tokens. Therefore, we restrict the translation to nets with a finite number of individual tokens on a finite number of places, i. e., with a finite marking.

Definition 4.4.1 (Collective Construction for Individual Petri Systems) For a category $\mathcal{IS} = \mathbf{ISystems}(\mathbf{Nets}, M)$ of individual systems, we define the collection construction $Coll_{\mathcal{IS}} \colon |\mathcal{IS}_{fin}| \to CSystems(\mathbf{Nets}, M)$, where $\mathcal{IS}_{fin}$ is the full subcategory of $\mathcal{IS}$ of nets having finite token sets and

$$CSystems(\mathbf{Nets}, M) = \bigcup_{PN \in |\mathbf{Nets}|} \{PN\} \times M(PN)$$

is the class of nets from **Nets** with collective monoid markings defined by M. For a marked net $NI = (PN, I, m) \in |\mathcal{IS}_{fin}|$ with finite token set I, $Coll_{\mathcal{IS}}$ yields the same net with a monoid marking

$$Coll_{\mathcal{IS}}(NI) = (PN, \mu) \text{ where } \mu = \sum_{i \in I} m(i) \in M(PN)^{\oplus}$$

Note that we may denote the collective marking of $Coll_{\mathcal{IS}}(PN)$ (or short $Coll(PN)$ if $\mathcal{IS}$ is understood from the context) alternatively as the sum with explicit coefficients $\mu = \sum\limits_{x \in M(PN)} |m^{-1}(x)| \cdot x$. △

Example. The following collection construction flattens a PTI net to a P/T system with collective marking by forgetting the individuality of token elements. According to Theorem 4.3.3, we have $\mathbf{PTINets} \cong \mathbf{ISystems}(\mathbf{PTNets}, M_{PT})$ for PTI nets with a marking functor $M_{PT} \colon \mathbf{PTNets} \to \mathbf{Sets}$ with $M(P, T, pre, post) = P$. It is easy to verify that $CSystems(\mathbf{PTNets}, M_{PT}) = \mathbf{PTSys}$. We get the collection construction for PTI nets with finite markings

$$Coll_{\mathbf{PTINets}}(PN, I, m) = \left(PN, \sum_{i \in I} m(i) \in P_{PN}^{\oplus} \right)$$

In the following, we may denote $Coll_{\mathbf{PTINets}}$ also short by $Coll_{PT}$.

Similarly, by Theorem 4.3.4 we have for AHLI nets that

$$\mathbf{AHLINets} \cong \mathbf{ISystems}(\mathbf{AHLNets}_{\emptyset}, M_{AHL})$$

with the marking functor $M_{AHL} \colon \mathbf{AHLNets}_{\emptyset} \to \mathbf{Sets}$ where

$$M_{AHL}(SP, P, T, pre, post, cond, type, A) = A \otimes P).$$

It is easy to verify that $CSystems(\mathbf{AHLNets}, M_{AHL}) = \mathbf{AHLSys}$. We get the collection construction for AHLI nets with finite markings

$$Coll_{\mathbf{AHLINets}}(AN, I, m) = \left(AN, \sum_{i \in I} m(i) \in (A_{AN} \otimes P_{AN})^{\oplus} \right)$$

In the following, we may denote $Coll_{\mathbf{AHLINets}}$ also short by $Coll_{AHL}$.

It is easy to see that every P/T system in the collective approach as defined in [EHP+07] corresponds to a PTI net. We show that this property hols for arbitrary collection constructions.

Lemma 4.4.2 (Collective Constructions are Surjective)
Given a category $\mathcal{IS} = \mathbf{ISystems}(\mathbf{Nets}, M)$ of individual systems, for every net $PN \in |\mathbf{Nets}|$ with a collective marking $\mu \in M(PN)^{\oplus}$, there exists an individual system $NI \in |\mathcal{IS}_{fin}|$ with $Coll_{\mathcal{IS}}(NI) = (PN, \mu)$.

Proof. For $\mu = \sum\limits_{x \in M(PN)} \lambda_x \cdot x$, consider for each $x \in M(PN)$ a set I_x of $\lambda_x \in \mathbb{N}$ individuals with all I_x being mutually disjoint. We choose $NI = (PN, I, m)$ with $I = \bigcup\limits_{x \in M(PN)} I_x$ and $m \colon I \to M(PN)$ with $m(i) = x$ for $i \in I_x$. Hence, $Coll_{\mathcal{IS}}(NI) = (PN, \hat{\mu})$ with

$$\hat{\mu} = \sum_{i \in I} m(i) = \sum_{x \in M(PN)} \sum_{i \in I_x} m(i) = \sum_{x \in M(PN)} \lambda_x \cdot x = \mu$$

□

Remark. As shown in Lemma 4.4.2, for any net with collective marking $(PN, \mu) \in CSystems(\mathbf{Nets}, M)$ there is some corresponding individual system $NI \in |\mathcal{IS}|$. For this reason, in the following we may simply write $Coll_{\mathcal{IS}}(\mathcal{IS})$ (or short $Coll(\mathcal{IS})$) instead of $CSystems(\mathbf{Nets}, M)$ for the class of collective nets over **Nets** and M.

Next, we define for a category $\mathcal{IS}$ of individual systems an equivalence relation $\approx_{\mathcal{IS}}$ on its marked nets and show that two nets are equivalent if and only if they correspond to the same system with collective marking.

Definition 4.4.3 (Equivalence of Individual Petri Systems)
We call two marked nets $NI = (PN, I, m)$ and $NI' = (PN', I', m')$ of a category $\mathcal{IS}$ of individual systems *equivalent* and write $NI \approx_{\mathcal{IS}} NI'$ if $PN = PN'$ and if there exists a bijective function $f \colon I \to I'$ with $m' \circ f = m$.

Note that because bijective functions are closed under composition and inversion, $\approx_{\mathcal{IS}}$ (or short $\approx$ if $\mathcal{IS}$ is understood from the context) is an equivalence relation. △

Lemma 4.4.4 (Equivalence and Collective Equality)
Given a category $\mathcal{IS} = \mathbf{ISystems}(\mathbf{Nets}, M)$ of individual systems, for any two marked nets $NI = (PN, I, m)$ and $NI' = (PN', I', m')$ of $\mathcal{IS}_{fin}$ (with finite markings) the following equivalence holds:

$$Coll_{\mathcal{IS}}(NI) = Coll_{\mathcal{IS}}(NI') \Leftrightarrow NI \approx_{\mathcal{IS}} NI'$$

Proof. We assume $Coll_{\mathcal{IS}}(NI) = (PN, \mu)$ and $Coll_{\mathcal{IS}}(NI') = (PN', \mu')$.
"$\Rightarrow$": From $Coll_{\mathcal{IS}}(NI) = Coll_{\mathcal{IS}}(NI')$, we get $PN = PN'$ and

$$\sum_{i \in I} m(i) = \mu = \mu' = \sum_{i \in I'} m'(i)$$

We construct a bijection $f \colon I \to I'$ compatible with m and m'. Choose for each marking element $x \in M(PN)$ an arbitrary bijection $f_x \colon m^{-1}(x) \to m'^{-1}(x)$ between the subsets of tokens of I and I' that are mapped to x by m and m', respectively. Such bijections exist because from $\mu = \mu'$, we get by the equality of their coefficients for all $x \in M(PN)$ that $|m^{-1}(x)| = |m'^{-1}(x)|$. Consider the function $f \colon I \to I'$ with $f(i) = f_x(i)$ for $i \in m^{-1}(x)$, which is well-defined because $I = \bigcup\limits_{x \in M(PN)} m^{-1}(x)$, $I' = \bigcup\limits_{x \in M(PN)} m'^{-1}(x)$, and the preimage subsets of m and m' are disjoint. Moreover, f is bijective and for all $x \in M(PN)$ and all $i \in m^{-1}(x)$, we have $m' \circ f(i) = m'(f_x(i)) = x = m(i)$, from which we conclude $NI \approx_{\mathcal{IS}} NI'$.

"$\Leftarrow$": From $NI \approx_{\mathcal{IS}} NI'$, we get $PN = PN'$ and bijective $f\colon I \to I'$ with $m' \circ f = m$. We show that $\mu = \mu'$:

$$\begin{aligned} \mu &= \sum_{i\in I} m(i) = \sum_{i \in I} m' \circ f(i) = \sum_{x \in M(PN)} |(m' \circ f)^{-1}(x)| \cdot x \\ &= \sum_{x \in M(PN)} |f^{-1} \circ m'^{-1}(x)| \cdot x \overset{f \text{ bij.}}{=} \sum_{x \in M(PN)} |m'^{-1}(x)| \cdot x = \sum_{i \in I'} m'(i) = \mu' \end{aligned}$$ □

Theorem 4.4.5 (Correspondence of Individual and Collective Markings)
Given a category $\mathcal{IS} = \mathbf{ISystems}(\mathbf{Nets}, M)$ of individual systems, the quotient $|\mathcal{IS}_{fin}|/_{\approx_{\mathcal{IS}}}$ of the class of all individual systems of $\mathcal{IS}$ with finite markings by their equivalence relation corresponds bijectively to the class $CSystems(\mathbf{Nets}, M)$ of all nets with collective markings over **Nets** *and M.*

$$\begin{tikzcd}
|\mathcal{IS}_{fin}| \arrow[rr, "Coll_{\mathcal{IS}}"] \arrow[dr, "n"'] & & CSystems(\mathbf{Nets}, M) \\
& |\mathcal{IS}_{fin}|/_{\approx_{\mathcal{IS}}} \arrow[ur, "i"'] &
\end{tikzcd}$$

Proof. Consider the function $i\colon |\mathcal{IS}_{fin}|/_{\approx_{\mathcal{IS}}} \to CSystems(\mathbf{Nets}, M)$ with $i([NI]_{\approx_{\mathcal{IS}}}) = Coll_{\mathcal{IS}}(NI)$ for $NI \in |\mathcal{IS}_{fin}|$. Note that $i \circ n = Coll_{\mathcal{IS}}$, where n is the natural function mapping an individual system to its equivalence class. By Lemma 4.4.4, we get that i is well-defined and injective because all individual systems in the same equivalence class w. r. t. $\approx_{\mathcal{IS}}$ have the same collective construction image to which only elements of this particular equivalence class are mapped by $Coll_{\mathcal{IS}}$ and i, respectively. By Lemma 4.4.2, $Coll_{\mathcal{IS}}$ is surjective. Therefore, also i is surjective and hence bijective. □

Corollary 4.4.6 (Correspondence of PTI Nets and P/T Systems)
The quotient $|\mathbf{PTINets}_{fin}|/_{\approx}$ of the class of all PTI nets with finite markings corresponds bijectively to the class $|\mathbf{PTSys}|$ of all P/T systems.

Corollary 4.4.7 (Correspondence of AHLI Nets and AHL Systems)
The quotient $|\mathbf{AHLINets}_{fin}|/_{\approx}$ of the class of all AHLI nets with finite markings corresponds bijectively to the class $|\mathbf{AHLSys}|$ of all AHL systems.

PTI and AHLI nets are very close to their *collective* pendants w. r. t. firing behavior. For every finite individual marking there exists an equivalent collective marking, i. e., the net can perform the same firing steps (up to token selections), and vice versa. For the concrete cases of PTI and AHLI nets, we show in the following theorems that the firing behavior of the individual approach is compatible with the well-known firing behavior of the collective approach because a firing step in one representation implies a firing step in the respective other one.

Theorem 4.4.8 ($Coll_{PT}$ preserves and reflects Firing Behavior)

1. *Given a PTI net NI with finite marking and a transition $t \in T_{NI}$ enabled under token selection $S = (M, m, N, n)$ with firing step $NI \xrightarrow{t,S} NI'$, then t is enabled in $Coll_{PT}(NI)$ with firing step $Coll_{PT}(NI) \xrightarrow{t} Coll_{PT}(NI')$.*
2. *Vice versa, given an enabled transition t in $Coll_{PT}(NI)$ with firing step $Coll_{PT}(NI) \xrightarrow{t} (PN', \mu')$, there exists a token selection $S = (M, m, N, n)$ such that t is enabled in NI under S with firing step $NI \xrightarrow{t,S} NI'$ and $Coll_{PT}(NI') = (PN', \mu')$.*

Proof. Assume $NI = (P, T, pre, post, I, m)$ and $Coll_{PT}(NI) = (P, T, pre, post, \mu)$.

1. Transition t is enabled in $Coll_{PT}(NI)$ under μ because

$$pre(t) \overset{t \text{ enabled}}{=} \sum_{i \in M} m(i) \overset{M \subseteq I}{\leq} \sum_{i \in I} m(i) = \mu$$

Firing changes just the markings, so we have $NI' = (P, T, pre, post, I', m')$ and $Coll_{PT}(NI') = (P, T, pre, post, \mu')$. We show that μ' is the marking resulting from firing t in $Coll_{PT}(NI)$.

$$\begin{aligned}
&\mu \ominus pre(t) \oplus post(t) \\
&= \sum_{i \in I} m(i) \ominus \sum_{i \in M} m(i) \oplus \sum_{i \in N} n(i) && (Coll_{PT}, (t, S) \text{ enabled in } NI) \\
&= \sum_{i \in I \setminus M} m(i) \oplus \sum_{i \in N} n(i) \\
&= \sum_{i \in (I \setminus M) \uplus N} m'(i) && (m' \text{ as in Def. 4.1.2}) \\
&= \sum_{i \in I'} m'(i) = \mu' && (Coll_{PT} \text{ and } I' \text{ as in Def. 4.1.2})
\end{aligned}$$

2. Because transition t is enabled in $Coll_{PT}(NI)$ and we have $pre(t) \leq \mu = \sum_{i \in I} m(i)$, we can choose for each $p \in P$ a subset $M_p \subseteq m^{-1}(p)$ such that $|M_p| = pre(t)(p)$. Note that $m(x) = p$ for $x \in M_p$. Similarly, we choose for each $p \in P$ a set N_p such that $|N_p| = post(t)(p)$ and all N_p being mutually disjoint and disjoint to $I \setminus \bigcup_{p \in P} M_p$.

   Consider the selection $S = (M, m, N, n)$ with $M = \bigcup_{p \in P} M_p$, $N = \bigcup_{p \in P} N_p$, and function $n \colon N \to P$ with $n(x) = p$ for $x \in N_p$. The transition t is enabled in NI under S because $\sum_{i \in M} m(i) = \sum_{p \in P} \sum_{i \in M_p} m(i) = \sum_{p \in P} |M_p| \cdot p = \sum_{p \in P} pre(t)(p) \cdot p = pre(t)$ and analogously for N, n, and $post$. For the firing step $NI \xrightarrow{t,S} NI'$, we have $NI' = (P, T, pre, post, I', m')$ according to Def. 4.1.2 and $PN' = (P, T, pre, post)$ because firing changes the marking, only. We show for $Coll_{PT}(NI') = (PN', \hat{\mu})$ that $\mu' = \hat{\mu}$. The arguments are analogous to the ones for the equations of item 1.

$$\begin{aligned}\mu' &= \mu \ominus pre(t) \oplus post(t) = \sum_{i\in I} m(i) \ominus \sum_{i\in M} m(i) \oplus \sum_{i\in N} n(i) \\ &= \sum_{i\in I\setminus M} m(i) \oplus \sum_{i\in N} n(i) = \sum_{i\in (I\setminus M)\uplus N} m'(i) = \sum_{i\in I'} m'(i) = \hat{\mu}\end{aligned}$$ □

Remark. Because the only condition for t to be enabled in $Coll_{PT}(NI)$ is that $\forall p \in P\colon pre(t)(p) \leq \left(\sum_{i\in I} m(i)\right)(p)$, there are possibly many different valid token selections corresponding to the same firing step of t in $Coll_{PT}(NI)$, depending only on isomorphic N and n.

With all the results at hand, we can show that the equivalence relation $\approx$ on PTI nets characterizes equivalent firing behavior.

Theorem 4.4.9 (Equivalent Firing Behavior of PTI Nets)
Given PTI nets $NI_1 \approx NI_2$ with finite markings and a firing step $NI_1 \xrightarrow{t,S} NI_1'$, then there is a corresponding firing step $NI_2 \xrightarrow{t,S'} NI_2'$ with $NI_1' \approx NI_2'$.

Proof. By Lemma 4.4.4, we have $Coll_{PT}(NI_1) = (PN, \mu) = Coll_{PT}(NI_2)$ and by Theorem 4.4.8 there is a firing step $(PN, \mu) \xrightarrow{t} (PN, \mu') = Coll_{PT}(NI_1')$, implying a reflected step $NI_2 \xrightarrow{t,S'} NI_2'$ with $Coll_{PT}(NI_2') = (PN, \mu') = Coll_{PT}(NI_1')$. Hence, by Lemma 4.4.4 there is $NI_1' \approx NI_2'$. □

The result that $Coll_{PT}$ preserves and reflects firing behavior and the characterization of firing equivalence for PTI nets can be lifted to $Coll_{AHL}$ and AHLI nets.

Theorem 4.4.10 ($Coll_{AHL}$ preserves and reflects Firing Behavior)

1. *Given an AHLI net ANI with finite marking and a consistent transition assignment $(t, asg) \in CT_{ANI}$ enabled under token selection $S = (M, m, N, n)$ with firing step $ANI \xrightarrow{t,asg,S} ANI'$, then (t, asg) is enabled in $Coll_{AHL}(ANI)$ with firing step $Coll_{AHL}(ANI) \xrightarrow{t,asg} Coll_{AHL}(ANI')$.*
2. *Vice versa, given an enabled transition assignment (t, asg) in $Coll_{AHL}(ANI)$ with $Coll_{AHL}(ANI) \xrightarrow{t,asg} (AN', \mu')$, there exists a token selection $S = (M, m, N, n)$ such that (t, asg) is enabled in ANI under S with firing step $ANI \xrightarrow{t,asg,S} ANI'$ and $Coll_{AHL}(ANI') = (AN', \mu')$.*

Proof. Assume $ANI = (SP, P, T, pre, post, cond, type, A, I, m)$ and
$Coll_{AHL}(ANI) = (SP, P, T, pre, post, cond, type, A, \mu)$.

1. The transition assignment (t, asg) is enabled in $Coll_{AHL}(ANI)$ under μ because $pre_A(t, asg) \overset{(t,asg)\text{ enabled}}{=} \sum_{i \in M} m(i) \overset{M \subseteq I}{\leq} \sum_{i \in I} m(i) = \mu$. Firing changes just the markings, so we have $ANI' = (SP, P, T, pre, post, cond, type, A, I', m')$ and $Coll_{AHL}(ANI') = (SP, P, T, pre, post, cond, type, A, \mu')$. We show that μ' is the marking resulting from firing (t, asg) in $Coll_{AHL}(ANI)$ under S.

$$\begin{aligned}
&\mu \ominus pre_A(t, asg) \oplus post_A(t, asg) \\
&= \sum_{i \in I} m(i) \ominus \sum_{i \in M} m(i) \oplus \sum_{i \in N} n(i) && (Coll_{AHL}, (t, asg, S) \text{ enabled in } ANI) \\
&= \sum_{i \in I \setminus M} m(i) \oplus \sum_{i \in N} n(i) \\
&= \sum_{i \in (I \setminus M) \uplus N} m'(i) && (m' \text{ as in Def. 4.2.2}) \\
&= \sum_{i \in I'} m'(i) = \mu' && (Coll_{AHL} \text{ and } I' \text{ as in Def. 4.1.2})
\end{aligned}$$

2. Because the transition assignment (t, asg) is enabled in $Coll_{AHL}(ANI)$ and we have $pre_A(t) \leq \mu = \sum_{i \in I} m(i)$, we can choose for each $(a, p) \in A \otimes P$ a subset $M_{a,p} \subseteq m^{-1}(a, p)$ such that $|M_{a,p}| = pre_A(t, asg)(a, p)$. Note that $m(x) = (a, p)$ for $x \in M_{a,p}$. Similarly, we choose for each $(a, p) \in A \otimes P$ a set $N_{a,p}$ such that $|N_{a,p}| = post_A(t, asg)(a, p)$ and all $N_{a,p}$ being mutually disjoint and disjoint to $I \setminus \bigcup_{(a,p) \in A \otimes P} M_{a,p}$. Consider the selection $S = (M, m, N, n)$ with $M = \bigcup_{(a,p) \in A \otimes P} M_{a,p}$, $N = \bigcup_{(a,p) \in A \otimes P} N_{a,p}$, and function $n \colon N \to P$ with $n(x) = (a, p)$ for $x \in N_{a,p}$. The transition assignment (t, asg) is enabled in ANI under S because

$$\begin{aligned}
\sum_{i \in M} m(i) &= \sum_{(a,p) \in A \otimes P} \sum_{i \in M_{a,p}} m(i) = \sum_{(a,p) \in A \otimes P} |M_{a,p}| \cdot (a, p) \\
&= \sum_{(a,p) \in A \otimes P} pre_A(t, asg)(a, p) \cdot (a, p) = pre_A(t, asg)
\end{aligned}$$

and analogously for N, n, and $post_A$. For the firing step $ANI \xrightarrow{t,asg,S} ANI'$, we have $ANI' = (SP, P, T, pre, post, cond, type, A, I', m')$ according to Def. 4.2.2 and $AN' = (SP, P, T, pre, post, cond, type, A)$ because firing changes the marking, only. We show for $Coll_{AHL}(ANI') = (AN', \hat{\mu})$ that $\mu' = \hat{\mu}$. The arguments are analogous to the ones for the equations of item 1.

$$\begin{aligned}
\mu' &= \mu \ominus pre(t) \oplus post(t) = \sum_{i \in I} m(i) \ominus \sum_{i \in M} m(i) \oplus \sum_{i \in N} n(i) \\
&= \sum_{i \in I \setminus M} m(i) \oplus \sum_{i \in N} n(i) = \sum_{i \in (I \setminus M) \uplus N} m'(i) = \sum_{i \in I'} m'(i) = \hat{\mu}
\end{aligned}$$ □

Theorem 4.4.11 (Equivalent Firing Behavior of AHLI Nets)
Given AHLI nets $ANI_1 \approx ANI_2$ with finite markings and a firing step $ANI_1 \xrightarrow{t,asg,S} ANI'_1$, then there is a corresponding firing step $ANI_2 \xrightarrow{t,asg,S'} ANI'_2$ with $ANI'_1 \approx ANI'_2$.

Proof. By Lemma 4.4.4, we have $Coll_{AHL}(ANI_1) = (AN, \mu) = Coll_{AHL}(ANI_2)$ and by Theorem 4.4.10 there is a firing step $(AN, \mu) \xrightarrow{t,asg} (AN, \mu') = Coll_{AHL}(ANI'_1)$, implying a reflected step $ANI_2 \xrightarrow{t,asg,S'} ANI'_2$ with $Coll_{AHL}(ANI'_2) = (AN, \mu') = Coll_{AHL}(ANI'_1)$. Hence, by Lemma 4.4.4 there is $ANI'_1 \approx AeNI'_2$. □

4.5. Functorial Relationships between Net Classes with Individual and Collective Markings

In this subsection, we express the vicinity of PTI nets and AHLI nets to their collective counterparts by extending the concrete collective constructions from Def. 4.4.1 to functors. In addition, we define a flattening functor from AHLI to PTI nets — similar to the flattening of AHL nets — that preserves enabling and firing. Finally, we show a compatibility result for all these functors.

We extend the collection construction $Coll_{PT}$ for PTI nets to a functor $Coll_{PT}\colon \mathbf{PTINets}_I \to \mathbf{PTSys}$. **PTSys** is the category of P/T nets with a marking $M \in P^{\oplus}$, where $P^{\oplus}$ is the free commutative monoid over the set of places. Morphisms in **PTSys** are pairs

$$(f_P\colon P_1 \to P_2, f_T\colon T_1 \to T_2)\colon N_1 \to N_2$$

that are compatible to the nets' *pre* and *post* mappings (just as **PTINets** morphisms) and preserve markings placewise, i. e., $\forall p \in P_1\colon M_{N_1}(p) \leq M_{N_2}(f_P(p))$.

Definition and Fact 4.5.1 (Functor $Coll_{PT}\colon \mathbf{PTINets}_I \to \mathbf{PTSys}$)
We define as domain of the functor the category $\mathbf{PTINets}_I$ of PTI nets with finite markings and token-injective **PTINets** morphisms. The PTI nets are mapped as in the collection construction $Coll_{PT}$ from Def. 4.4.1. For morphisms $f = (f_P, f_T, f_I)\colon NI_1 \to NI_2$, we just have to forget the individual token component by defining

$$Coll_{PT}(f) = (f_P, f_T)\colon Coll_{PT}(NI_1) \to Coll_{PT}(NI_2).$$

Proof. Obviously, $Coll_{PT}(f)$ is well-defined on the net structure parts of $Coll_{PT}(NI_1)$ and $Coll_{PT}(NI_2)$. Furthermore, it holds that

$$\forall p \in P_1\colon M_1(p) = |m_1^{-1}(p)| \leq |m_2^{-1}(f_P(p))| = M_2(f_P(p)).$$

The inequality is valid because each token x on place p implies a unique image $f_I(x)$ on $f_P(p)$ due to the **PTINets** morphism properties and because f_I being injective does not merge token images in I_2.

The compositionality follows directly from the componentwise composition of PTI net morphisms. □

Now, we extend the collection construction Coll_{AHL} to a functor $\mathit{Coll}_{AHL}\colon \mathbf{AHLINets}_I \to \mathbf{AHLSys}$. **AHLSys** is the category of AHL nets with a marking $M \in (A \otimes P)^{\oplus}$, where $(A \otimes P)^{\oplus}$ is the free commutative monoid over pairs of values from the net's algebra and the net's places of compatible type. Morphisms in **AHLSys** are tuples

$$(f_\Sigma\colon \Sigma_1 \to \Sigma_2, f_P\colon P_1 \to P_2, f_T\colon T_1 \to T_2, f_A\colon A_1 \to A_2)\colon AN_1 \to AN_2$$

that comply to all compatibility properties of AHLI net morphisms (of course, except the one regarding the individual token component) and that preserve markings place–valuewise, i. e.,

$$\forall (a,p) \in A_1 \otimes P_1\colon M_{AN_1}(a,p) \leq M_{AN_2}(f_A \otimes f_P(a,p)).$$

Definition and Fact 4.5.2 (Functor Coll_{AHL}: AHLINets$_I$ → AHLSys)
We define as domain of the functor the category $\mathbf{AHLINets}_I$ of AHLI nets with finite marking and AHLI net morphisms that are injective on tokens and variables[20]. The AHLI nets are mapped as in the collection construction Coll_{AHL} from Def. 4.4.1. For morphisms $f = (f_\Sigma, f_P, f_T, f_I, f_A)\colon ANI_1 \to ANI_2$, we just forget the individual token component by defining

$$\mathit{Coll}_{AHL}(f) = (f_\Sigma, f_P, f_T, f_A)\colon \mathit{Coll}_{AHL}(ANI_1) \to \mathit{Coll}_{AHL}(ANI_2).$$

Proof. Obviously, $\mathit{Coll}_{AHL}(f)$ is well-defined on the net structure parts of $\mathit{Coll}_{AHL}(ANI_1)$ and $\mathit{Coll}_{AHL}(ANI_2)$. All properties for AHL net morphisms are already valid for the corresponding components of AHLI net morphisms.

The compositionality follows directly from the componentwise composition of AHLI net morphisms. □

Definition and Fact 4.5.3 (Functor Flat_I: AHLINets$_\mathbf{I}$ → PTINets$_\mathbf{I}$)
The following construction flattens an AHLI net to a PTI net:

$$\mathit{Flat}_I(AN, I, m) = (CP, CT, pre_A, post_A, I, m)$$

Note that $(CP, CT, pre_A, post_A) = \mathit{Flat}(AN)$ and $m\colon I \to (A \otimes P) = CP$.

To extend the construction to a functor $\mathit{Flat}_I\colon \mathbf{AHLINets_I} \to \mathbf{PTINets_I}$, we define for each $f = (f_\Sigma, f_P, f_T, f_A, f_I)\colon ANI_1 \to ANI_2$ the flattening

$$\mathit{Flat}_I(f) = (f_A \otimes f_P, f_T \times \left(f_A \circ \_ \circ f^{-1}_{\Sigma|Var(t)}\right), f_I)\colon \mathit{Flat}_I(ANI_1) \to \mathit{Flat}_I(ANI_2).$$

Proof. To prove that

$$\mathit{Flat}_I(f\colon ANI_1 \to ANI_2)\colon \mathit{Flat}_I(ANI_1) \to \mathit{Flat}_I(ANI_2)$$

[20] We need injectivity on variables of a signature for the flattening of AHLI net morphisms.

is a valid PTI net morphism, we have to show the following equalities:

$$f_I \circ m_1 = m_2 \circ (f_A \otimes f_P) \quad \text{and}$$

$$(f_A \otimes f_P)^{\oplus} \circ pre_{A1} = pre_{A2} \circ \left(f_T \times \left(f_A \circ \_ \circ f^{-1}_{\Sigma|Var(t)}\right)\right),$$

analogously for $post_A$. The first one follows directly from f being a AHLI net morphism.

For the second one, we have for all $(t, asg) \in CT_1$ that

$$\begin{aligned}
& pre_{A2} \circ \left(f_T \times \left(f_A \circ \_ \circ f^{-1}_{\Sigma|Var(t)}\right)\right)(t, asg) \\
&= pre_{A2}\left(f_T(t), f_A \circ asg \circ f^{-1}_{\Sigma|Var(t)}\right) \\
&= pre_{A2}\left(f_T(t), asg_f\right) && \text{(def. } asg_f\text{)} \\
&= (\overline{asg_f}, id_{P_2})^{\oplus} \circ pre_2 \circ f_T(t) && \text{(def. } pre_{A2}\text{)} \\
&= (\overline{asg_f}, id_{P_2})^{\oplus} \circ (f^{\#}_{\Sigma} \otimes f_P)^{\oplus} \circ pre_1(t) && \text{(}f\text{ AHLI morph.)} \\
&= \sum_{i=1}^{n} \left(\overline{asg_f} \circ f^{\#}_{\Sigma}(term_i), f_P(p_i)\right) && \text{for } \sum_{i=1}^{n}(term_i, p_i) = pre_1(t) \\
&= \sum_{i=1}^{n} (f_A \circ \overline{asg}(term_i), f_P(p_i)) && \text{(Lemma A.3.2)} \\
&= (f_A \otimes f_P)^{\oplus} \circ (\overline{asg}, id_{P_1})^{\oplus} \circ pre_1(t) \\
&= (f_A \otimes f_P)^{\oplus} \circ pre_{A1}(t, asg) && \text{(def. } pre_{A1}\text{)}
\end{aligned}$$

and analogously for $post_A$.

The compositionality follows directly from the componentwise composition of AHLI net morphisms. □

Theorem 4.5.4 (*Flat$_I$* preserves and reflects Enabling and Firing)
Given an AHLI net $ANI = (AN, I, m)$ with $Flat_I(AN, I, m) = (CN, I, m)$ and a token selection $S = (M, m, N, n)$, the following holds:

1. *$(t, asg) \in CT$ is enabled under S in ANI, iff (t, asg) is enabled under S in $Flat_I(ANI)$.*
2. *$ANI \xrightarrow{t,asg,S} (AN, I', m')$ via $S \quad \Leftrightarrow \quad (CN, I, m) \xrightarrow{t,asg,S} (CN, I', m')$ via S.*

Proof.

1. $(t, asg) \in CT$ is enabled under S in (AN, I, m)
$\Leftrightarrow \sum_{i \in M} m(i) = pre_A(t, asg) \wedge \sum_{i \in N} n(i) = post_A(t, asg)$
$\Leftrightarrow (t, asg) \in CT$ is enabled under S in $Flat_I(AN, I, m) = (CP, CT, pre_A, post_A, I, m)$
2. $(AN, I, m) \xrightarrow{t,asg,S} (AN, I', m')$ via S
$\Leftrightarrow (t, asg)$ enabled under S in both ANI and $Flat_I(ANI)$,
$$\text{and } I' = (I \setminus M) \uplus N, \quad m'(x) = \begin{cases} m(x), & x \in I \setminus M \\ n(x), & x \in N \end{cases}$$
$\Leftrightarrow Flat_I(AN, I, m) = (CN, I, m) \xrightarrow{t,asg,S} (CN, I', m') = Flat_I(AN, I', m')$ via S □

Theorem 4.5.5 (Compatibility of Collection and Flattening Functors) *The previously defined collection and flattening functors are compatible, i. e., $Coll_{PT} \circ Flat_I = Flat \circ Coll_{AHL}$.*

$$\begin{tikzcd}
\mathbf{AHLINets_I} \arrow[r, "Flat_I"] \arrow[d, "Coll_{AHL}"'] \arrow[dr, phantom, "="] & \mathbf{PTINets_I} \arrow[d, "Coll_{PT}"] \\
\mathbf{AHLSys} \arrow[r, "Flat"] & \mathbf{PTSys}
\end{tikzcd}$$

Proof. Given an AHLI net

$$ANI = (\Sigma, P, T, pre, post, cond, type, A, I, m) \in \mathbf{AHLINets}_I, \text{ we have}$$

$$\begin{aligned}
&Coll_{PT} \circ Flat_I(ANI) \\
&= Coll_{PT}(CP, CT, pre_A, post_A, I, m) && \text{(def. } Flat_I) \\
&= \left(CP, CT, pre_A, post_A, M = \sum_{i \in I} m(i) \in (A \otimes P)^{\oplus} \right) && \text{(def. } Coll_{PT}) \\
&= Flat \left(\Sigma, P, T, pre, post, cond, type, A, M = \sum_{i \in I} m(i) \right) && \text{(def. } Flat) \\
&= Flat \circ Coll_{AHL}(ANI) && \text{(def. } Coll_{AHL})
\end{aligned}$$

and for some AHL net morphism $f = (f_\Sigma, f_P, f_T, f_A, f_I) \colon ANI_1 \to ANI_2$ with injective f_I we have

$$\begin{aligned}
&Coll_{PT} \circ Flat_I(f) \\
&= Coll_{PT}(f_A \otimes f_P, f_T \times \left(f_A \circ \_ \circ f^{-1}_{\Sigma|Var(t)} \right), f_I) && \text{(def. } Flat_I) \\
&= (f_A \otimes f_P, f_T \times \left(f_A \circ \_ \circ f^{-1}_{\Sigma|Var(t)} \right)) && \text{(def. } Coll_{PT}) \\
&= Flat(f_\Sigma, f_P, f_T, f_A) && \text{(def. } Flat) \\
&= Flat \circ Coll_{AHL}(f) && \text{(def. } Coll_{AHL})
\end{aligned}$$

□

4.6. Algebraic Higher-Order Nets

In Sect. 3.2.7, we mention the need for a control of rule application and firing steps w. r. t. our modeling principle that reconfigurations are somehow triggered by firing steps (cf. Sect. 3.2.6). Anyway, in reconfigurable Petri nets the concepts of Petri net behavior via transition firing and Petri net reconfiguration with transformation rules are only loosely coupled; the formalism of reconfigurable AHLI nets itself is more similar to graph transformation systems in the sense that rules are assumed to be applied "at the right time". So, we need a possibility to express that needed reconfigurations must be processed immediately, i. e., a control structure for the firing behavior and reconfiguration of AHLI nets. Several formalisms for controlling graph transformations have been proposed [HEET99, KK99] but due to their orientation to graph transformation they lack the possibility to relate reconfigurations (by rule applications) to firing steps.

A useful approach for controlling firing steps and transformations of low-level Petri nets has been introduced in [HEM05] with AHL nets that contain P/T nets and transformation rules as tokens. We call such nets algebraic higher-order (AHO) nets due to the "nets on nets" structure. AHO nets have two important aspects that distinguish them from the original notion of nested Petri nets called object nets in [Val98]. First, AHO nets do not only have Petri nets as tokens but also transformation rules, which qualifies them especially for controlling reconfigurable Petri nets. Second, AHOs are formulated as regular AHL nets with signatures and algebras that provide operations for firing the token nets and for applying token rules on token nets. This means, that we do not need a new formalism and can use all results for AHO nets that we already have for AHL nets. Moreover, we can use any kind of Petri nets and their corresponding transformation rules as tokens, as long as we can define their firing behavior and transformation algebraically for the containing AHO net.

We see that AHO nets are not a new net class in the strict sense of a formal definition, but rather a methodology of using AHL nets to control arbitrary kind of Petri nets, their firing behavior and reconfiguration by rules. We can easily transfer this methodology to the reconfigurable AHLI nets of Sect. 4.2 to model algebraic higher-order nets with individual tokens (AHOI nets) for controlling any kind of reconfigurable Petri nets that can be specified in the underlying algebra of the AHOI net; both are equivalent up to the representation of their tokens.

With AHOI[21] nets, relating firing behavior and reconfigurations of a token net becomes a matter of modeling with AHLI nets. This approach is more flexible than to give ad-hoc definitions that restrict the possible configuration changes of a reconfigurable Petri net in order to force the platform to perform the triggered reconfigurations immediately after the triggering firing step.

Here, we just want to exemplify and illustrate the principle of AHO(I) nets on an informal level and go into the formal details later when we adapt this technique for Communication Platform models. We shortly recapitulate the example from [HEM05] that simulates reconfigurable Petri nets — i. e., a Petri net with a set of transformation rules without restrictions on rule applications and firing steps — as an algebraic higher-order net.

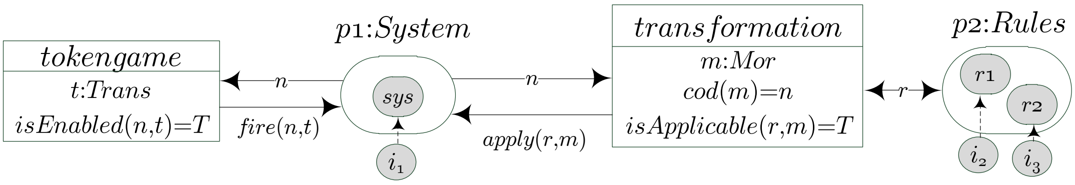


Figure 4.11.: Algebraic higher-order net for reconfigurable P/T nets

[21] Perhaps the reader finds the reference to AHOIs in the picture on this book's cover page.

Fig. 4.11 depicts the AHO net structure for reconfigurable Petri nets. It can carry marked P/T nets — as tokens of the sort *System* — on its place p_1 and Petri net transformation rules — as tokens of the sort *Rules* — on its place p_2. To let the AHO net simulate the reconfigurable Petri net $(sys, \{r_1, r_2\})$ consisting of a marked P/T net *sys* and a set with two rules r_1 and r_2, we choose as its initial marking the net *sys* as token on the net place p_1 and the rules r_1 and r_2 as tokens on the rule place p_2.

For simulating a firing step of a net token on p_1, the AHO net fires its transition *tokengame*, which selects a net token n from p_1 if this net contains some enabled transition, which is ensured by the firing condition $\mathit{enabled}(n, t) = T$ of the higher-order transition. The firing of *tokengame* then produces a token with value $\mathit{fire}(n, t)$ that is the net n where t has fired.

Rule applications are realized similarly by firing the higher-order transition *transformation*, which selects a net token n from p_1 and a rule token r from p_2 if the rule is applicable at some match m into net n. The applicability is ensured algebraically by the firing condition of the transition *transformation*, where the equation $\mathit{cod}(m) = n$ is true if the codomain of the match m is n. Firing *transformation* then produces the token net that results from the application of the selected rule r on this match m, which is calculated algebraically by $\mathit{apply}(r, m)$.

5. Transformation Systems for Modeling Communication Platforms

That which is static and repetitive is boring.
That which is dynamic and random is confusing.
In between lies art.

(John A. Locke)

With the formal approach of AHLI nets and their rule-based transformation at hand (cf. Sect. 4.2), we further develop and concretize the modeling principles from Sect. 3.2.6 to this particular approach in Sect. 5.1. The approach of transformations of algebraic high-level nets with individual tokens (AHLI nets) is the formal basis that we tailor to Communication Platform models in Sect. 5.3 by extending it with advanced transformation concepts that we introduce and adapt in Sect. 5.2 in order to deal with the principal challenges of reconfigurable Petri nets that we gathered in Sect. 3.2.7. Moreover, we use higher-order nets with a specially suited algebra as a structure for the behavioral semantics of a Communication Platform model in Sect. 5.4, relating and controlling the possible firing steps and triggered reconfigurations of an AHLI platform net.

5.1. Basic Concepts for Modeling Communication Platforms as Reconfigurable AHLI Nets

In the following, we motivate, develop, and discuss choices of methods for several aspects of modeling Communication Platforms with AHLI net transformation systems to further concretize the preliminary modeling principles from Sect. 3.2.6. For parts of the discussion and justifications for design choices, we refer to the concrete example of Skype as a Communication Platform as we have described it in detail in Sect. 3.1. Later, we rely on these modeling "building blocks" as explanations of the general principles we use in the case study in Chap. 7.

5.1.1. General Concepts

First, we discuss the general approaches not related to reconfiguration for modeling Communication Platforms. These modeling concepts are based on assumptions on the structure of the AHLI net representing the whole system and its components representing the actors as well as on general modeling techniques for high-level nets.

5.1.1.1. Actions and their owners

In order to model AHLI nets for the Communication Platforms and its clients that comply to the first three principles of Sect. 3.2.6 (clients as components of a single net, user identity as algebraic values on a distinguished place, implicit owner of transitions), we design the clients as a connected net component around a central identity place carrying an identity token, whose value is some unique identifier, possibly a string representing the corresponding user name. Every transition that is supposed to represent an action of this user is connected via an incoming and an outgoing arc with this identity place so that the transition consumes and reproduces the identity token when firing. But this might still not be enough for identifying the owner of a transition: e. g., if we model the effect of contact exchange in Skype (see Sect. 3.1.1) with a transition for accepting the exchange request that copies the names of the asked and the asking owner (the first being the owner) crosswise to their contact lists, this transition has to be connected to both user's identity places to access their identity tokens. In such cases, we also need to distinguish the active user — whose accepting action the fired transition represents — from the other contact identifiers that this transition might access.

A solution for this is to use reserved variables for inscriptions on the arcs to and from the identifier place of the active user invoking the action. Typically, to let a transition "know" the name of its firing user, it is reasonable to connect this transition to the identity place carrying the identity token with an incoming and an outgoing arc that is just inscribed by a reserved variable u indicating the user name (for an example see Fig. 7.3). Thus, the user's identity is available for further arc inscriptions from or to the firing transition but a firing does not change the client's identifier. In Communication Platforms it practically never occurs that any user action changes the system-internal name of this user. This would otherwise lead to severe inconsistencies with contact lists in other clients.[1]

Some transitions may also be functional without any arcs to the identity place, e. g., as for activating and deactivating the Skype client (if we do not want to log information about the invoking user in some history). But clearly, these superfluous arcs do not change the firing semantics of the AHLI net considerably, so that we can use them as a technical possibility to connect all transitions to the place identifying the user who can perform the actions represented by these transitions. We may need this for the matching of the reconfiguring rules in order to identify net structures "belonging" to a specific user.

Further, we also identify some places as belonging to a specific user as data repositories such as the places carrying history tokens for chats, calls and conferences. But these depend on the context of the channel's structure and should be simple to figure out in concrete cases as we see later.

[1] We could now demand that this pattern be followed strictly in the platform net and in the reconfiguring rules, but such a restriction could also impede the modeling of Communication Platforms with unusual user actions. Instead, we propose a method for formulating consistency properties for a concrete platform model and how to validate them in Sect. 6.1. There, we give some examples for typical properties regarding user identity tokens.

5.1.1.2. Modeling the user's perception of data

In most cases, communication in Petri nets is modeled as distributing tokens to some places (by assuming that the places belong in some way to the involved users) by firing transitions. The algebras of the AHLI nets suggest that we use some type of alphanumeric data to be communicated in our Communication Platform models. Basically, we could choose any type of audible or visual data for this but the choice of simple strings is just convenient for our concrete formalism at hand and for a first approach to modeling.

(A)synchronous vs. (non)exclusive communications Traditional modeling of communication systems distinguishes the notions of synchronous and asynchronous communications. For humans who communicate via electronic systems, we have intuitive ideas about these communication forms, e. g., that visual message-based forms such as chats are asynchronous (because the incoming message is stored and can be read and answered later) and that audible forms such as calls are synchronous (in the sense that they are perceived immediately[2]). Because human-centricity is an essential part of modeling Communication Platforms, we should take this aspect into account when considering these communication modalities. We observe that due to natural physical constraints the average human is capable to (consciously) participate in at most one call at the same time; the same being true for arbitrary forms of synchronous communications that have to be perceived when they are being transmitted. A reasonable concrete Communication Platform has to be designed to not overburden the user's capability of apprehension of data. Hence, in our modeling context, we need to treat synchronous communications as exclusive. On the other hand, asynchronous communications such as chats are nonexclusive in most Communication Platforms; but this is not a necessary assumption like the implication of exclusiveness from synchronicity, because chats can be realized differently (and possibly exclusive) in other concrete systems. Based on these observations, in the following we rather distinguish exclusive from nonexclusive communication modes.

For both communication modes, we need a way to distribute the data to the recipients, which in groups and conferences can be any number of users. According to the discussion about the problem of multicasting in Petri nets in Sect. 3.2.5.1, we present a technique in Sect. 5.2.2 in order to deal with this issue by suitable reconfigurations based on the idea we outlined in [BEE+09].

5.1.1.3. General techniques for high-level Petri nets

Besides the aspects of modeling with Petri nets for Communication Platforms, we also have an inventory of patterns for modeling dependencies of transitions and data in high-level Petri nets as well as for simplifying nets that we shortly review in the following by demonstrating some examples. We do not distinguish collective or individual tokens here because the firing behaviors of the approaches are equivalent if we do not explicitly regard the individuals.

[2] Technically, we can understand this as that a passed message is acknowledged instantly and that the sender can be sure that it has been received.

Actions excluding or depending on other actions Although we are using high-level nets, it is useful to have available a sort with a singleton data set as a simple possibility for modeling sequential dependencies between actions that cannot be expressed by using the algebraic payload data. Places of this data type behave as in low-level place–transition nets with valueless "black" tokens. We use this kind of control places for two cases: First, we use control places to model that one action can occur if and only if another action has occurred before like blocking and unblocking a contact (see the RHS of the rule in Fig. 7.7), and second that of a set of actions at most one can occur like accepting or denying a request for contact exchange (see the RHS of the rule in Fig. 7.10).

Counters and Switches Sometimes, actions depend not only on the presence of resources or on the occurrence of one specific action, but they may depend on one or several occurrences of one of a set of actions. An example for an action that requires this is the deactivation of Skype clients, because a client is not allowed to be still busy in a call nor to have a call on hold or incoming when going offline. This means that for every call that the user of this client started, joined, or is awaited to join, he also has to have quit it and that all calls that he has put on hold have also to be resumed (and then quit) at this time.[3]

The decision of being busy can also be made with complementary places for being busy and being not busy[4] that carry a token in the respective case but it is impossible for a transition to demand the absence of tokens (without extending the Petri net formalism at hand by the notion of inhibitor arcs). Even when counting the ongoing calls (possibly on hold) with black tokens, we cannot enforce the transition for deactivation to fire only when there is no ongoing call (see Fig. 5.1). But if we replace the control place with a place that carries an explicit integer value, the deactivation transition can demand the token value 0 for the number of ongoing calls (see Fig. 5.2). The actions for call and hang up increase and decrease the current value on the counter place, respectively. Note that in this figure, we also simplified the net by replacing the two complementary control places "IsBusy" and "IsIdle" with one Boolean place.

Folding by products of data sorts One of the main advantages of using high-level nets instead of low-level nets is that the marking of one place in a high-level net can represent the same information as many (possibly infinite) places in a low-level net when there is a high-level token value for each of the combined low-level markings. For example, the merging of the two control steps in the last item is such a folding: The two Boolean values F and T (for false and true) on "IsBusy" in Fig. 5.2 can represent the two possible markings of "IsBusy" and "IsIdle" in Fig. 5.1 because they are complementary, i. e., one of them carries exactly one token if and only if the other does not. This classic notion of folding (as an initial step from low- to

[3] Actually, if a user switches his Skype client to the offline status, all calls — even if on hold — will be quit automatically. We further discuss this discrepancy in the modeling case study on Skype in Sect. 7.1.4.4.

[4] The PTI net in Fig. 4.2 models such a client but it is limited to a single call.

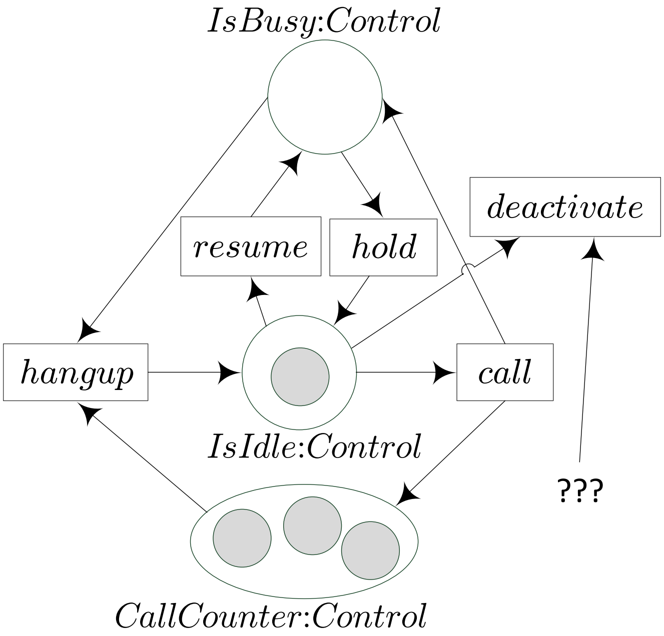


Figure 5.1.: Low-level control structure is not sufficient for realizing deactivation demanding no calls on hold

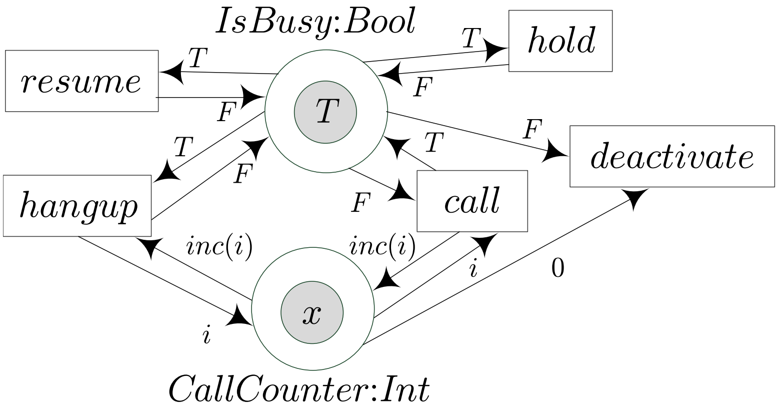


Figure 5.2.: Using integer token values for counting the calls and merging the complementary places for busy and idle to one Boolean place

high-level nets) can also be used to further simplify a high-level net's structure if the information on some places is strongly coupled as the busy switch and the hold counter in Fig. 5.2. If we extend the signature of this net by a sort for the product of both sorts, we can simplify this net to the equivalent one in Fig. 5.3. We use this kind of data token for this status information in the case study (see Fig. 7.3).

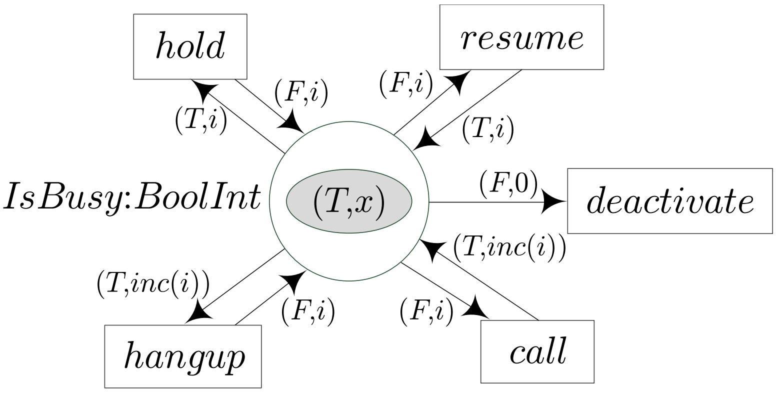


Figure 5.3.: Folding the Boolean and the counting place by their sort product

5.1.2. Reconfiguration related to Firing Behavior

If we follow the principle “every action is a transition”, then everything in our platform model is about managing the variable set of transitions a user “owns” such that their activation status corresponds to the status of this user’s software client. Every time when the invocation of an action by the user in its client brings up new possible actions to or removes actions from the user interface, we need to capture this change with transformations by defining one or more transformation rules for this particular case causing the appropriate result. Note that we already consider it a change if only the preconditions or effect of an action change due to invocation of another action.

Note that, when using a concrete Communication Platform, one may notice that specific effects can be achieved in different ways in the user interface. If we carefully examine the effects that menus and buttons should realize, we can simplify the task of modeling by identifying similar or identical effects. But we discuss such simplifications in the case study on Skype in Sect. 7.1.4 because they highly depend on the concrete model at hand.

5.1.2.1. Tokens announcing requests for reconfiguration

An intuitive way to let the application of reconfiguring rules depend on firing of transitions (according to our modeling principles in Sect. 3.2.6) is to let the rules — in order to be applicable — demand a distinguished kind of “request tokens”. Consider in the combined firing and transformation sequence of AHLI nets below the triggering firing step, which produced a request token carrying the necessary information for the system to realize the request.

$$ANI_0 \xrightarrow{trigger} ANI'_0 \xRightarrow{handling}{}^{*} ANI_1$$

From now on, we consider plain transition firings and such sequences of triggering firing steps together with a suitable transformation sequence as a *user action*.[5] If we

[5] We give a formal definition of user actions in Def. 5.3.1 when we also define Communication Platform models.

have sorts and constructors for request tokens (which typically are tuple types wrapping up identifier and communication data values) in the signature and algebra of the AHLI nets, we can infer that a rule handles a specific kind of request that is represented by the constructor operation of the required request token in its left-hand side. For example, if a rule has a token $\mathit{callReq}(\mathit{u1}, \mathit{u2})$ in its left-hand side that does not occur in the right-hand side — i. e., it will be deleted by this rule if applied — and if this rule then is applied in a transformation sequence after the triggering firing step, it is natural to say that this transformation handles the request for calls from one user to another.

We present a simple kind of control structure on rules that depend on specific type of request tokens in Sect. 5.2.3 and use these for defining a higher-order AHLI net (cf. Sect. 4.6) that contains all the handler rules and the AHLI net representing the Communication Platform as tokens. The firing behavior of the higher-order net controls the reconfiguration rules and their applications depending on the firing behavior of the Communication Platform (see Sect. 5.4) and therefore gives a simulation semantics for platform models.

5.1.2.2. Disjunctive firing conditions

Here, we shortly discuss a particular problem of modeling with Petri nets that can be solved by reconfiguration: In a classical Petri net, we can only express conjunctive preconditions for transitions such as that a transition needs a number x_A of resources of place p_A satisfying some predicate ϕ_A *and* a number x_B of resources of place p_B satisfying another predicate ϕ_B. For each part of this conjunction, we can formulate a suitable arc to the corresponding place. But is not possible to express a disjunctive condition of these parts (at least not if the places are different). For example, the condition for sending messages in Skype is that the recipient allows all users to send him messages *or* that he has the sender on his contact list. If we try to combine these conditions into one configuration, we would need in any case that the transition for sending messages demands an identifier token from the recipients contact list for firing. But this condition cannot be *overridden* by some tokens for liberal privacy settings, hence it is impossible to realize this disjunctive condition in an immutable net.

However, if we can reconfigure the net, we can first model the precondition for the particular transition representing the user action of sending a message such that it only depends on the privacy settings of the opposite user. In the moment when this contact is added to the contact list, we reconfigure the sending transition such that afterward it ignores the privacy settings of the added contact because the condition of being present on the contact list is already satisfied. With this approach of reconfiguration between the different conditions of the disjunctive conditions we achieve the desired effect.

5.1.2.3. Conditional but nonlocal dependencies

Another aspect of flexibility for modeling with reconfiguration are nonlocal dependencies. We understand the term “nonlocal” in the sense that Wikipedia describes as

> “a direct influence of one object on another distant object, in violation of the principle of locality”,

which in a Petri net can be interpreted as describing missing structures, i. e., the places with the influencing data elements are not connected to the acting transition. Moreover, an action can also depend on the explicit nonexistence of another structure, which is another level of dependency totally out of the expressive power of Petri nets. For example, for initiating a conference in Skype it is important if some member of the group is blocked by the initiating host, which would prevent that this member participates in the conference. Moreover, a user is considered not to be blocked by another user as long as neither of them explicitly blocked the other. Consider the case that there is a group member who has never had direct contact to the initiating host (although they belong to the same group) so that he has to be regarded as not blocked by the host. This means that there is not any information in the net about the "unblocked by default" setting between the two users. Therefore, the reconfiguration for initiating a conference has to depend on the one hand on explicit data or structure expressing the blocking status and on the other hand on the absence of such a structure. The best possibility to check a condition such as "either there is a structure that decides this condition or otherwise it is true by default" in our formal framework is by employing application conditions for reconfiguring rules possibly handling exceptional cases.[6]

Note that modeling dependencies of actions and their effects by expressing conditions for reconfiguring rules instead of disabling the representing transition is not necessarily different from how this actually works in concrete Communication Platforms: For example, one can always try to call a contact in Skype, even if he seems to be offline. This means, the action itself (and its representing transition) is enabled for the user but the (reconfiguration) effect may just be that nothing happens if the called user really is offline.

5.1.3. Signatures and Algebras for Communication Platform Models

For the general concepts of how to model data and control structures (cf. Sect. 5.1.1.2 and Sect. 5.1.1.3), we provide a basic common signature that should be used for all models of Communication Platforms. The complete signature and an algebra that should be suitable for many modeling cases are given in Table A.1 and Table A.2. Here, we show just the signature without variables in Table 5.1 and explain the algebra informally.

The sort *ID* (and its corresponding carrier set) is meant to represent the identities of the client's users by single tokens on distinguished identity places of the clients in the platform net. The operation *neq* that evaluates to the Boolean value *true* for two different identity strings is essential for many Communication Platforms. Without this operation, we could only formulate firing conditions for transitions that demand the equality of *ID* token values.

The *Data* sort represents the data transmitted among the platform users. In the example algebra, identity and data values are strings of alphanumeric symbols, the latter

[6] Anyway, there is no point in trying to realize the operation for initiating a conference by reconfiguring the transition initiating a call to be connected with all the places containing information about the blocking status of the other members. Such a size-dependent operation is similar to multicasting and demands handling by requests and reconfiguration with amalgamated rules in order to manipulate each member's client structure part in this group.

$\Sigma_{CP} =$	
sorts:	$\mathit{ID}, \mathit{Data}, \mathit{Control}, \mathit{Bool}, \mathit{Int}, \mathit{BoolInt}$
opns:	$T, F \colon \to \mathit{Bool}$
	$\bullet \colon \to \mathit{Control}$
	$0 \colon \to \mathit{Int}$
	$\mathit{neq} \colon \mathit{ID}\ \mathit{ID} \to \mathit{Bool}$
	$\mathit{make} \colon \mathit{Bool}\ \mathit{Int} \to \mathit{BoolInt}$
	$\mathit{inc} \colon \mathit{Int} \to \mathit{Int}$
	$\mathit{dec} \colon \mathit{Int} \to \mathit{Int}$
	$\mathit{and} \colon \mathit{Bool}\ \mathit{Bool} \to \mathit{Bool}$
	$\mathit{or} \colon \mathit{Bool}\ \mathit{Bool} \to \mathit{Bool}$
	$\mathit{not} \colon \mathit{Bool} \to \mathit{Bool}$

Table 5.1.: Abridged Signature Σ_{CP} for Communication Platform Models

additionally of special symbols such as punctuations.

The remaining sorts are used as in Sect. 5.1.1.3 with a constructor *make* for pairs of Boolean and integer values, the operations *inc* and *dec* for increasing and decreasing integer values, and the common Boolean operators.

5.2. Advanced Rule-Based Transformation Concepts

In this section, we discuss two useful extensions of rule-based transformation systems and adapt them to our specific AHLI transformation systems in order to use them for modeling of Communication Platforms.

In order to gain some necessary properties and to simplify the following formal definitions, we consider a fixed $\mathcal{M}$-adhesive category $(\mathbf{AHLINets}(\mathbf{\Sigma}), \mathcal{M})$ for the rest of the thesis. This category consists of the AHLI nets that have an algebra over some fixed signature Σ for the Communication Platform we actually model and a class $\mathcal{M}$ of morphisms as described in Theorem 4.3.4. This means that all the AHLI nets morphisms in this category (as defined in Def. 4.2.3) have id_Σ as their signature component and we may simply omit this component in the following.

5.2.1. Nested Application Conditions

Application conditions have been used for a long time to control the application of transformation rules in transformations of graphs and other structures [EH85, HHT96]. Recently, important analysis results for $\mathcal{M}$-adhesive transformation systems of [EEPT06] have been transfered to transformation systems over rules with nested application conditions [EHL10b, EHL$^+$10a]. We consider the approach of nested application conditions [HP05, HP09], which is one of the most flexible ones for rule-based DPO transformation that we use for reconfigurable AHLI nets. We need nested application conditions for controlling the application of reconfiguring rules in complex cases as we discussed in Sect. 5.1.2 that depend on various structural and data properties.

We first review the definition of rules with nested application conditions and constraints from [HP09].

Definition 5.2.1 (Nested Conditions and Satisfaction)
A *nested condition* (or short: condition) ac over an object P is one of the following (recursive) expressions:

- $ac = true$
- $ac = \exists(a, ac')$, where $a\colon P \to C$ is a morphism and ac' is a condition over C,
- $ac = \neg ac'$, where ac' is a condition over P,
- $ac = \bigvee\limits_{i \in \mathcal{I}} ac'_i$, where (ac'_i) is some $\mathcal{I}$-indexed set of conditions over P

Additionally, there are some shorthand notations based on the previous ones:

$$\mathit{false} := \neg true,\ \exists a := \exists(a, true),\ \forall(a, ac) := \neg\exists(a, \neg ac),\ \bigwedge_{i \in \mathcal{I}} ac_i := \neg \bigvee_{i \in \mathcal{I}} \neg ac_i$$

Given a condition ac over P, a morphism $p\colon P \to N$ *satisfies* ac, written $p \models ac$, if either

- $ac = true$,
- $ac = \exists(a, ac')$ and there exists a morphism $q \in \mathcal{M}$ with $q \circ a = p$ and $q \models ac'$,
- $ac = \neg ac'$ and $p \not\models ac'$,
- $ac = \bigvee\limits_{i \in \mathcal{I}} ac'_i$ and $\exists i \in \mathcal{I}\colon p \models ac_i$ △

$$\begin{tikzcd}
ac \rhd P \arrow[rr, "a"] \arrow[dr, "m"'] & & C \lhd ac' \arrow[dl, "\exists q \in \mathcal{M}"] \\
& N &
\end{tikzcd}$$

Definition 5.2.2 (Rules with Application Conditions)
A rule with application condition is a pair (ϱ, ac) with ac being a condition over the left-hand side L of a span of $\mathcal{M}$-morphisms $\varrho = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$. △

5.2.1.1. Application Conditions and Rules with Variable Algebraic Values

Although the general theory of application conditions for $\mathcal{M}$-adhesive transformation systems is well-elaborated [Gol11], there are some technical problems for special modeling cases with objects having algebraic values, such as attributed graphs and algebraic high-level Petri nets. We shortly point out these problems and discuss possible solutions.

Matching algebraic variables Consider an application condition $\exists a$ for a rule ϱ with $a\colon L_\varrho \to C$ (see Fig. 5.4). This condition is satisfied by a match $o\colon L_\varrho \to ANI$ if there exists an $\mathcal{M}$-morphism $q\colon C \to ANI$ such that $q \circ a = o$. Note that in general a does not have to be an $\mathcal{M}$-morphism!

$$\begin{tikzcd}
C \arrow[dr, "q\in\mathcal{M}"'] & L_\varrho \arrow[l, "a"'] \arrow[d, "o"] \\
 & ANI
\end{tikzcd}$$

Figure 5.4.: Satisfaction of an application condition $\exists a$

Up to now, it is unknown whether the category **Algs** of algebras is $\mathcal{M}$-adhesive for the class of all injective morphisms [Pra08], and the algebra isomorphisms are the only obvious choice for a class $\mathcal{M}$ (cf. Theorem 4.3.4). For AHLI nets (and attributed graphs as well), we have a problem to use application conditions for expressing structural conditions in the case that the algebra of C is not already isomorphic to the one of the target net ANI. Obviously, in this case there cannot exist any $\mathcal{M}$-morphism $q\colon C \to ANI$ and therefore ϱ is not applicable to any net ANI with nonisomorphic algebra. This problem arises in the typical case when we define some term algebra with variables $T_\Sigma(Y)$ for the net in the LHS L_ϱ so that we can use the variables as data elements for tokens in L_ϱ for matching arbitrary token values in some target net ANI (cf. Sect. 5.3). We need to specify an infinite[7] disjunction of application conditions with one condition for each possible evaluation of the variables. For negated conditions, this would have to be an infinite conjunction. Anyway, the requirement that the algebras of C and ANI need to be isomorphic is inconvenient for an intuitive modeling of Communication Platforms with application conditions, especially, if we want to model data transmission of arbitrary data values by rule applications (cf. Sect. 3.2.5.1).

$$\begin{tikzcd}
NAC \arrow[dr, "\nexists q\in\mathcal{M}"', "="] & L_\varrho \arrow[l, "a"'] \arrow[d, "o"] \\
 & ANI
\end{tikzcd}$$

(a) Satisfaction of $\neg\exists a$

$$\begin{tikzcd}
NAC \arrow[d, "n'"] \arrow[dr, phantom, "(\mathbf{PO})"] & L_\varrho \arrow[l, "a"'] \arrow[d, "e"] \arrow[dd, bend left=60, "o"] \\
NAC' \arrow[dr, "\nexists q\in\mathcal{M}"', "="] & L'_\varrho \arrow[l, "a'"'] \arrow[d, "m"] \arrow[r, phantom, "="] & {} \\
 & ANI
\end{tikzcd}$$

(b) Instantiation of $\neg\exists a$ along match o

Figure 5.5.: Satisfaction and instantiation of a negative application condition $\neg\exists a$

In [Her11], the same problem with negative application conditions (NACs)[8] for at-

[7] This depends on the cardinality of the variable's data type in the target net.

[8] NACs are application conditions of the fixed form $\neg\exists a$.

tributed graphs is approached by instantiating the NACs for a given match o (see Fig. 5.5a). Consider an $\mathcal{E}$-$\mathcal{M}$-factorization for the matches, i. e., $o = m \circ e$, $o \in \mathcal{E}$, $m \in \mathcal{M}$. To instantiate a NAC, we construct the pushout over e and a and check whether the instantiated NAC $\neg\exists a'$ is satisfied by the match remainder m (see Fig. 5.5b). This generalizes the method of how the AGG tool (see [Tae04] and Sect. 8.2) instantiates application conditions with variables as attribute values, where an epi-mono-factorization instead of an $\mathcal{E}$-$\mathcal{M}$-factorization is used so that $L'_\varrho = o(L'_\varrho)$ [KHM06]. Both approaches need to construct intermediate application conditions to be checked, and they are limited to NACs.

To solve this problem for AHLI nets and nested application conditions, we propose a practical approach in the next section that adapts the nested application conditions such that there is no need for instantiations. Instead, we define a weak variant of "structural" satisfaction for application conditions of rules with algebraic variables, which we can check directly for matches.

Noninjective matches Unfortunately, the satisfaction of application conditions can be unintuitive for noninjective matches. Consider the example of a NAC like in Fig. 5.6, where L contains two places p_1, p_2 of the same type and NAC contains three places of the same type, i. e., the NAC forbids that there exist a third unmatched place p_3 in a target net ANI. But, if the match o is noninjective and maps the two places p_1, p_2 to the same place p_{12} in ANI, there cannot exist a proving morphism $q \in \mathcal{M}$ (which necessarily would have to be injective on places), regardless how many additional (forbidden) places of this type like p_3 ANI contains. In short, this NAC loses its effect on noninjective matches. On the other hand, we do not want to restrict transformations to injective matches, because different variables could not be matched to the same value any more. As a solution for graphs, [HEOG10] proposes "almost injective" matches, which are injective only on the graph parts and may be noninjective on the data part.

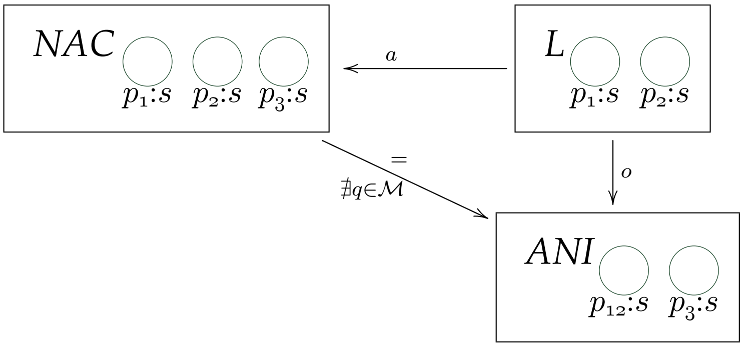


Figure 5.6.: Negative application condition and noninjective match

Inspired by the notion of almost injective matches for graph transformation systems, we consider in the rest of this thesis only AHLI net transformations at matches that are injective in their structural parts, i. e., the components for places, transitions, and tokens. With this restriction, we avoid the described unintuitive satisfaction of application conditions for AHLI net transformations. Our adapted approach of application conditions that we present in the following features a data–structure factorization of AHLI morphisms, which we use to describe this property.

5.2.1.2. Application Conditions for AHLI Nets with Variables

To be able to express conditions for AHLI net transformation rules over some term algebra with variables in a compact way, we consider a special class of application conditions for rules with variables as tokens in the following and define a modified satisfiability relation for this kind of conditions. In order to avoid the previously described problem of morphisms that need to have an isomorphic algebra component for proving the satisfaction of a constraint, the idea of the new "structural" satisfiability relation $\models_S$ is to ignore the data part of a match and to take into account only structural features. We show an equivalence between the satisfaction of application conditions with variables under $\models_S$ and under $\models$, which proves that $\models_S$ follows our intentions.

First, we need to distinguish and separate the structural and the data "part" of an AHLI net morphism before we can formulate how a condition is satisfied structurally. For this, we define a unique decomposition of AHLI net morphisms in a data component and a structural component.

Definition 5.2.3 (AHLI Morphism Factorization)
Given an AHLI net morphism $f = (f_P, f_T, f_A, f_I)\colon ANI_1 \to ANI_2$ with $ANI_i = (P_i, T_i, pre_i, post_i, type_i, A_i, I_i, m_i)$, we denote its *data–structure factorization* by

- $f^D := (id_{P_1}, id_{T_1}, f_A, id_{I_1})\colon ANI_1 \to ANI_{12}$ with $ANI_{12} = (P_1, T_1, pre_1, post_1, type_1, A_2, I_1, m_{12})$ and $m_{12} = (f_A \otimes id_{P_1}) \circ m_1\colon I_1 \to A_2 \otimes P_1$,
- $f^S := (f_P, f_T, id_{A_2}, f_I)\colon ANI_{12} \to ANI_2$

f^D and f^S are well-defined AHLI net morphisms because the following diagram in the right commutes and obviously $f^S \circ f^D = f$.

$$\begin{tikzcd}
ANI_1 \arrow[d, "f^D"] \arrow[dd, bend right=60, "f"'] \\
ANI_{12} \arrow[d, "f^S"] \\
ANI_2
\end{tikzcd}
\qquad
\begin{tikzcd}
I_1 \arrow[r, "m_1"] \arrow[d, "id_{I_1}"] \arrow[dd, bend right=60, "f_I"'] & A_1 \otimes P_1 \arrow[d, "f_A \otimes id_{P_1}"'] \arrow[dd, bend left=60, "f_A \otimes f_P"] \\
I_1 \arrow[r, "m_{12}"'] \arrow[d, "f_I"] & A_2 \otimes P_1 \arrow[d, "id_{A_2} \otimes f_P"'] \\
I_2 \arrow[r, "m_2"'] & A_2 \otimes P_2
\end{tikzcd}$$

The following properties follow from this definition:

1. $f = g \Rightarrow f^D = g^D \wedge f^S = g^S$
2. $(f^S)^S = f^S \wedge (f^D)^D = f^D$

3. $(f^D)^S = id_{ANI_{12}} = (f^S)^D$
4. $f \in \mathcal{M} \Rightarrow f^S \in \mathcal{M} \wedge f^D \in \mathcal{M}$ △

Definition 5.2.4 (Almost Injective AHLI morphisms)
We call an AHLI morphism f *almost injective* iff $f^S \in \mathcal{M}$. △

With this factorization, we can define application conditions for rules over a term algebra with variables. For the result that we show later, it is important to restrict this conditions such that they only express structural conditions. This means that we demand a morphism in an application condition to be the same as its structural component, i. e., its algebra component is the identity algebra morphism.

Definition 5.2.5 (Variable Application Conditions)
An application condition ac over P (as in Def. 5.2.1) is a *variable application condition* (VAC) if the AHLI net P has the term algebra with variables $T_\Sigma(Y)$ (for some set Y of variables) and if for every morphism a in ac holds that $a^S = a$, i. e., its algebra component is $a_A = id_{T_\Sigma(Y)}$ and $a^D = id_P$. △

For variable application conditions, we weaken the notion of satisfiability to structural satisfiability. The main difference is that the morphism proving the satisfaction of a condition does not have to be an $\mathcal{M}$-morphism, but only its structural component has to. With this, we allow a proving morphism to have an algebra component that is not isomorphic and hence can evaluate tokens with variables as values to concrete values of the target system algebra.

Definition 5.2.6 (Structural Satisfaction of Application Conditions)
Given a condition ac over P, a morphism $p\colon P \to ANI$ *structurally satisfies* ac, written $p \models_S ac$, if either

- $ac = true$,
- $ac = \exists(a, ac')$ with $a\colon P \to C$ and there exists an almost injective morphism $q\colon C \to ANI$ with $q \circ a = p$ and $q \models_S ac'$,
- $ac = \neg ac'$ and $p \not\models_S ac'$,
- $ac = \bigvee\limits_{i \in \mathcal{I}} ac_i'$ and $\exists i \in \mathcal{I}\colon p \models_S ac_i$ △

Before we can show how the satisfiability relation $\models_S$ is related to the classic $\models$, we need to introduce some additional notations for the factorizations of AHLI net morphisms and the factorization of composed AHLI net morphisms.

Definition 5.2.7 (Data Extension and Shifting of AHLI Net Morphisms)
Given an AHLI net morphism $f\colon ANI_0 \to ANI_1$ and an algebra Σ-homomorphism $h\colon A_{ANI_1} \to A_2$, we denote the *data extension* of f by h as

$$f^{+h} := (f_P, f_T, h \circ f_A, f_I)\colon ANI_0 \to ANI_1^{\uparrow h}$$

where $ANI_1^{\uparrow h} = (P_1, T_1, pre_1, post_1, type_1, A_2, I_1, (h \otimes id_{P_1}) \circ m_1)$. The argument for the well-definedness of f^{+h} is the same as for f^D above.

Further, given a morphism $a\colon ANI_1 \to ANI_1'$ with $a^S = a$, we define the *data shifting* of a along h as $a^{\uparrow h} = (a_P, a_T, a_I, id_{A_2})\colon P^{\uparrow h} \to C^{\uparrow h}$ in the following commuting diagram.

$$\begin{tikzcd}
ANI_1 \arrow[r, "a=a^S"] \arrow[d, "id^{+h}_{ANI_1}"'] \arrow[dr, phantom, "="] & ANI_1' \arrow[d, "id^{+h}_{ANI_1'}"] \\
ANI_1^{\uparrow h} \arrow[r, "a^{\uparrow h}=(a^{\uparrow h})^S"] & ANI_1'^{\uparrow h}
\end{tikzcd}$$

Note that $(a^{\uparrow h})^S = a^{\uparrow h}$ and that the diagram is a pushout because it has componentwise pushouts as can be easily seen by the opposing identity components.

With this, we get

1. $f^D = id^{+f_A}_{ANI_0}$, which we may use as an alternative notation for the data part of f together with $f^S\colon ANI_0^{\uparrow f_A} \to ANI_1$
2. so that $(id^{+f_A}_{ANI_0})^S = (f^D)^S = id_{ANI_1^{\uparrow h}}$,
3. and also that $f^{+h} = id^{+h}_{ANI_1^{\uparrow h}} \circ f$
4. and for $h = id_{A_{ANI_1}}$ the identity reductions $f^{+h} = f$ and $a^{\uparrow h} = a$. △

Fact 5.2.8 (Decomposition of Data–Structure Factorizations)
For all morphisms $f\colon ANI_1 \to ANI_2$ and $g\colon ANI_2 \to ANI_3$ holds that

$$(g \circ f)^D = id^{+g_A}_{ANI_1^{\uparrow f_A}} \circ f^D \quad \text{and} \quad (g \circ f)^S = g^S \circ (f^S)^{\uparrow g_A}$$

Proof. In the following commuting diagram, it can be shown that $(g^D \circ f^S)^D = id^{+g_A}_{ANI_1}$ and because of the commuting square in the middle, $(g^D \circ f^S)^S$ is the data shifting of f^S along g_A.

$$\begin{tikzcd}
ANI_1 \arrow[rrr, "f"] \arrow[dr, "f^D=id^{+f_A}_{ANI_1}"] \arrow[ddrrr, bend right, "(g\circ f)^D"'] & & & ANI_2 \arrow[rrr, "g"] \arrow[drr, "g^D=id^{+g_A}_{ANI_2}"] & & & ANI_3 \\
& ANI_1^{\uparrow f_A} \arrow[urr, "f^S"'] \arrow[rrrr, "g^D\circ f^S"'] \arrow[drr, "(g^D\circ f^S)^D"'] & & & & ANI_2^{\uparrow g_A} \arrow[ur, "g^S"'] & \\
& & & ANI_1^{\uparrow g_A\circ f_A} \arrow[urr, "(g^D\circ f^S)^S"'] \arrow[uurrr, bend right, "(g\circ f)^S"'] & & &
\end{tikzcd}$$

Definition 5.2.9 (Data Shifting of Application Conditions)
Given a condition ac over an AHLI net P with algebra A_1 such that for every morphism a in ac holds that $a^S = a$ (i. e., its algebra component is id_{A_1}) and an algebra homomorphism $h\colon A_1 \to A_2$. The data shifting of $ac = \exists(a, ac')$ with $a\colon P \to C$ along h is defined recursively as

$$DShift(h, ac) := \exists(a^{\uparrow h}, DShift(h, ac'))$$

as in the following commuting diagram:

$$\begin{tikzcd}
ac \triangleright P \arrow[r, "a=a^S"] \arrow[d, "id_P^{\uparrow h}"'] & C \triangleleft ac' \arrow[d, "id_C^{\uparrow h}"] \\
DShift(h, ac) \triangleright P^{\uparrow h} \arrow[r, "a^{\uparrow h}"] & C^{\uparrow h} \triangleleft DShift(h, ac')
\end{tikzcd}$$

Moreover, $DShift(h, true) = true$ and the data shifting construction is extended to Boolean operators in the usual way. It is noteworthy that the commuting diagram is a pushout in our category of AHLI nets because it consists of componentwise pushouts due to the opposing identities (cf. Fact 4.2.5). △

The following main theorem states that a morphism structurally satisfies a variable application condition if the "evaluation" of this condition along the data part of the considered morphism is satisfied by its structural component. This result relates $\models_{\mathrm{S}}$ to $\models$ and shows that it follows our intentions of disregarding the concrete variable evaluation of a morphisms.

Theorem 5.2.10 (Structural Satisfaction of Variable Appl. Conditions)
Given a variable application condition ac over P. For all morphisms $p\colon P \to ANI$ holds the equivalence

$$p \models_{\mathrm{S}} ac \Leftrightarrow p^S \models DShift(p_A, ac)$$

Proof. See appendix A.3.5. □

The following corollary states that for morphisms with isomorphic algebra component the satisfiability of a variable application condition under $\models$ and $\models_{\mathrm{S}}$ coincides.

Corollary 5.2.11 (Structural Satisfaction of Variable Appl. Conditions)
For all variable application conditions ac and all morphisms p having an isomorphic algebra component $p_A\colon A_1 \to A_2$ holds the equivalence

$$p \models ac \Leftrightarrow p \models_{\mathrm{S}} ac$$

Proof. This follows directly from the previous Theorem 5.2.10 and Lemma A.3.3.
$p \models ac \Leftrightarrow p \models DShift(id_{A_1}, ac) \overset{Lemma}{\Leftrightarrow} p^S \models DShift(p_A \circ id_{A_1}, ac) \overset{Thm.}{\Leftrightarrow} p \models_{\mathrm{S}} ac$ □

5.2.2. Interaction Schemes and Rule Amalgamation for Quantity-dependent Actions in Communication Platforms

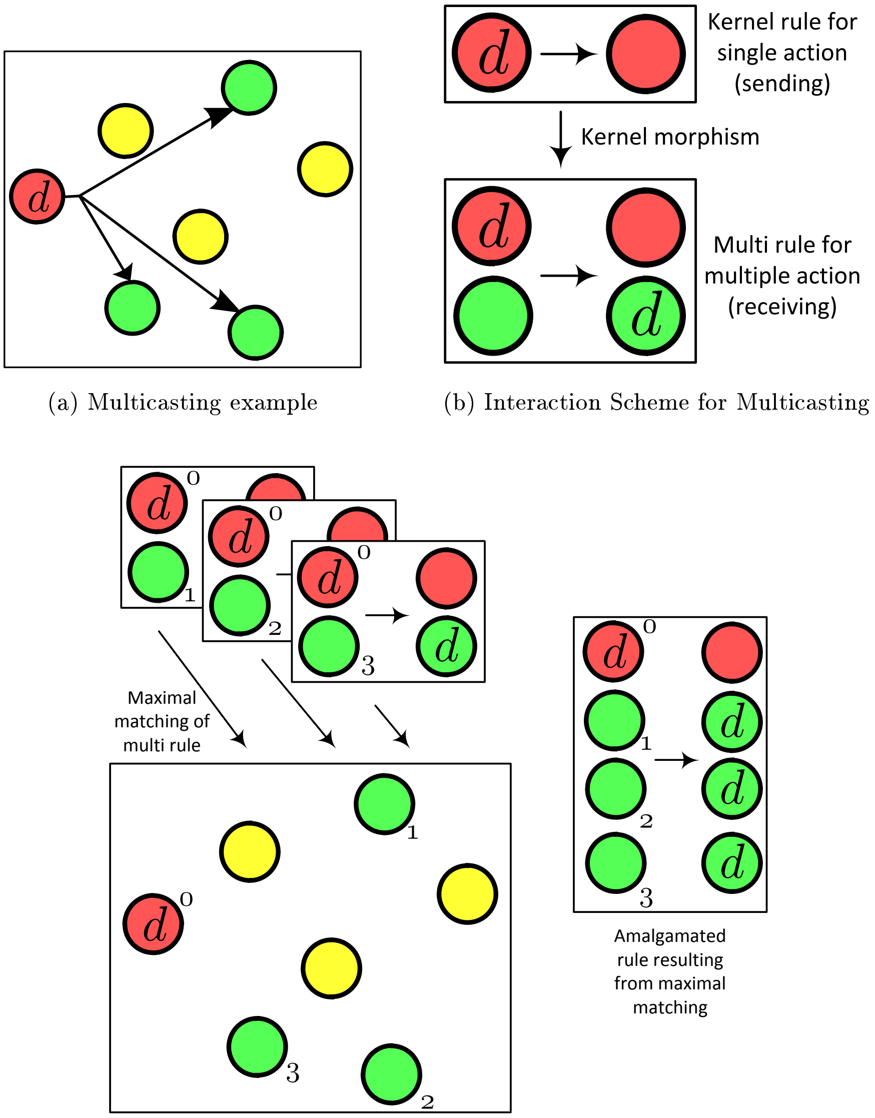
In Sect. 3.2.5.1, we discuss in detail the problems of realizing the operation of multicasting data to groups of arbitrary size in Petri nets, even if the nets are reconfigurable by application of rules as AHLI nets are. Such actions cannot be realized by a reconfigurations with a simple transformation rule as in Def. 4.2.7 because they depend on the exact number of group members.

As a solution to this problem, we use the concepts of interaction schemes and rule amalgamation for $\mathcal{M}$-adhesive systems from [GEH10, Gol11] and define interaction schemes with variable conditions (cf. Def. 5.2.5 and [BEE$^+$10]). The concepts of interaction schemes and the amalgamated rules resulting from a maximal matching of these schemes are useful for defining transformations that depend on a priori unknown amounts of parts to be manipulated. Note that we only consider almost injective matches (cf. Sect. 5.2.1.1, Def. 5.2.4).

The action pattern we find in the problem of multicasting data is a typical use case for employing amalgamated rules resulting from maximal matching of a scheme, which we illustrate on a very abstract level in Fig. 5.7: There is a type of action occurring once — the sending (and removal) of a data token from a certain place, another type of action whose number of occurrences depends on the actual structure — the receiving (on designated input places) of the data token sent by the single action, and the multiple action is executed on all parts in the net that satisfy some condition — the clients being a legitimate receiver of the multicasted data (see Fig. 5.7a, where the red node wants to send some data d). The single and multiple actions can be separately modeled by rules that are then related by kernel morphisms into an interaction scheme, where the multiple rules should be applicable only on clients that may receive the sent data, which is why the multi rules must contain the single rule (that we call *kernel rule*) as well (see Fig. 5.7b). These kernel morphisms represent the synchronization of the multiple actions at a fixed occurrence of the single action. Given a match for the single action rule, the notion of maximal matching for such an interaction scheme ensures that the multi rule is instantiated and matched at all possible matches that are compatible with the single action match. A suitable matching results in the construction of an amalgamated rule that performs all manipulations of the single and the multi action rules in a single transformation step (see Fig. 5.7c with the numbers indicating the matches). This is guaranteed by the Multi-Amalgamation Theorem in [GEH10].

Remark (Amalgamation of Variable Application Conditions). Because of the variable application conditions (VAC) and structural satisfaction (cf. Def. 5.2.6), we need to weaken the definitions from [GEH10] to be able to formulate VACs for the kernel and multi rules in interaction schemes and to get amalgamated rules for such matched interaction schemes at all.

The main reason for these modifications is that we do not have operations that shift VACs over arbitrary morphisms and rules yielding equivalent VACs as the shift operations on nested application conditions from [GEH10] do. The Multi-Amalgamation Theorem is still valid for the effect of the amalgamated rule w. r. t. the matched multi


d
Kernel rule for single action (sending)
Kernel morphism
Multi rule for multiple action (receiving)
Maximal matching of multi rule
0
1
2
3
Amalgamated rule resulting from maximal matching

(a) Multicasting example

(b) Interaction Scheme for Multicasting

(c) Maximal matching and resulting amalgamated rule

Figure 5.7.: Illustration of interaction schemes, maximal matching, and amalgamated rules

rules, but we cannot construct application conditions for the amalgamated rule such that it can only be applied at matches that can be decomposed to matches for multi rules satisfying the multi rule's application conditions. In short, we cannot simply amalgamate a maximally matched interaction scheme with VACs to an equivalent single rule with VACs. However, from a pragmatic point of view this is not critical for our modeling endeavor because we always apply amalgamated rule resulting from maximally matched interaction schemes and may decide to apply only those amalgamated rules that are constructed from matches satisfying the interaction scheme's VACs and still get the desired transformation result. Therefore, in the following we weaken some restrictions in the definitions of [GEH10] and define the applicability of amalgamated rules over the maximal matches of the interaction scheme. In [BEE$^+$10], the main definitions and theorems of [GEH10] are given for interaction schemes and rules without application conditions but we explicitly want the applicability of the amalgamated rule to depend on a maximal matching of the interaction scheme, which is not treated in this article.

We begin with the definition of kernel morphisms and bundles that do not impose restrictions on the application conditions as in [GEH10].

Definition 5.2.12 (Loose Kernel Morphisms, Bundles, Interaction Schemes) Given rules with variable application conditions $\varrho_i = (L_i \overset{l_i}{\leftarrow} K_i \overset{r_i}{\rightarrow} R_i, ac_i)$ for $i \in \{0, 1\}$, a *loose kernel morphism* is a triple of $\mathcal{M}$-morphisms $s = (s_L\colon L_0 \to L_1, s_K\colon K_0 \to K_1, s_R\colon R_0 \to R_1)\colon \varrho_0 \to \varrho_1$ such that the squares in the following diagram are pullbacks.

$$\begin{tikzcd}
L_0 \arrow[d, "s_L"'] & K_0 \arrow[l, "l_0"'] \arrow[r, "r_0"] \arrow[d, "s_K"] \arrow[dl, phantom, "(PB_1)"] \arrow[dr, phantom, "(PB_2)"] & R_0 \arrow[d, "s_R"] \\
L_1 & K_1 \arrow[l, "l_1"] \arrow[r, "r_1"'] & R_1
\end{tikzcd}$$

A *bundle* of loose kernel morphisms is a set of kernel morphisms $s = (s_i\colon \varrho_0 \to \varrho_i)_{1 \leq i \leq n}$ that have the same domain rule ϱ_0. The common domain rule ϱ_0 of a bundle is called *kernel rule*, and its other rules are called *multi rules*.

A *loose interaction scheme* is a bundle of loose kernel morphisms. △

Remark (Interaction schemes and resulting bundles). Though technically an interaction scheme is just a bundle, we do not regard every bundle as an interaction scheme. A scheme's abstract multi rules can be matched at several matches, which results in a bundle of *instantiated* multi rules, which in turn are used to build the amalgamated rules as in the next definition. The bundle of instantiated multi rules is just a temporary construction on the way to the amalgamated rule depending on a specific match, whereas the interaction scheme is a part of the transformation system like normal transformation rules.

Definition and Fact 5.2.13 (Loosely Amalgamated Rules)
Given rules $\varrho_i = (L_i \overset{l_i}{\leftarrow} K_i \overset{r_i}{\rightarrow} R_i, ac_i)$ for $1 \leq i \leq n$ and a bundle of loose kernel morphisms $s = (s_i\colon \varrho_0 \to \varrho_i)_{1 \leq i \leq n}$, then the loosely amalgamated rule $\tilde{\varrho}_s = (\tilde{L}_s \overset{\tilde{l}_s}{\leftarrow} \tilde{K}_s \overset{\tilde{r}_s}{\rightarrow} \tilde{R}_s, \tilde{ac}_s)$ is constructed as the componentwise colimit of the kernel morphisms:

- $\tilde{L}_s := Col((s_{i,L})_{1 \leq i \leq n})$, $\tilde{K}_s := Col((s_{i,K})_{1 \leq i \leq n})$, $\tilde{R}_s := Col((s_{i,R})_{1 \leq i \leq n})$,
- $\tilde{l}_s$ and $\tilde{r}_s$ are induced by $(t_{i,L} \circ l_i)_{1 \leq i \leq n}$ and $(t_{i,R} \circ r_i)_{1 \leq i \leq n}$, respectively,
- $\tilde{ac}_s = true$

$$\begin{tikzcd}
\varrho_0\colon \arrow[d, "s_i"'] & L_0 \arrow[d, "s_{i,L}"'] & K_0 \arrow[l, "l_0"'] \arrow[r, "r_0"] \arrow[d, "s_{i,K}"] & R_0 \arrow[d, "s_{i,R}"] \\
\varrho_i\colon \arrow[d, "t_i"'] & L_i \arrow[d, "t_{i,L}"'] & K_i \arrow[l, "l_i"'] \arrow[r, "r_i"] \arrow[d, "t_{i,K}"] & R_i \arrow[d, "t_{i,R}"] \\
\tilde{\varrho}_s\colon & \tilde{L}_s & \tilde{K}_s \arrow[l, "\tilde{l}_s"'] \arrow[r, "\tilde{r}_s"] & \tilde{R}_s
\end{tikzcd}$$

The amalgamated rule is well-defined and leads to kernel morphisms

$$(t_i = (t_{i,L}, t_{i,K}, t_{i,R})\colon \varrho_i \to \tilde{\varrho}_s)_{1 \leq i \leq n}$$

. △

Remark (Loose kernel morphisms and amalgamated rules). The condition from [GEH10] that we omit in the definition of kernel morphisms describes that the satisfaction of the application condition ac_1 of rule ϱ_1 implies the satisfaction of the shifted condition $\mathrm{Shift}(ac_0, s_L)$.

Consequently, the loosely amalgamated rule does not have the combined shifted application conditions $\tilde{ac}_s = \bigwedge_{1 \leq i \leq n} \mathrm{Shift}(t_{i,L}, ac_i)$ like the amalgamated rule in [GEH10] has.

For maximal matching of interaction schemes, we use the maximal disjoint matching from [Gol11], which means that the matches for the multi rules may overlap only in the elements matched by the kernel rule. We restrict the possible matches to almost injective ones (cf. Sect. 5.2.1.1, Def. 5.2.4).

Definition 5.2.14 (Maximal Disjoint Matching of Interaction Schemes) Given an interaction scheme $is = (s_i\colon \varrho_0 \to \varrho_i)_{1 \leq i \leq n}$ and a net N, a maximal disjoint matching $o = (o_k\colon \hat{L}_k \to N)_{0 \leq k \leq j}$ is defined and constructed as follows:

1. Set $j = 0$. Choose an almost injective match $o_0\colon L_0 \to N$ for the kernel rule $\hat{\varrho}_j = \varrho_0$ that satisfies the gluing condition and structurally satisfies its variable application condition, i. e., $N \xRightarrow{\varrho_0, o_0} N_0$ is a valid transformation.

2. As long as possible: Increase j, choose a multi rule $\hat{\varrho}_j = \varrho_i$ for $1 \leq i \leq n$ and find an almost injective match $o_j\colon L_i \to N$ for the multi rule ϱ_i that satisfies the gluing condition and structurally satisfies its variable application condition, i. e., $N \xRightarrow{\varrho_i, o_j} N_j$ is a valid transformation, and that is compatible to the kernel match, i. e., $o_j \circ s_{i,L} = o_0$. The matches have to be mutually different and disjoint up to the kernel match, i. e., for all $1 \leq k \leq j-1$, we have $o_j \neq o_k$ and the square $(\mathrm{P_{jk}})$ has to be a pullback.

$$\begin{tikzcd}
L_0 \arrow[r, "s_{i,L}"] \arrow[d, "\hat{s}_{k,L}"'] \arrow[dr, phantom, "(\mathbf{P_{jk}})"] & L_i \arrow[d, "o_j"] \\
\hat{L}_k \arrow[r, "o_k"'] & N
\end{tikzcd}$$

3. If no more almost injective match for any multi rule in the interaction scheme can be found, $o = (o_k\colon \hat{L}_k \to N)_{1 \leq k \leq j}$ is a maximal disjoint match for is. △

Remark. Fact 4.31 in [Gol11] remarks that for graphs and graph-based structures, the square (P_{ik}) is a pullback if and only if $o_i(L_i) \cap o_k(\hat{L}_K) = o_0(L_0)$.

Definition and Fact 5.2.15 (Application of Interaction Schemes)
Given an interaction scheme $is = (s_i\colon \varrho_0 \to \varrho_i)_{1 \leq i \leq n}$ and a maximal disjoint matching $o = (o_j\colon \hat{L}_j \to N)_{0 \leq j \leq k}$ into a net N, the result of applying is at o to N is defined by the following construction:

1. We construct the loosely amalgamated rule $\tilde{\varrho}_{\widehat{is}}$ over the bundle of instantiated multi rules $\widehat{is} = (\hat{s}_j\colon \varrho_0 \to \hat{\varrho}_j)_{1 \leq j \leq k}$ that results from the maximal matching o of is (see Def. 5.2.13).

2. Since we have consistent matches, i. e., $\forall 1 \leq j \leq k\colon o_j \circ \hat{s}_{j,L} = o_0$, the colimit $\tilde{L}_{\widehat{is}}$ as the left-hand side of $\tilde{\varrho}_{\widehat{is}}$ implies that there is a unique induced morphism $\tilde{o}\colon \tilde{L}_{\widehat{is}} \to N$ with $\tilde{o} \circ \hat{t}_{j,L} = o_j$.

3. We define the result of the transformation $N \xRightarrow{is,o} N'$ as the result of $N \xRightarrow{\tilde{\varrho}_{\widehat{is}},\tilde{o}} N'$.

For any maximal disjoint matching o of is, the transformation $N \xRightarrow{\tilde{\varrho}_{\widehat{is}},\tilde{o}} N'$ is well-defined and consists of the combined effects of the single multi rule transformations $N \xRightarrow{\varrho_j,o_j} N_j$ for $1 \leq j \leq k$.

Proof (idea). From Thm. 4.30 in [Gol11], we get that a maximal weakly disjoint matching leads to an s-amalgamable bundle of direct transformations without application conditions. The Multi-Amalgamation Theorem states that there exists a transformation for the amalgamated rule over this bundle, whose match is constructed as in item 2 (see the proof of the Multi-Amalgamation Theorem for the details). From the construction of the loosely amalgamated rule over the multi rules with application conditions, we get the same amalgamated rule as for the multi rules without application conditions. Hence, the loosely amalgamated rule can be applied at the constructed match.

Furthermore, the Multi-Amalgamation Theorem states that the amalgamated transformation can be decomposed into the direct transformations of the bundle and some corresponding complement rule. □

Example. For an example of a maximal matching of an interaction scheme, the construction of the corresponding amalgamated rule, and its application at the induced match, we refer to page 243 in the case study on modeling Skype in Fig. 7.56.

Other Modeling Use Cases for Interaction Schemes

Dispatching communication data to an a priori unknown number of recipients to realize multicasting that depends on communication rights and status of each recipient w. r. t. the sender is not the only modeling case we need interaction schemes and amalgamated rules for. Another one representing a class of similar actions is the starting of a conference for an arbitrary number of users. Typically, in conferences one user has a special role such as being its host so that his client structure has to be different from the (arbitrarily many) other participants' clients. The host's client structure is the only one of which we are sure that it has to be reconfigured by applying exactly one rule that equips his client with the appropriate conference actions. Then we need another rule for the "regular" participants that has to be applied according to the number of invited clients. We clearly recognize the interaction scheme pattern, where the kernel rule modifies the host's client and the multi rule is instantiated and matched for each invited client which leads to a suitable amalgamated rule. The multi rule may have application conditions that decide whether an invited client really is added to the conference or not.

5.2.3. Handling of Token Requests

In Sect. 5.1.2.1, we propose to let reconfiguring rule applications depend on the presence of special request tokens that are produced by transitions, which we therefore consider to "trigger" the reconfigurations by producing the request tokens that may encapsulate additional data determining the reconfiguration's effect, e. g., the identity of the target user to be connected with. Following this modeling principle, a reconfigurable AHLI net — i. e., an initial AHLI platform net together with a set of transformation rules, both modeled accordingly to the said principle — would allow us to simulate all possible actions and following reconfigurations. Unfortunately, for this simple approach there arise two problems we have to solve. Both stem from the fact that the semantics of this reconfigurable AHLI net would overapproximate the behavior of the system we want to model, i. e., its firing steps and reconfigurations allow more configurations of the AHLI platform net to occur than we (want to) have.

One problem is that the reconfigurable AHLI net does not specify whether or when a request produced by a transition firing has to be handled by a rule application at all. We would expect a concrete system like Skype to handle such requests instantly or at least as soon as possible. From the high-level perspective of modeling Communication Platforms we can even expect the posing of a request and its handling to be performed as an atomic action without the possibility that another user can request another reconfiguration before the first has been handled. This is a problem of coordinating and controlling firing steps and rule applications. We discuss a solution for this problem in Sect. 5.4 but before, we treat in this section the following second problem.

This other problem concerns the possibility of actions that may fail in a certain sense or that may have different outcomes depending on some environmental conditions. For example, in Skype an action for requesting a special channel for contact exchange between two clients (cf. Fig. 3.2) may be performed more than once[9], which would result

[9] For a concrete model, see *requestContact* from the Skype client core of the case study in Fig. 7.1

in multiple concurring request between the same two users. This situation is not allowed in Skype and therefore we also want to avoid it in our platform model. But if we require that contact exchange channels be unique, we need a possibility to discard such invalid requests. Without inhibitor arcs, it is impossible to model the client by a high-level net such that it prevents the request-creating transition for this action to fire twice for the same target user. Instead of extending the Petri net model by inhibitor arcs, we make use of the reconfiguring transformation system: It is clear that the request has to be discarded if there already exists a channel structure between the users. This can be simply modeled by a reconfiguring rule for this case that matches a significant part of a channel structure between the requesting and the target client and that just deletes the request token. More important, the discarding rule needs to match an optional structure so that it should be applied with a higher priority than the regular one, which is also applicable in the exceptional case. Another problem is that there may be reconfigurations that cannot be modeled by a single rule, especially when interaction schemes and amalgamated rules are involved and clients have to be treated differently. Concluding this discussion, we can say that we need a way to express the relation of a request token (or its type) to several rules that describe the possible outcomes of handling it.[10] To have the means to express this relation of rules to a type of request (represented by tokens), we introduce a control structure for rules and interaction schemes that covers this issue and assign such a rule structure to each operation in the signature that creates a request token.

Definition 5.2.16 (Request Sorts and Constructors)
Given a signature $\Sigma = (S, (OP_{dom,codom})_{(dom,codom) \in S^* \times S})$ and a distinguished subset of its sorts $S_{Req} \subseteq S$, called the *request sorts*, we define the set of *request constructors* as $Constr(S_{Req}) := \bigcup_{dom \in S^*} \bigcup_{rs \in S_{Req}} OP_{dom,rs}$, i. e., all Σ-operations $op\colon dom \to rs$ having a request sort rs as codomain. △

Definition 5.2.17 (Token Request Handlers and Priority List Sequences)
For a given set of request constructors $Constr(S_{Req})$ and sets of reconfiguration rules R and interaction schemes IS, we define a *token request handler* as a function $handler\colon Constr(S_{Req}) \to PSEQ(R \cup IS)$, where $PSEQ(X)$ is the set of *priority list sequences* over a set X defined as the least set satisfying the following conditions:

1. $PSEQ(X)$ contains all (nonempty) *priority lists* over X. Such a priority list of length n is an expression $x_1 \parallel \ldots \parallel x_n$ with $\forall 1 \le i \le n\colon x_i \in X$.

2. For each two elements $e_1, e_2 \in PSEQ(X)$ also the sequence $e_1; e_2$ is in $PSEQ(X)$.△

Remark (Simple request handlers and shared handler rules). An expression $handler(req)$ handles a single given request of the request constructor req (cf. Sect. 5.4). Therefore, the

[10] For an example, see the reconfiguration to a conference triggered by the action *call* in the case study in Fig. 7.2. In this reconfiguration, we need to reconfigure the actions of the group members that are participating in the conference differently than the actions of the members that do not participate, but this is not possible with a single rule, as we need one rule for each case to match the preconditions for participating in the conference or not.

marking of the LHS of each rule (or kernel rule for interaction schemes) in $\mathit{handler}(\mathit{req})$ typically contains exactly one token of request constructor req. Moreover, for complex handlings that encompass the effects of simpler handlings, the platform designer has the option to reuse rules from other handling expressions (for some different request constructor req'). The request token for req' that is probably needed to be matched by such a shared rule then has to be created by a preceding rule in $\mathit{handler}(\mathit{req})$. But usually, for simple handlings it is sufficient that no rule (plain, kernel, or multi rules) in $\mathit{handler}(\mathit{req})$ creates new tokens of any other request constructor in $\mathit{Constr}(S_{\mathit{Req}})$. An example for reusing rules is given in the Skype modeling case study on page 223 with the rule *ParticipateConference*.

The notation of priority lists with the delimiter $\|$ is inspired by the usage of this Boolean operator in JavaScript, which is the "short-circuit or" that evaluates (boolean) expressions up to the first one that succeeds or evaluates to true and then stops evaluating.

Definition 5.2.18 (Matching of Priority List Sequences)
Given an AHLI net ANI and a priority list sequence $\mathit{seq} = e_1; e_2; \dots e_n$ of length n, a valid match sequence of seq on ANI is defined by the following recursive construction. Start with the empty sequence of (multi) matches o and with $i = 1$ in the first step:

1. Consider the priority list $e_i = \varrho_{i,1} \parallel \dots \parallel \varrho_{i,m}$ of length m. Set $j_i = 1$.

2. If there are valid (maximal disjoint) almost injective matches o_{i,j_i} from ϱ_{i,j_i} to ANI_i satisfying the application conditions then choose any, add o_{i,j_i} to the list o, and continue in the next step with ANI_{i+1} resulting from the direct transformation $\mathit{ANI}_i \xRightarrow{\varrho_{i,j_i}, o_{i,j_i}} \mathit{ANI}_{i+1}$. Also, if $j = m$ then go on in the next step with $\mathit{ANI}_{i+1} = \mathit{ANI}_i$. Otherwise, increase j_i and go through this step again.

3. If $i = n$ then o is a valid match sequence for seq on ANI_0. Otherwise, increase i and go back to the first step. △

Definition 5.2.19 (Application of Request Handlers)
Given an AHLI net ANI_0 and a valid match sequence $o = (o_k)_{0 \leq k \leq m}$ for a priority list sequence seq, the transformation sequence $\mathit{ANI}_0 \xRightarrow{\mathit{seq}, o} \mathit{ANI}_m$ is defined as the combined sequence of direct transformations $\mathit{ANI}_{k-1} \xRightarrow{\varrho_k, o_k} \mathit{ANI}_k$ for all o_k in o, such that ϱ_k is the rule/interaction scheme corresponding to the (maximal disjoint) match o_k into ANI_{k-1}. △

The basic idea of a handler for a request like $\mathit{handler}(\mathit{someReq}) = e_1; e_2; e_3$ is that when a token with the constructor *someReq* occurs by firing a transition in the platform net for which *handler* is defined, we take the first (left-most) rule in the priority list e_1 that is applicable to the platform net and apply it; if there is no such applicable rule in e_1, we skip this list without applying any rule. Then we proceed with e_2 and e_3. Note that at most one rule of a priority list is applied to the platform net for each request! Because every sequence element is considered to be a priority list — single rules are considered as singleton lists —, every handler is processed till the end and yields a unique resulting reconfiguration depending only on the possible matching of its elements, which may be the empty reconfiguration without effect.

The handle conditions ensure that every rule in the handler demands exactly one token of the handled type to be present for matching, which allows to assume that all rules are applied to the same request.

Example. For an example of a handler that avoid multiple channels for contact exchange in Skype, we refer to Sect. 7.2.4.1 of the case study on Skype. The handler specifies a rule for discarding the request tokens for a contact exchange if needed and a lower prioritized rule that actually creates the contact exchange channel.

Remark. For transformation systems, there exist some control structures such as transformation units for graph transformation systems [KK99], which are very powerful with operations such as "as long as possible", nondeterministic choice, and regular expressions [Kus98], or the layering of rules in the AGG tool [Tae04] that could be generalized to transformation systems of other structures than graphs. Sequences can be realized by simple layering or prioritizing of rules. The choice of the first applicable rule in a priority list could possibly be expressed with nested if-then constructs of transformation units together with suitable graph class expressions as conditional predicates characterizing the rules' applicability. Anyway, we use our minimalistic definition and compact notation. An advantage of our simple expressions is that we do not have any termination issues for the handler reconfigurations.

Another recent approach for controlling rule applications is presented in [EGLT11], where an "ActiGra" is a graph transformation system with control flows defined by UML-style activity diagrams that consist of transformation rules as nodes. An activity diagram represents a use case in the sense that the rules are applied according to possible runs through the diagram. Unfortunately, it is not possible for the control flow to decide whether another rule has been applied previously so that is it not clear how to use this approach for defining prioritized rules as we need them.

5.3. Communication Platform Models

We discuss some slight modifications to the concepts and definitions we have so far and give some hints on simplifying the usage of the quite general theory. These modifications are necessary to integrate them properly to a definition of Communication Platform models.

Signatures with token variables In the general definition, a signature for AHLI nets contains a family of variable sets X that can be used for arc inscriptions and firing conditions. This kind of variables is related to the firing behavior of the net. To be able to define flexible reconfiguring transformation rules that can match not only tokens of certain values but rather have token markings with variables that are matched/assigned to values of tokens in the actual platform net, we also need to specify variables for the token values. As a preparation for such rules, define a common family Y of variable sets that may be used as tokens values in rules for this platform model. In the tabular notation of signatures as in Table A.1, we define the arc inscription variables as usual in the section after the keyword **vars** and the additional token variables after the new keyword **tokenvars**. Note that the variable sets X and Y do not necessarily have to be disjoint, but with regard to model clarity this is a preferable property.

Reserved variable expressing transition ownership From the signature with token variables in Table A.1, we choose the variable u of type ID as the reserved variable we use for inscripting arcs from the identity place of a client to the transitions that are supposed to be performed by the user of this client (cf. Sect. 5.1.1.1).

Rules with token variables There are different approaches on how to use rules for reconfiguration. The simplest one is to model the desired effect directly, i. e., the rule has the same algebra as the target net and in the LHS the tokens have concrete values to be matched. If we used this approach, we would have to define a rule for every possible request token in order to have it handled correctly! This is not an elegant way of defining platform models so that we use a different algebra than the actual platform net's algebra, which allows us to formulate rules with variables for token values. We prepared this approach with the explicit definition of token variables in the signature. A matching of the rule then assigns concrete values found in the target net to these variables.

Formally, for a rule ϱ we first choose a family $Y_\varrho = (Y_{\varrho,s})_{s \in S} \subseteq Y_\Sigma$ of variable sets from the token variables of the signature that we want to use on the rule net places. We then define the algebra for the LHS, interface, and RHS of ϱ as $A_\varrho = T_\Sigma(Y_\varrho)$, the Σ-term algebra over the variables of Y_ϱ.

In addition to the structural parts such as places, transitions, and tokens, a match morphism o for ϱ into a net ANI needs an algebra homomorphism $o_A \colon T_\Sigma(Y_\varrho) \to A_{ANI}$ that is compatible to the other morphism components according to Def. 4.2.3. Because the term algebra of a set of variables is a free construction over the set of variables, such an algebra homomorphism is already induced by a token variable assignment $asg : Y_\varrho \to A_{AN}$. Intuitively, for matching a rule with token variables it is sufficient to assign values of A_{AN} to the rule's variables Y_ϱ that is consistent with the match components for places, transitions, and tokens. Note that we only consider almost injective matches (cf. Sect. 5.2.1.1, Def. 5.2.4).

Representation of rules The definition of AHLI transformation rules (and interaction schemes) is very general regarding the rule morphisms. Pushouts are unique up to isomorphisms, so all nets of a DPO transformation rule (LHS, interface, RHS) can be exchanged by isomorphic nets and the rule is still applicable to the same matches (up to an isomorphism) with the same possible results. The most intuitive way of formulating a rule is certainly by choosing inclusions as the rule morphisms. Assuming this representation (that is equivalent to all isomorphic rules), we can understand a rule's effect by its nets without the rule morphisms being given explicitly. For this, we can restrict our rules w. r. t. practical aspects without considerably losing expressiveness for the modeling approach, but instead we gain possibilities for simpler notations. Even the interface is not needed in this case because it is just the intersection of the LHS and the RHS. In the following, we assume the rules to consist of inclusion $\mathcal{M}$-morphisms so that we can denote them simply by depicting just the LHS and RHS.

Minimal changes on rule application The behavior of AHLI nets (and general Petri nets) is defined as firing steps of transitions depending on tokens and their values and we are usually interested in examining this behavior, for reconfigurable systems even together with interleaving reconfigurations. If in the result of a DPO transformation the transitions, tokens, and possibly values are exchanged isomorphically — i. e., basically they are renamed — even if they are being preserved by the rule application, it is considerably harder to relate the behavior of the resulting Petri net to the previous one. Usually, when computing the results of DPO transformations in graphs and Petri nets, a set-based construction of pushouts just removes the deleted parts and adds the newly created ones without altering the preserved ones. In this case, the horizontal morphisms in a DPO diagram that are parallel to the rule morphisms are inclusions, too. For an intuitive representation of the results of AHLI net transformations and to obtain intuitive results on the relation of possible firing behavior before and after reconfiguration, we assume the application of rules to properly maintain the preserved parts and call this kind of transformation *along inclusions.*

User actions We incorporate these simplifications into a formal definition of user actions in a Communication Platform models.

Definition 5.3.1 (User Actions in Communication Platform Models)
For a given Communication Platform model with a current configuration ANI_0, a *user action* is

1. a firing step $ANI_0 \xrightarrow{t,asg,S} ANI'_0$ such that $post(t)$ does not contain any term of a request sort in S_{Req}, or

2. a firing step followed by a transformation sequence $ANI_0 \xrightarrow{t,asg,S} ANI'_0 \xRightarrow{seq,o} {}^{*}ANI_1$ such that $post(t)$ contains a term of a request sort over a request constructor $req \in Constr(S_{Req})$, o is a valid match sequence for $seq = handler(req)$ on ANI'_0, and every transformation of (seq, o) is along inclusions. △

A problem arises if there does not exist a valid match for the handling rule sequence *seq*. Regarding the semantics given for sequences of priority lists (see Sect. 5.2.3), we have a valid application of a handler expression if neither of its rules is applied because a single rule is considered as a priority list of length 1 and thus it is an optional rule in the handler expression. Intuitively, we would not consider the handling successful in this case where no reconfiguration is carried out at all! Rather, we would expect that an exceptional case occurred that the platform designer has not foreseen, which leads to an undefined system configuration. Another reason for unhandled requests can be a transition that produces more than one request token in a firing step. But this can also be avoided more easily than faulty reconfigurations.

On the other hand, there might be special modeling cases in which request tokens are used for further transformations in user actions after the request-producing user action. To be able to test the success of a handling in appropriate cases when we need it, we define the property of success for user actions separately.

Definition 5.3.2 (Success of User Actions)
Given a user action with a handling in a platform model, i. e., a firing step followed by a transformation sequence $ANI_0 \overset{t,asg,S}{\rightarrowtail} ANI_0' \overset{seq,o}{\Longrightarrow}{}^* ANI_1$, we call the user action and the handler expression *successful* if ANI_1 does not contain any token of the request sorts in S_{Req}. △

Simple interaction schemes For Communication Platform models, it should be sufficient to define and use interaction schemes with just a single multi rule like the multicasting scheme in Sect. 5.2.2. Especially in combination with token request handlers, the platform designer can model some consecutive interaction schemes in a handler expression, one for each type of multi action that should be performed for a kind of net parts to be reconfigured. We do not formally restrict interaction schemes to simple schemes, but it is strongly advisable to use the possibilities of handler expressions in combination with simple schemes to be clear about the effects of maximal disjoint matching of the interaction schemes.

Rules for external actions Rules that are not part of a handler expression for any request token constructor can never be applied as part of a user action in a Communication Platform. This is reasonable because our definition of Communication Platforms demands that "every reaction of the system is triggered directly by the users' actions", where we can read "reaction" as "reconfiguration" and "users' action" as "transition firing that possibly creates a request token". In a concrete platform, there might be some exceptions for this principle, especially because we consider Communication Platforms (as Spaces) to be dynamic in their structure. Thus, one of these exceptions we find in almost every Communication Platform is the creation of a client for a new user. Usually, this kind of action is not performed by other users but by the new user itself, but we simply cannot provide an initial transition representing this action. For such kind of external actions that are performed by actors that are not (yet) part of the platform, we provide for *external actions* in a platform model, which we consider as the rules and interaction schemes that do not occur in the handler definition. Therefore, we allow the application of rules for external actions to happen at any time in the platform model.

On the other hand, we regard the application of rules and interaction schemes in the handler expressions as bound to triggering firing steps, such that they may not be performed as external actions but rather as part of user actions, only.

With these clarifications, we can finally combine the concepts and definitions from this chapter to a definition of Communication Platform models.

Definition 5.3.3 (Communication Platform Models)
A Communication Platform model is a tuple

$$CPM = (\Sigma, A, ANI, R, IS, S_{Req}, handler), \text{ where}$$

- $\Sigma = (S, OP; X, Y)$ is a signature with a family of variable sets $X = (X_s)_{s\in S}$ for arc inscriptions and a family of variable sets for token values $Y = (Y_s)_{s\in S}$, encompassing the common signature Σ_{CP} for Communication Platform models (Table 5.1),
- A is a Σ-algebra for the Communication Platform,
- ANI is the platform AHLI net over Σ, A representing the platform's current configuration,
- R is a set of AHLI transformation rules, each over its term algebra $T_\Sigma(Y_\varrho)$ over a token variable set $Y_\varrho \subseteq Y$ and with variable application conditions (Def. 5.2.5),
- IS is a set of loose interaction schemes (Def. 5.2.12), each over its term algebra $T_\Sigma(Y_\varrho)$ over a token variable set $Y_\varrho \subseteq Y$ and with variable application conditions,
- $handler\colon Constr(S_{Req}) \rightarrow PSEQ(R \cup IS)$ is a token request handler (Def. 5.2.17) for a set $S_{Req} \subseteq S$ of request sorts. △

5.4. Higher-Order Nets for Simulating and Controlling Communication Platform Models

Although the definition of Communication Platform models contains — besides transformation rules and interaction schemes — a token request handler, which loosely relates reconfiguration sequences to signature terms, we have not yet defined a formal semantics for platform models regarding our definition of user actions (Def. 5.3.1). Intuitively, as the operational semantics of a Communication Platform model, we consider all user actions and external reconfigurations that can occur in it. We give the construction of an AHOI net (see Sect. 4.6) for a given Communication Platform model Sect. 5.2.3 that — besides the normal firing and reconfiguration of an AHLI platform net (see Sect. 4.2) given as a token — additionally controls the handling of requests (see Sect. 5.2.3). The firing behavior of the resulting AHOI net to yield the simulation semantics of the underlying Communication Platform model.

Based on an idea presented in [HM10], we now consider an extension of the AHO approach in [HEM05] to express with an AHOI net that actions in platform AHLI nets posing requests must be handled immediately by reconfiguring rule applications. For an AHOI net that contains platform AHLI nets as tokens, we have to extend the definitions of the sorts *System* and *Rules* from P/T nets as in Fig. 4.11 to AHLI nets (with firing steps, rules, and interaction schemes with maximal matching), respectively. The signature Σ_{AHOI} in Table 5.2 is fixed for all control AHOI nets that we construct in

$\Sigma_{AHOI} =$			
sorts:	$System, Rules, Trans, VarAsg, Sel, Mor, ReqConstr, MorSeq, Bool$		
opns:	$T, F\colon \to Bool$		
	$isEnabled\colon System\ Trans\ VarAsg\ Sel \to Bool$		
	$fire\colon System\ Trans\ VarAsg\ Sel \to System$		
	$producesReq\colon Trans\ System \to Bool$		
	$cod\colon Mor \to System$		
	$isApplicable\colon Rules\ Mor \to Bool$		
	$apply\colon Rules\ Mor \to System$		
	$producesReq\colon Trans\ System\ ReqConstr \to Bool$		
	$isHandleable\colon System\ ReqConstr\ MorSeq \to Bool$		
	$handle\colon System\ ReqConstr\ MorSeq \to System$		
vars:	$n, n'\colon System$	$r\colon Rules$	$t\colon Trans$
	$asg\colon VarAsg$	$s\colon Sel$	$o\colon Mor$
	$req\colon ReqConstr$	$os\colon MorSeq$	

Table 5.2.: Signature Σ_{AHOI} for platform-controlling AHOI nets

the following for a given Communication Platform model. Several of its part resemble the operations used in Fig. 4.11.

Now, we consider a concrete Communication Platform model

$$CPM = (\Sigma_{ANI}, A_{ANI}, ANI, R, IS, S_{Req}, handler)$$

to build a control AHOI net for. The algebra Σ_{AHOI}-algebra A^{AHOI}_{CPM} of this particular AHOI net depends on the given platform model CPM. We explain only the important details informally here. For a formal definition, we refer to Table A.3.

Sorts of Σ_{AHOI} The sort $System$ represents all AHLI nets over the signature Σ_{ANI} and the algebra A_{ANI}. The carrier set for the sort $Rules$ contains all AHLI net rules over a Σ_{ANI}-term algebra with a distinct set of variables for each rule and with variable application conditions (see Def. 5.2.5). Also, $Rules$ contains all loose interaction schemes (see Def. 5.2.12) that can be built over these rules. The sort $Trans$ represents a universe of transitions for the AHLI nets in sort $System$. The sort $VarAsg$ is intended for all possible variable assignments for the arc variables of Σ_{ANI}, and the sort Sel for all possible token selections in firing steps. The almost injective (multi) matches for the applications of the rules and interaction schemes of the sort $Rules$ are gathered by the sort Mor. For rule sequences of token request handlers, the sort $MorSeq$ holds all sequences of the matches in Mor. The carrier set for the sort $ReqConstr$ is defined as the set $Constr(S_{Req})$ of token constructors of CPM.

Operations of Σ_{AHOI}

$isEnabled(n, t, asg, s), fire(n, t, asg, s)$ These operations are used for simulating the firing of a transition t in an AHLI net n with the variable assignment asg under the selection s. The operation *isEnabled* decides whether the firing step $n \xrightarrow{t,asg,s} n'$ exists according to Def. 4.2.2, and *fire* yields n' if this is true.

$producesReq(t, n, req), producesReq(t, n)$ With these operations, we can check if a transition t in a net n produces a token of any request constructor (or of a specific constructor req) from a request sort in S_{Req}. We need these operations to decide whether a firing step in a user action requests a reconfiguration by a token request handler.

$cod(o)$ This operation yields the codomain of a match or the common codomain of a bundle of matches, depending on the nature of the argument o.

$isApplicable(r, o), apply(r, o)$ For simulating the application of a rule or interaction scheme r of the sort *Rules* at some almost injective (multi) match o, we use first *isApplicable* to decide whether r can be applied at o, i. e., the gluing condition for AHLI nets is satisfied (see Def. 5.2.6) and o structurally satisfies the variable application condition(s) of r (see Def. 5.2.6). The operation *apply* then yields the result n' of the (amalgamated) transformation $cod(o) \overset{r,o}{\Longrightarrow} n'$, which we demand to be along inclusions to get a unique result.

$isHandleable(n, req, os), handle(n, req, os)$ This pair of operations works similar as the ones for applying rules and interaction schemes. Instead of a rule or interaction scheme, a request constructor req determines a rule sequence $seq = handler_{CPM}(req)$ with the handler function of CPM. The operation *isHandleable* checks whether os is a valid matching sequence for seq, and *handle* yields the result n' of the transformation sequence $n \overset{seq,os}{\Longrightarrow} n'$ if this is true (see Def. 5.2.19).

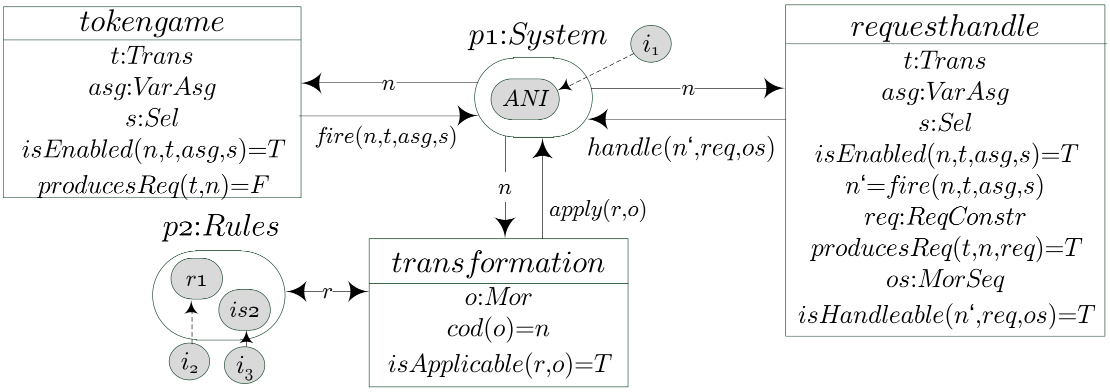


Figure 5.8.: Algebraic higher-order net for a Communication Platform model

We construct the controlling AHOI net for the platform model CPM over this algebra as shown in Fig. 5.8. For this example, we assume that R contains exactly one rule r_1 and that IS contains exactly one interaction scheme is_2 that both do not occur in any request handler definition, i. e., they are external rules and schemes according to Sect. 5.3. As the AHOI net's initial marking, we set the AHLI platform net ANI on place p_1 and the external rules and interaction schemes as tokens on place p_2. In the AHOI net in Fig. 5.8, there are three main conceptual differences to the AHO net for reconfigurable P/T nets in Fig. 4.11.

1. The transition *tokengame* now only fires transitions in the token AHLI net ANI that are not producing a request token or can be understood as a signaling request in any way. This is decided by the condition $\mathit{producesReq}(t, n) = F$, which ensures that the transition t in n does not create any request token over a request sort from S_{Req}. Note that for the AHLI net tokens on place p_1, we also need to consider a valid variable assignment for a firing step and a valid token selection.

2. The transition *transformation* is basically the same as in the control net for reconfigurable P/T nets in Fig. 4.11. But, because we set only the external actions of CPM on place p_2, by firing this transition only external rules and interaction schemes can be applied to ANI on place p_1.

3. Firing steps of the other transitions in the platform net ANI that trigger a handling sequence are simulated via the higher-order transition *requesthandle*. In addition to the firing of a transition t resulting in the net n', the firing condition $\mathit{producesReq}(t, n, \mathit{req}) = T$ of *requesthandle* ensures that the firing of t in n produces a token of a request constructor. Moreover, the variable os must be bound to a valid match sequence (see Def. 5.2.18) for the handler expression of $\mathit{handle}(\mathit{req})$ into the net n', which is checked by the operation $\mathit{isHandleable}(n', \mathit{req}, os) = T$. The term $\mathit{handle}(n', \mathit{req}, os)$ on the arc back to p_1 denotes the net n' after $\mathit{handle}(\mathit{req})$ has been applied at the valid match sequence os (see Def. 5.2.19).

It is easy to see that every firing step of the control AHOI net simulates either a user action (according to Def. 5.3.1, with or without handling reconfiguration) or an application of an external rule in the net ANI on place p_1.

6. Validation Techniques for Communication Platform Models

It might be noted here, for the benefit of those interested in exact solutions, that there is an alternative formulation of the many-body problem, i. e., how many bodies are required before we have a problem? G. E. Brown points out that this can be answered by a look at history. In eighteenth-century Newtonian mechanics, the three-body problem was insoluble. With the birth of general relativity around 1910 and quantum electrodynamics in 1930, the two- and one-body problems became insoluble. And within modern quantum field theory, the problem of zero bodies (vacuum) is insoluble. So, if we are out after exact solutions, no bodies at all is already too many!

(Richard D. Mattuck)

In this chapter, we treat the requirement for formal means that allow us to formulate and validate model properties that we posed in Sect. 3.2.7 based on the considerations of model properties in Sect. 3.1.7 for the concrete system Skype and in Sect. 3.2.1 for Communication Platforms in general. The validation during runtime — i. e., while simulating a platform model — aims at diagnosing and locating problems in a platform model.

For expressing and checking conditions for the integrity of a Communication Platform model, we present platform constraints in Sect. 6.1, which are similar to the technique of application conditions in Sect. 5.2.1.

Afterwards, we discuss functional behavior of user actions in Sect. 6.2, which leads to a formal functionality condition for the token request handlers from Sect. 5.2.3 we use in Communication Platform models as control structure for reconfiguring rule applications triggered by request tokens. Although functional behavior is not directly motivated by properties of concrete platforms, it is an important property of the formal modeling approach and the behavior occurring in it, e. g., to avoid unintended reconfigurations that result from underspecification of the rules' application conditions.

In Sect. 6.3, we finally examine the independence of behavior in a Communication Platform model, i. e., independence of transition firings and possibly triggered reconfigurations in user actions. For this, we give conditions for checking and diagnosing the (in)dependency of two given user actions w. r. t. the modeling intentions. The functionality of user actions is a notably helpful property for the concepts of user action independence.

6.1. Platform Constraints

In Sect. 5.3, we present a concrete approach for modeling Communication Platforms according to the modeling principles in Sect. 3.2.6. For the approach of algebraic high-level nets without reconfiguration, we could possibly give guidances for modeling systems that avoid firing steps breaking some fundamental principles, e. g., such as the unique token identifying a user on the dedicated identifier place. A simple possibility for this, which we describe in Sect. 5.1.1.1, is that the incoming and outgoing arcs incident to a transition must be inscribed exclusively by a reserved variable so that no firing can ever change the identifier token. But with reconfigurations, the modeling approach becomes much more powerful and the modeler might formulate defective reconfiguring rules that break the integrity, e. g., by simply deleting the identifier token.

We could restrict the possible rules a modeler may formulate, e. g., to comply to the demand for a unique owner of a transition (also cf. Sect. 5.1.1.1). It is easy to see that if the nets of a rule are not violating this integrity condition, then no application of this rule can break the integrity of a valid platform AHLI net w. r. t. this property. However, limiting the possible transformation rules with many ad-hoc conditions like “if the left-hand side matches x with an environment y, the right-hand side must preserve/contain z” for each single *model-dependent* integrity condition is not an option because these conditions cannot be formulated on the abstract level of Communication Platforms as the Skype role constraints from Sect. 3.1.7, and it is not advisable to change the general approach for every concrete modeling endeavor.

Instead, for the model integrity properties as in Sect. 3.2.1 and Sect. 3.1.7, we use the technique of nested constraints (based on graph constraints) from [HP09] to formulate these conditions for having them checked after every configuration change in the platform. Nested constraints are similar to application conditions for rules (cf. Sect. 5.2.1) with the difference that they are satisfied by objects (i. e., AHLI nets) instead of morphisms (i. e., matches for the left-hand side of a rule). With nested constraints, we can formulate conditions that either must or must not hold in an AHLI net (to be proven by a suitable morphism) if a precondition holds (given by a match-like morphism), as for the applicability of rules.

6.1.1. Formal Constraints based on Nested Conditions

We first review the definition of constraints from [HP09] based on the nested conditions that are used as application conditions for transformation rules.

Definition 6.1.1 (Constraints and Satisfaction)
A *constraint* is a nested condition (Def. 5.2.1) over the initial object $\emptyset_{T_\Sigma}$, which is the empty AHLI net over the Σ-term algebra T_Σ. An object N *satisfies* a constraint ac, written $N \vDash ac$, if $i_N \vDash ac$, where i_N is the unique morphism from $\emptyset_{T_\Sigma}$ to N. △

Similar to the definition of constraints for objects in [HP09], we define structural constraints and structural satisfiability by objects based on the variable application conditions that we use for rules in transformation systems for Communication Platforms (see Sect. 5.2.1).

Definition 6.1.2 (Structural Constraints)
A *structural constraint* is a condition ac over the initial object $\emptyset_{T_\Sigma}$, with

- $ac = \mathit{true}$
- $ac = \exists(i_P, ac')$, where $i_P \colon \emptyset_{T_\Sigma} \to P$ is the initial morphism to P and ac' is a variable condition over P according to Def. 5.2.5, or
- ac is some Boolean construction over the previous cases, analogously to nested conditions.

An object N *structurally satisfies* a structural constraint ac, written $N \models_S ac$, if $i_N \models_S ac$, where i_N is the unique morphism from $\emptyset_{T_\Sigma}$ to N. △

Remark (Notation). It is easy to see that a constraint $\exists(i_P, ac')$ is structurally satisfied by an object N if and only if there exists some morphism $p\colon P \to N$ with $p^S \in \mathcal{M}$ and $p \models_S ac'$: Because the initial morphism $i_N \colon \emptyset_{T_\Sigma} \to N$ is unique, we have for every morphism $p\colon P \to N$ that $p \circ i_P = i_N$. It remains to check all morphisms $p\colon P \to N$ with $p^S \in \mathcal{M}$ if they structurally satisfy the subconstraint ac'. Similarly, $\forall(i_P, ac')$ is structurally satisfied by an object N if and only if all morphisms $p\colon P \to N$ with $p^S \in \mathcal{M}$ structurally satisfy ac'.

Because of this and the uniqueness (up to isomorphism) of the initial object, we use a shorter notation for these constraints. For $\exists(i_P, ac')$, we may simply write $\exists ac'$ and for $\forall(i_P, ac')$ we write $\forall ac'$ without loss of information. The subconstraint ac' typically has a leading quantifier (if it is not a Boolean combination) so that constraints in short notation characteristically start with two quantifiers.

Remark (Constraints related to conditions). Note that the definition of constraints as conditions over an initial object from [HP09] differs significantly from the definition of graph constraints given in [EEPT06]. The main difference is that the definition from [EEPT06] explicitly require that every morphism $p\colon P \to N$ satisfies the inner constraint ac'. Therefore, with the variant from [EEPT06], we could only express constraints like $\forall ac'$ (in short notation), because it is impossible to formulate constraints equivalent to $\exists ac'$. By using the variant with "initial satisfaction" from [HP09], we get that constraints can be defined directly using the definition of (application) conditions as in Defs. 6.1.1 and 6.1.2.

We give examples in Sect. 6.1.2, after showing that structural satisfiability $\models_S$ of a constraint by AHLI nets is equivalent to normal satisfiability $\models$ where the constraint is shifted along a suitable variable assignment.

Theorem 6.1.3 (Structural Satisfaction of Structural Constraints)
Given a structural constraint $ac = \exists(i_P \colon \emptyset_{T_\Sigma} \to P, ac')$ with $A_P = T_\Sigma(Y)$ for some variable set Y. For all objects N holds the equivalence

$$N \models_S ac \Leftrightarrow \exists asg\colon Y \to A_N \colon N \models \exists(i_{P\uparrow\overline{asg}}, \mathit{DShift}(\overline{asg}, ac'))$$

where $\overline{asg}\colon T_\Sigma(Y) \to A_N$ is the free extension induced by the variable assignment asg.

Proof.

"$\Rightarrow$:" If $N \models_{\mathrm{S}} ac$, we have $i_N \models_{\mathrm{S}} ac$ with some $q\colon P \to N$ such that $q \circ i_P = i_N$, $q^S \in \mathcal{M}$, and $q \models_{\mathrm{S}} ac'$. Consider $\overline{asg} = q_A$. From Theorem 5.2.10, we get that $q^S \models \mathit{DShift}(q_A, ac) = \mathit{DShift}(\overline{asg}, ac)$. Because of the initiality of $i_{P\uparrow\overline{asg}}$, we have $i_{P\uparrow\overline{asg}} = id_P^{+\overline{asg}} \circ i_P = q_A \circ i_P$ and hence $q^S \circ i_{P\uparrow\overline{asg}} = q \circ i_P = i_N$, from which we conclude $N \models \exists(i_{P\uparrow\overline{asg}}, \mathit{DShift}(\overline{asg}, ac'))$.

"$\Leftarrow$:" Assume an $asg\colon Y \to A_N$ such that $N \models \exists(i_{P\uparrow\overline{asg}}, \mathit{DShift}(\overline{asg}, ac'))$, i. e., there exists an $\mathcal{M}$-morphism $q\colon P^{\uparrow\overline{asg}} \to N$ with $q \circ i_{P\uparrow\overline{asg}} = i_N$ and $q \models \mathit{DShift}(\overline{asg}, ac'))$.

Consider the morphism $\hat{q} = q \circ id_P^{+\overline{asg}}\colon P \to N$. Because of the initiality of $i_{P\uparrow\overline{asg}}$, we have $i_{P\uparrow\overline{asg}} = id_P^{+\overline{asg}} \circ i_P$ and therefore $\hat{q} \circ i_P = q \circ i_{P\uparrow\overline{asg}} = i_N$. Because $q \in \mathcal{M}$, q_A is isomorphic, and we get from Lemma A.3.3 that $q^S \models \mathit{DShift}(q_A \circ \overline{asg}, ac'))$ and from Theorem 5.2.10 that $\hat{q} = q \circ id_P^{+\overline{asg}} = \models_{\mathrm{S}} \mathit{DShift}(id_{A_N}, ac') = ac'$. Finally, $\hat{q}^S \in \mathcal{M}$, because from $q \in \mathcal{M}$ we have $q^S \in \mathcal{M}$ and it is $q^S = \hat{q}^S$.

Hence, $\hat{q}$ proves that $i_N \models_{\mathrm{S}} ac$ and $N \models_{\mathrm{S}} ac$. □

6.1.2. Integrity Conditions of Communication Platforms by Platform Constraints

For an example of how to validate integrity conditions with platform constraints, consider the AHLI net in Fig. 6.1. This example net N shows a minimalistic platform with three clients that have just one possible action *activate*, similar to the one shown in Fig. 4.7 and with number suffixes for the different clients. Additionally, the transition *activate* is connected via arcs inscribed with the reserved user identity variable u to a place that should carry a token of the type *ID*, according to Sect. 5.1.1.1.

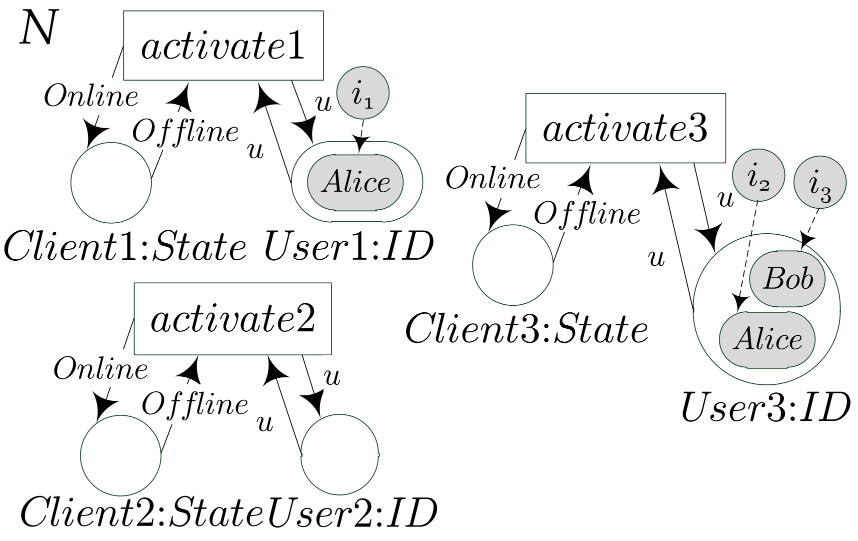


Figure 6.1.: Broken integrity in example AHLI net N for platform with three clients

In N, the second and third client have inconsistent identifiers because the second client does not have any identifier token and the third one has two. Both cases are problematic, because the third client is not addressable unambiguously and the second one is not even addressable at all by other clients that want to send messages to him. It is even worse, as the third client shares an identifier value with the first client. We want to avoid such configurations by formulating and checking platform constraints.

To some extent, the property of unique and pairwise different identities is model-independent, i. e., it is reasonable to demand it for all kinds of Communication Platform models. If we want to formulate constraints that should hold for all clients, we need to pinpoint a core part of a client that is characteristical for a client. In most concrete models, there are actions that belong to an immutable core structure — i. e., they are intended to be present in all possible reconfigurations — as *activate* in the example net N. If we consider a structure as a client if and only if it contains a transition as *activate* connected to the identifier place as in N, we can formulate a precondition for platform constraints that locates this part and then demand that extensions must or must not exist. We give two examples of platform constraints for validating whether the described problems of identifier tokens occur in a platform net configuration.

Example (Platform Constraint for unique identifier tokens). First, we define a constraint ac_1 that is satisfied by any net modeling a concrete platform type as in N, in which the user identity place carries exactly one identifier token.

Consider the condition $ac_1 = \forall(\exists a \wedge \neg\exists a')$ with the morphisms a, a' as shown in Fig. 6.2 that comply to the definition of variable conditions in Def. 5.2.5. This is the shorthand notation for the constraint $\forall(i_P\colon \emptyset_{T_\Sigma} \to P, \exists a \wedge \neg\exists a')$, for which we now check where N violates it.

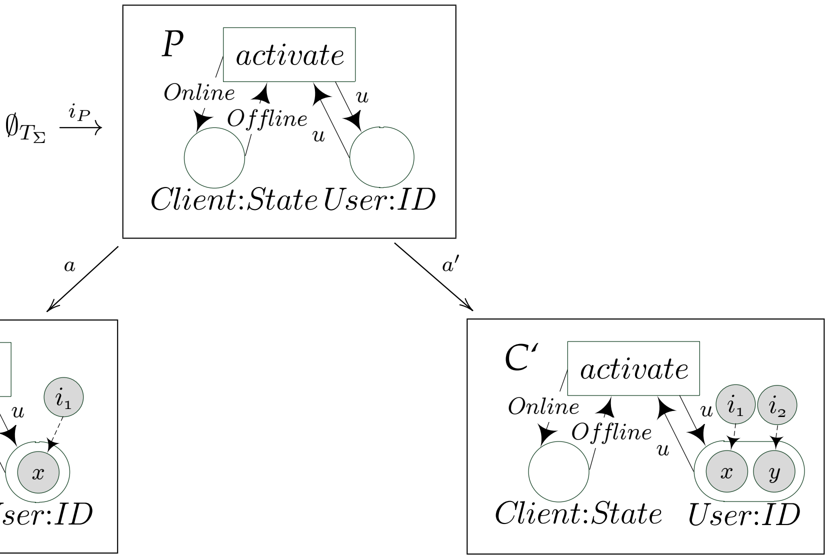


Figure 6.2.: Morphisms of unique identifier token constraint

According to Def. 6.1.2, we have $N \vDash_S ac_1$ if $i_N \vDash_S ac_1$. This means we have to check all morphisms $p\colon P \to N$ with $p^S \in \mathcal{M}$ if $p \vDash_S \exists a \wedge \neg\exists a'$ holds. There are only three different morphisms $p_1, p_2, p_3\colon P \to N$, each mapping the transition *activate* and its environment to the transition *activate* $\langle n \rangle$, according to its index number n. All these morphisms p_i are injective on the net structure part and therefore $p_i^S \in \mathcal{M}$. We check each of these separately:

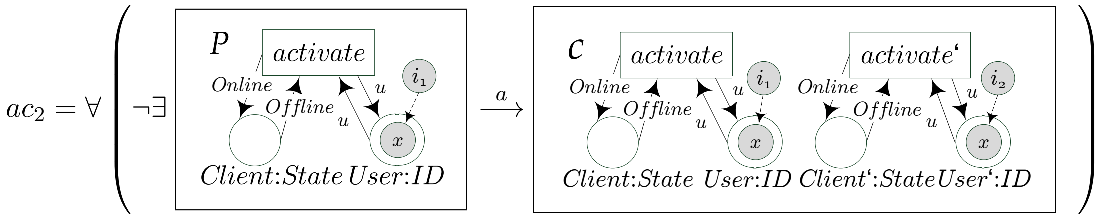
p_1 **satisfies** $\exists a \wedge \neg\exists a'$ Consider the morphism $q\colon C \to N$ that maps the net structure just as p_1 and the token i_1 to $q_I(i_1) = i_1$ (with $q_A(x) = \mathit{Alice}$). We have $q^S \in \mathcal{M}$ and that $q \circ a = p_1$, therefore $p_1 \vDash_S \exists a$. But there does not exist an morphism $q'\colon C' \to P$ such that $q'^S \in \mathcal{M}$ and $q' \circ a = p_1$ because q' would have to be injective in its token component q'_I and there is only one token available for mapping on the place `User1` to map both tokens i_1, i_2 of C' to. Therefore, also $p_1 \vDash_S \neg\exists a'$.

p_2 **violates** $\exists a$ There does not exist a morphism $q\colon C \to N$ with $q \circ a = p_2$ because q needs to map the net structure as p_2 does but there is no token available on place `User2` to map i_1 to. Therefore, p_2 violates the subconstraint $\exists a$.

p_3 **violates** $\neg\exists a'$ There does exist a morphism $q\colon C' \to N$ with $q \circ a' = p_3$ that maps the net structure just as p_3, the token i_1 to $q_I(i_1) = i_2$ (with $q_A(x) = \mathit{Alice}$), and the token i_2 to $q_I(i_2) = i_3$ (with $q_A(y) = \mathit{Bob}$). On the net structure and on the tokens, q is injective and hence $q^S \in \mathcal{M}$. Therefore, p_3 violates the subconstraint $\neg\exists a'$.

Hence, N violates ac_1 exactly because of the cases violating the condition of a unique identifier token on the identity place. The first client complies to the integrity condition, but the second and third do not. The subcondition $\exists a$ demands that there exist at least one identifier token in the client determined by the precondition P, and the subcondition $\neg\exists a'$ forbids that there exist more than one identifier token in the same client. If the places `User2` and `User3` carried exactly one token as `User1`, we would have $N \vDash_S ac_1$.

Example (Platform Constraint for pairwise different identifier tokens). This example shows how to validate whether in a platform the token values on the clients' identifier places are pairwise different, i. e., all users have different names in the platform. Consider the following platform constraint:



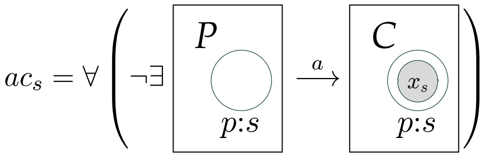
It is easy to see that the example platform net N from Fig. 6.1 violates the platform constraint ac_2, which demands that for every morphism $p\colon P \to N$ with $p^S \in \mathcal{M}$ there do not exist a morphism $q\colon C \to N$ with $q^S \in \mathcal{M}$ such that $q \circ a = p$. We look at p_1 that maps the transition *activate* and its environment from P to the transition *activate1* in

N, including the token i_1, which is mapped to $p_{1,I}(i_1) = i_1$ together with $q_A(x) = Alice$. Now, there exists the morphism $q\colon C \to N$ that maps the transition *activate′* and its environment from C to the transition *activate3* in N, the token i_2 to $q_I(i_2) = i_2$, and the rest just as p_1. Obviously, $q^S \in \mathcal{M}$ because it is injective on the net structure and the individual tokens, and $q \circ a = p$. Therefore, p_1 violates $\neg\exists a$ and hence N violates ac_2. If there were not a second token with the value *Alice* on any other place than on `User1` in N, then N would satisfy ac_2.

If a net (structurally) satisfies both of the platform constraints ac_1 and ac_2, we can consider it to comply to the integrity condition of unique identifier tokens. Of course, one could argue that also the state places should be marked with exactly on token value (online, offline etc.), each. A constraint that covers this issue would be very similar to the presented examples. The other examples for model integrity from Sect. 3.1.7 are too model-dependent to be discussed in this section on the abstract level of a minimalistic platform. We show and validate them in the case study on modeling Skype in Sect. 7.4.

6.1.3. Validation of Successful User Actions

Since we regard a handling of a request in a user action successful if the result of the action does not contain any request token, the platform designer can try to carefully model the handling rules such that the request token is deleted as soon as a handling is considered to be successful. If the following rules in the handler — possibly handling other exceptional cases for the request — then demand the request token for a valid matching, they are skipped in this user action handling. If after a user action — including its request handling reconfiguration — the platform net still contains a request token, this is a significant indicator that something went wrong during the handling and should be considered as an undefined state of the platform. Such validation tests for the success of a handling can easily be carried out with platform constraints that forbid the existence of request tokens on any place. Given a set of request sorts S_{Req}, we just need to validate a set of constraints like the following ac_s with a token variable x_s for each sort $s \in S_{Req}$.



If and only if the success constraint $ac_{suc} = \bigwedge_{s \in S_{Req}} ac_s$ is satisfied by the current platform net, then the previous user action has been successful.

6.2. User Action Functionality

When modeling computer and software systems for the everyday user — e. g., like Skype —, the designer usually thinks of a user pushing some real or virtual buttons to let the system perform some action and change its state accordingly. Apart from special use cases such as simulated randomness in games, a user expects the resulting system state to be unique and well-defined, which are fundamental characteristics of a system's usability and user-friendliness concerning its *functionality*. We look at our approach of Communication Platform models (see Sect. 5.3) for possible problems related to functionality that can result from deficient modeling.

From the user's point of view, every action he performs in the platform model is a transition firing. The firing of a transition is unproblematic w. r. t. functionality because of a suitable variable assignment into the available token values and newly created token values in the environment of the transition.

If the firing steps triggers a request-handling transformation sequence with some request token as below, this becomes more complicated.

$$\begin{tikzcd}[column sep=huge]
ANI_0 \arrow[r, tail, "trigger", "\text{produces token } req{\in}Constr(S_{Req})"'] & ANI'_0 \arrow[r, Rightarrow, "handling", "\text{by } handler(req)"'] & {}^{*}\, ANI_1
\end{tikzcd}$$

Rule-based transformation is a powerful technique because it is possible to choose match morphisms for rule applications nondeterministically. Even if we assume that the request handler function defines a single rule to handle the produced request as in

$$\begin{tikzcd}[column sep=large]
ANI_0 \arrow[r, tail, "trigger"] & ANI'_0 \arrow[r, Rightarrow, "handling", "\text{rule } \varrho\text{, match } o_1"'] & ANI_1
\end{tikzcd}$$

there may exist more valid matches leading to valid direct transformation steps:

$$\begin{tikzcd}[column sep=large]
 & & ANI_2 \\
ANI_0 \arrow[r, tail, "trigger"] & ANI'_0 \arrow[ur, Rightarrow, "\varrho\text{, match } o_2"] \arrow[r, Rightarrow, "\varrho\text{, match } o_1"] \arrow[dr, Rightarrow, "\varrho\text{, match } o_3"'] & ANI_1 \\
 & & ANI_3
\end{tikzcd}$$

To be able to tell whether a problem related to functionality occurs during the simulation of a platform model, we define this property for user actions in a Communication Platform model (see Def. 5.3.1) formally.

Definition 6.2.1 (Functionality of User Actions)
Given a firing step $ANI_0 \overset{t,asg,S}{\rightarrowtail} ANI'_0$ in a Communication Platform model that creates a token with a constructor $req \in Constr(S_{Req})$, which causes an immediate transformation $ANI'_0 \overset{handler(req),o}{\Longrightarrow} {}^{*}ANI_1$, such that o is a valid match sequence for the priority list sequence $handler(req)$ (see Def. 5.2.18). We call the transformation and the handler expression *functional on* ANI'_0 if there does not exist another valid match sequence for the priority list sequence $handler(req)$ into ANI'_0 than o. △

Whether a user action is functional can be tested directly by constructing the valid match sequences for the token handler sequence $handler(req)$ on ANI'_0 according to Def. 5.2.18. If and only if at every any point during the construction of the match sequence there exist at most one possible match, then the user action is functional.

Remark (Modeling aspects of functional handlers). The functionality condition demands that for every matched rule of the handler expression there exist only this particular valid match. If this property is violated for a handling sequence of a triggering firing step, this user action probably can have ambiguous results depending on the chosen matches for the match sequence. As said above, functionality and unique results are expected for most user actions. If the platform designer becomes aware of a nonfunctional handler (due to ambiguities in the possible matches), the reason is most likely an underspecification in the left-hand side of a rule. More detailed application conditions could be a possible solution for this problem. Another possibility to fix such issues and to repair the platform model without interfering too much with the other existing parts is to provide in the request token for this handler more specific data that further restricts the applicability of the handling rules. For this, the constructor operation and the transition producing a corresponding token has to be adapted. This approach is advisable especially if the possible handling results are not per se erroneous w. r. t. the platform so that it should be left to the triggering user which result he prefers. If finally the outcome of the handling is fully determined by the data in the request that is produced by the triggering firing step, the user has full control of its actions and their effects.

6.3. User Action Independence

From the concrete characteristical properties about the mutual influence of actions in Skype (cf. Sect. 3.1.7), we get the general notion of independence of user actions in platform models as an interesting property for validation in Sect. 3.2.1. Intuitively, two actions are independent if they can be performed in any order, leading to the same result. A platform designer usually has an idea of the circumstances under which certain actions should be completely independent or when one action should exclude another one or depend on another one to happen first. If action sequences that are not allowed due to such dependencies can occur in the formal platform model, we clearly should consider it deficient. The same holds if actions that are intended to be mutually independent depend in some way on each other in the formal model.

Inferring dependencies between concrete user actions Usually, in Communication Platforms it is not always possible to state that all concrete occurrences of some actions are independent. For the example of contact exchanges in Skype (see Sect. 3.1.7), we demand that a user can ask any other user for a contact exchange and that these actions do not depend on each other, i. e., that one impedes another one. But, there is the exception that every user can only pose one request for contact exchange to a certain other user at the same time. For almost every action in Communication Platforms, there are similar exceptions for dependence and exclusiveness of action types. Therefore, action (in)dependencies should be validated on representative test cases under the right circumstances represented in the model context. In this sense, the concrete action of asking a specific user for contact exchange in a platform model of Skype should inhibit this action (or its effect) to occur again.

$$\begin{tikzcd}
 & & ANI_0 \arrow[dl, "\mathit{trigger}1"', "\text{produces } req_1"] \arrow[dr, "\mathit{trigger}2", "\text{produces } req_2"'] & & \\
 & ANI'_0 \arrow[dl, Rightarrow, "{\mathit{handler}(req_1),o_1}"', "*" near end] & & ANI''_0 \arrow[dr, Rightarrow, "{\mathit{handler}(req_2),o_2}", "*"' near end] & \\
ANI'_1 \arrow[dr, "\text{produces } req_2", "\mathit{trigger}2"'] & & & & ANI''_2 \arrow[dl, "\text{produces } req_1"', "\mathit{trigger}1"] \\
 & ANI'''_1 \arrow[dr, Rightarrow, "{\mathit{handler}(req_2),o'_2}"', "*" near end] & & ANI'''_2 \arrow[dl, Rightarrow, "{\mathit{handler}(req_1),o'_1}", "*"' near end] & \\
 & & X & &
\end{tikzcd}$$

Figure 6.3.: Independent user actions

Parallel and sequential independence of transformations We have to distinguish different notions of independence, depending on the relation of the considered users action. On the one hand, we may have two actions that can be performed at the same configuration with the question if they can be performed in any order (see the span of user actions in the top of Fig. 6.3). On the other hand, we may have a sequence of actions and the question if the last one can be performed before the first one (see the sequences of user actions in the left and the right part of Fig. 6.3). In both cases, we expect the user actions to lead to the same result configuration if we consider them to be independent. From the domain of graph transformations, we have the notions of parallel and sequential independence for rule applications describing these relations, and the local Church-Rosser theorem for transformations in $\mathcal{M}$-adhesive systems links these (see [EEPT06]). Because our formal definition of user actions in Communication Platform models uses ($\mathcal{M}$-adhesive) double pushout transformations of AHLI nets for the reconfiguring handling of requests, we adapt and extend these independence concepts to user actions.

Grades of user action dependence A main benefit of the local Church-Rosser theorems for graph transformations in [EEPT06] is that for concurring graph rule applications, we know about the existence of the complement (sequentially independent) rule applications if we can show that the given ones are parallel independent, and therefore we do not have to construct and to check the transformation results to know that there is a confluent independency diagram. To check the independence of concurrent or consecutive user actions as in Fig. 6.3, however, we have to take into account all parts of the user actions — the triggering firing step and the possible handling transformation sequence — and their dependencies. For this, we have to construct intermediate steps for all mutual pairings of transformations and firing steps of the two user actions and to check the independence for each of these. It may occur that even if two user actions are not independent, they do not necessarily have to be dependent in the strict sense that neither of their intermediate steps are independent. They may lead to the same result, applying the same sequence of rules from their handler expressions. Or they lead to the same result, but the applied rule sequence differs, depending whether the reconfiguration of the concurring user action takes place before or afterward, which is a mild form of dependency. Even "more dependent" are two user actions if one handling transformation sequence disables (and possibly deletes) the triggering transition of the other user action. We see that for user actions, it is promising to consider more "degrees" of independence and dependence. Validating concrete concurrent or consecutive user actions by checking their degree of (in)dependence can provide valuable information for the diagnosis of unwanted effects if the actual degree of dependence does not match the intention of the platform designer for these user actions.

Note that unlike for independence of graph transformations, we do not get the benefit for user action independence that it is considerably simpler to check the independence conditions instead of trying to perform the given user actions and to check whether the result is isomorphic.

To obtain a formal definition of user action independence, we first need a possibility to decide whether the triggering firing step of one user action (such as $trigger1$ in Fig. 6.3) is independent from the parallel action's firing step (such as $trigger2$) and its handling reconfiguration (such as $handler(req_2), o_2$). In the following, we develop notions of permutability for firing steps and of independence for firing steps and rule applications in AHLI nets. Similar to independence of transformations, a special local Church-Rosser theorem relates parallel and sequential independence of firing steps and rule applications. Based on these results, we then define several degrees of independence of user actions by considering the independence of intermediate pairs of firing steps and rule applications of the user actions in question.

6.3.1. Permutability of Firing Steps

In this section, we first give formal conditions for the permutability of two parallel concurring or consecutive AHLI firing steps. Note that the permutability is not as strict as the common notion of conflict-freeness in Petri nets that states that two firing steps may occur independently "in parallel" because there are enough tokens for both to consume. Instead, for permutability we consider also the tokens that are produced by each firing step to decide whether these firing steps can occur in any order with equivalent results.

Definition 6.3.1 (Parallel Permutability of Firing Steps)
We call two parallel concurring firing steps as shown in the top of Fig. 6.4

$$\begin{tikzcd}(NI, I_1, m_1) & (NI, I_0, m_0) \arrow[l, tail, "{t_1, asg_1, S_1}"'] \arrow[r, tail, "{t_2, asg_2, S_2'}"] & (NI, I_2, m_2)\end{tikzcd}$$

parallel permutable if the following two conditions hold:

$$(\mathrm{Pa})\colon pre_A(t_1, asg_1) \oplus pre_A(t_2, asg_2) \leq \sum_{i \in I_0} m_0(i) \oplus post_A(t_1, asg_1)$$

$$(\mathrm{Pb})\colon pre_A(t_1, asg_1) \oplus pre_A(t_2, asg_2) \leq \sum_{i \in I_0} m_0(i) \oplus post_A(t_2, asg_2) \qquad \triangle$$

$$\begin{tikzcd}
& (NI, I_0, m_0) \arrow[dl, tail, "t_1{,}asg_1{,}S_1"'] \arrow[dr, tail, "t_2{,}asg_2{,}S'_2"] & \\
(NI, I_1, m_1) \arrow[dr, tail, "t_2{,}asg_2{,}S_2"'] & & (NI, I_2, m_2) \arrow[dl, tail, "t_1{,}asg_1{,}S'_1"] \\
& (NI, I_3, m_3) \approx (NI, I'_3, m'_3) &
\end{tikzcd}$$

Figure 6.4.: Permutability of firing steps

Definition 6.3.2 (Sequential Permutability of Firing Steps)
We call two consecutive firing steps as shown in the left of Fig. 6.4

$$(NI, I_0, m_0) \overset{t_1,asg_1,S_1}{\rightarrowtail} (NI, I_1, m_1) \overset{t_2,asg_2,S_2}{\rightarrowtail} (NI, I_3, m_3)$$

sequentially permutable if the following two conditions hold:

$$(\mathrm{Sa})\colon pre_A(t_2, asg_2) \leq \sum_{i \in I_0} m_0(i)$$

$$(\mathrm{Sb})\colon pre_A(t_1, asg_1) \leq \sum_{i \in I_0} m_0(i) \ominus pre_A(t_2, asg_2) \oplus post_A(t_2, asg_2) \qquad \triangle$$

Next, we relate parallel and sequential permutability by a compatibility theorem similar to the local Church Rosser theorem in [EEPT06], stating that AHLI firing steps can be applied in any order under certain conditions, leading to equivalent results in the sense of Def. 4.4.3.

Theorem 6.3.3 (Compatibility of Parallel and Sequential Permutability)

1. *Given two parallel permutable firing steps as shown in the top of Fig. 6.4*

$$(NI, I_1, m_1) \overset{t_1,asg_1,S_1}{\leftarrowtail} (NI, I_0, m_0) \overset{t_2,asg_2,S'_2}{\rightarrowtail} (NI, I_2, m_2)$$

then there exist two firing steps $(NI, I_1, m_1) \overset{t_2,asg_2,S_2}{\rightarrowtail} (NI, I_3, m_3)$ *and* $(NI, I_2, m_2) \overset{t_1,asg_1,S'_1}{\rightarrowtail} (NI, I'_3, m'_3)$ *— called* complement *firing steps — such that both the firing sequences* $(NI, I_0, m_0) \overset{t_1,asg_1,S_1}{\rightarrowtail} (NI, I_1, m_1) \overset{t_2,asg_2,S_2}{\rightarrowtail} (NI, I_3, m_3)$ *and* $(NI, I_0, m_0) \overset{t_1,asg_1,S'_1}{\rightarrowtail} (NI, I_2, m_2) \overset{t_2,asg_2,S'_2}{\rightarrowtail} (NI, I'_3, m'_3)$ *are sequentially permutable and* $(NI, I_3, m_3) \approx (NI, I'_3, m'_3)$*.*

2. *Given two sequentially permutable firing steps as shown in the left of Fig. 6.4*

$$(NI, I_0, m_0) \overset{t_1,asg_1,S_1}{\rightarrowtail} (NI, I_1, m_1) \overset{t_2,asg_2,S_2}{\rightarrowtail} (NI, I_3, m_3)$$

then there exists a firing sequence $(NI, I_0, m_0) \overset{t_2,asg_2,S'_2}{\rightarrowtail} (NI, I_2, m_2) \overset{t_1,asg_1,S'_1}{\rightarrowtail} (NI, I'_3, m'_3)$ *— called* complement *firing steps — such that* $(NI, I_3, m_3) \approx (NI, I'_3, m'_3)$ *and that the firing steps* $(NI, I_1, m_1) \overset{t_1,asg_1,S_1}{\leftarrowtail} (NI, I_0, m_0) \overset{t_2,asg_2,S'_2}{\rightarrowtail} (NI, I_2, m_2)$ *are parallel permutable.*

Proof. See appendix A.3.6. □

Remark. The previous theorem states that permutable firing steps and their complement firing steps lead to equivalent AHLI nets. The equivalent nets and their markings are the same up to a bijection on the individual tokens, and firing steps and transformations of one net can be performed on all nets that are equivalent to the first one (by "appending" the token equivalence bijection in the matches and token selections) with results that are equivalent to the results of the first one. In the following, we assume that the token selections of the given firing steps are chosen such that they lead to the same firing result instead of equivalent results, because by suitable substitutions of the individual tokens in the selections, we can always find firings equivalent to the given ones for the consistent transition assignments (t_1, asg_1) and (t_2, asg_2) having the properties stated in Theorem 6.3.3 that lead to the same result .

To avoid a problem that may arise when considering the independency of user actions (see the remark on dependencies induced by token selections on page 162), we additionally define a stronger variant of parallel permutability.

Definition 6.3.4 (Strong Parallel Permutability of Firing Steps)
We call two parallel concurring firing steps as shown in the top of Fig. 6.4

$$(NI, I_1, m_1) \overset{t_1,asg_1,S_1}{\leftarrowtail} (NI, I_0, m_0) \overset{t_2,asg_2,S'_2}{\rightarrowtail} (NI, I_2, m_2)$$

strongly parallel permutable if they are parallel permutable and if (Pc): $N_{S_1} \cap N_{S_2} = \emptyset$ holds. △

Corollary 6.3.5 (Compatibility of Parallel and Sequential Permutability)
Both items of Theorem 6.3.3 also hold if we consider strongly parallel permutable firing steps instead of parallel permutable ones.

Proof. The proof of the first item is not affected by the additional condition (Pc). For the second item, we can always construct the set N'_2 such that $N_1 \cap N'_2$ holds and the complement firing step is strongly parallel independent. □

6.3.2. Independence of Firing Steps and Rule Applications

Due to Theorem 4.2.12, we immediately get the important results of the independence of firing and transformation steps. Thus, we obtain the notion of parallel independence of these steps for AHLI nets by relying on the definition of certain properties for the corresponding transition rule.

For P/T systems, [EHP+07] defines parallel and sequential independence of a transformation step and a firing step and proves results that are similar to the local Church-Rosser theorem of [EEPT06], which relates sequential and parallel independence of rule applications without regarding application conditions. In [MGH11], the local Church-Rosser theorem for the parallel independence of firing and transformation steps of PTI nets is introduced.

For expressing the ownership of actions in Communication Platform models, we use arcs from the corresponding transition to places with identifier tokens (see Sect. 5.1.1.1). But usually, these identifiers must not be changed by any firing step, which we formalize by integrity conditions in Sect. 5.1.1.1. Therefore, the identifier token with the owning user's name is consumed and recreated in every firing step. The concept of [MGH11] und [EHP+07], which characterizes the independence of a firing step and a transition by the independence of the application of the firing rule with the transformation, therefore is too strong for Communication Platform models, because every two actions performed of the same user are considered to be dependent by this.

The following definitions of parallel and sequential independence for firing steps and transformations in AHLI nets regards the tokens that are produced by the firing step so that the independence relation of the transformation to the firing step is more similar to the permutability of the previous section. This concept is more adequate for the consideration of user action independence in Communication Platform models. Note that the rules in these definitions do not have application conditions and that the matches of the complement transformation may violate the rule's application conditions. We treat this issue in the next section about user action independence.

Definition 6.3.6 (Parallel Independence of Transformations and Firing Steps) A transformation with almost injective match along inclusions and a parallel concurring and a firing step $\mathit{ANI_1} \overset{\varrho_1,o_1}{\Longleftarrow} \mathit{ANI_0} \overset{t,asg,S}{\rightarrowtail} \mathit{ANI_0}'$ are *parallel independent* if, considering the application of the firing rule $\varrho(t, asg, S) = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ at the inclusion match $o_t \colon L \to \mathit{ANI_0}$ (according to item 1 of Theorem 4.2.12) and the double pushout diagram of $\mathit{ANI_0} \overset{\varrho_1,o_1}{\Longrightarrow} \mathit{ANI_1}$ in Fig. 6.5,

- there exists an almost injective morphism $i_1 \colon L_1 \to \mathit{ANI_0}'$ such that $i_{1,P} = o_{1,P}$, $i_{1,T} = o_{1,T}$, and $i_{1,A} = o_{1,A}$, and
- there exists an almost injective morphism $i \colon L \to D_1$ such that $f_1 \circ i = o_t$. △

Definition 6.3.7 (Sequential Independence of Transformations and Firing Steps) A transformation with almost injective match along inclusions and a following firing step $\mathit{ANI_0} \overset{\varrho_1,o_1}{\Longrightarrow} \mathit{ANI_1} \overset{t,asg,S}{\rightarrowtail} \mathit{ANI_1}'$ are *sequentially independent* if, considering the

$$\begin{tikzcd}
R_1 \arrow[d, "o'_1"'] & K_1 \arrow[l, "r_1"'] \arrow[r, "l_1"] \arrow[d] & L_1 \arrow[dr, "o_1"] \arrow[drrr, dashed, "i_1"] & & L \arrow[dl, "o_t"'] \arrow[dlll, dashed, "i"'] & \\
ANI_1 & D_1 \arrow[l, "g_1"] \arrow[rr, "f_1"'] & & ANI_0 \arrow[rr, tail, "{t,asg,S}"] & & ANI'_0
\end{tikzcd}$$

Figure 6.5.: Conditions for parallel independence of a transformation and a firing step

application of the firing rule $\varrho(t,asg,S) = (L \xleftarrow{l} K \xrightarrow{r} R)$ with the inclusion match $o_t \colon L \to ANI_0$ (according to item 1 of Theorem 4.2.12) and the double pushout diagram of $ANI_0 \xRightarrow{\varrho_1,o_1} ANI_1$ in Fig. 6.6a,

- there exists an almost injective morphism $i_1 \colon R_1 \to ANI_1{}'$ such that $i_{1,P} = o'_{1,P}$, $i_{1,T} = o'_{1,T}$, and $i_{1,A} = o'_{1,A}$, and
- there exists an almost injective morphism $i \colon L \to D_1$ such that $g_1 \circ i = o_t$.

A firing step and a following transformation with almost injective match along inclusions $ANI_0 \overset{t,asg,S}{\rightarrowtail} ANI'_0 \xRightarrow{\varrho_1,o_1} ANI_1{}'$ are *sequentially independent* if, considering the application of the firing rule $\varrho(t,asg,S) = (L \xleftarrow{l} K \xrightarrow{r} R)$ with the inclusion comatch $o'_t \colon R \to ANI_0$ (according to item 1 of Theorem 4.2.12) and the double pushout diagram of $ANI'_0 \xRightarrow{\varrho_1,o_1} ANI'_1$ in Fig. 6.6b,

- there exists an almost injective morphism $i_1 \colon L_1 \to ANI_0$ such that $i_{1,P} = o_{1,P}$, $i_{1,T} = o_{1,T}$, and $i_{1,A} = o_{1,A}$, and
- there exists an almost injective morphism $i \colon R \to D_1$ such that $f_1 \circ i = o'_t$. △

$$\begin{tikzcd}
L_1 \arrow[d, "o_1"'] & K_1 \arrow[l, "l_1"'] \arrow[r, "r_1"] \arrow[d] & R_1 \arrow[dr, "o'_1"] \arrow[drrr, dashed, "i_1"] & & L \arrow[dl, "o_t"'] \arrow[dlll, dashed, "i"'] & \\
ANI_0 & D_1 \arrow[l, "f_1"] \arrow[rr, "g_1"'] & & ANI_1 \arrow[rr, tail, "{t,asg,S}"] & & ANI'_1
\end{tikzcd}$$

(a) Transformation followed by a firing step

$$\begin{tikzcd}
& R \arrow[dr, "o'_t"] \arrow[drrr, dashed, "i"] & & L_1 \arrow[dl, "o_1"'] \arrow[dlll, dashed, "i_1"'] & K_1 \arrow[l, "l_1"'] \arrow[r, "r_1"] \arrow[d] & R_1 \arrow[d, "o'_1"] \\
ANI_0 \arrow[rr, tail, "{t,asg,S}"] & & ANI'_0 & & D_1 \arrow[ll, "f_1"] \arrow[r, "g_1"'] & ANI'_1
\end{tikzcd}$$

(b) Firing step followed by a transformation

Figure 6.6.: Conditions for sequential independence

Next, the local Church-Rosser theorem from [EEPT06] is transferred to parallel and sequentially independent transformations and firing steps. As a result, under certain conditions transformations and firing steps can be applied in any order, leading to the same result with independent complement transformations and firing steps.

$$\begin{tikzcd}
& ANI_0 \arrow[dl, Rightarrow, "\varrho_1{,}o_1"'] \arrow[dr, tail, "t{,}asg{,}S"] & \\
ANI_1 \arrow[dr, tail, "t{,}asg{,}\overline{S}"'] & & ANI'_0 \arrow[dl, Rightarrow, "\varrho_1{,}\overline{o}_1"] \\
& ANI'_1 &
\end{tikzcd}$$

Figure 6.7.: Independency diagram for a transformation and a firing step

Theorem 6.3.8 (Local Church-Rosser Theorem)

1. *Given a direct transformation and firing step* $ANI_1 \xLeftarrow{\varrho_1,o_1} ANI_0 \overset{t,asg,S}{\rightarrowtail} ANI'_0$ *that are parallel independent (see the top of Fig. 6.7), then there exist the firing step* $ANI_1 \overset{t,asg,\overline{S}}{\rightarrowtail} ANI'_1$ *and a direct transformation* $ANI'_0 \xRightarrow{\varrho_1,\overline{o}_1} ANI'_1$ *— called* complement *firing step and transformation — such that* $ANI_0 \xRightarrow{\varrho_1,o_1} ANI_1 \overset{t,asg,\overline{S}}{\rightarrowtail} ANI'_1$ *and* $ANI_0 \overset{t,asg,S}{\rightarrowtail} ANI'_0 \xRightarrow{\varrho_1,\overline{o}_1} ANI'_1$ *both are sequentially independent.*

2a. *Given a direct transformation followed by a firing step* $ANI_0 \xRightarrow{\varrho_1,o_1} ANI_1 \overset{t,asg,\overline{S}}{\rightarrowtail} ANI'_1$ *that are sequentially independent (see the left of Fig. 6.7), then there exist the firing step* $ANI_0 \overset{t,asg,S}{\rightarrowtail} ANI'_0$ *and a direct transformation* $ANI'_0 \xRightarrow{\varrho_1,\overline{o}_1} ANI'_1$ *— called* complement *firing step and transformation — such that* $ANI_1 \xLeftarrow{\varrho_1,o_1} ANI_0 \overset{t,asg,S}{\rightarrowtail} ANI'_0$ *are parallel independent.*

2b. *Given a firing step followed by a direct transformation* $ANI_0 \overset{t,asg,S}{\rightarrowtail} ANI'_0 \xRightarrow{\varrho_1,\overline{o}_1} ANI'_1$ *that are sequentially independent (see the right of Fig. 6.7), then there exists a direct transformation* $ANI_0 \xRightarrow{\varrho_1,o_1} ANI_1$ *and a firing step* $ANI_1 \overset{t,asg,\overline{S}}{\rightarrowtail} ANI'_1$ *— called* complement *firing step and transformation — such that* $ANI_1 \xLeftarrow{\varrho_1,o_1} ANI_0 \overset{t,asg,S}{\rightarrowtail} ANI'_0$ *are parallel independent.*

Proof. See appendix A.3.7. □

6.3.3. Independence of User Actions in Platform Models

With the result from Theorem 6.3.8, we get information about the case of two given user actions as in Fig. 6.3 by investigating the dependencies between the user actions' components. As pointed out on page 148, the different extents to which the parts of concurrent of consecutive user actions can be independent represent grades of independence that are valuable information for validating whether user actions comply the modeling intentions. We start with the maximal grade of independence that all firing steps and all transformation steps of two user actions are mutually independent. Then, we discuss and define weaker grades of independence.

$$\begin{tikzcd}
ANI_0 \arrow[r, Rightarrow, "seq{,}o\,*"] \arrow[d, Rightarrow, "\hat{\varrho}{,}\hat{o}"'] & ANI_n \arrow[d, Rightarrow, "\hat{\varrho}{,}\hat{o}_n"'] \\
\widehat{ANI}_0 \arrow[r, Rightarrow, "seq{,}o\,*"] & \widehat{ANI}_n
\end{tikzcd}$$

(a) Direct transformation and concurring sequence

$$\begin{tikzcd}
ANI_0 \arrow[r, Rightarrow, "\varrho_1{,}o_1"] \arrow[d, Rightarrow, "\hat{\varrho}{,}\hat{o}"'] & ANI_1 \arrow[r, Rightarrow, "\varrho_2{,}o_2"] \arrow[d, Rightarrow, "\hat{\varrho}{,}\hat{o}_1"'] & \ldots \arrow[r, Rightarrow, "\varrho_n{,}o_n"] & ANI_n \arrow[d, Rightarrow, "\hat{\varrho}{,}\hat{o}_n"'] \\
\widehat{ANI}_0 \arrow[r, Rightarrow, "\varrho_1{,}o'_1"] & \widehat{ANI}_1 \arrow[r, Rightarrow, "\varrho_2{,}o'_2"] & \ldots \arrow[r, Rightarrow, "\varrho_n{,}o'_n"] & \widehat{ANI}_n
\end{tikzcd}$$

(b) Intermediate parallel independent transformations

Figure 6.8.: Parallel independence of a direct transformation and a transformation sequence

Independence of transformation sequences For the definitions of independence of user actions, we need to extend the notions of independence of rule applications to stepwise independence of transformations sequences, which we describe informally. Consider a single rule application $(\hat{\varrho}, \hat{o})$ and a transformation sequence (seq, o) as shown in Fig. 6.8a, where the sequence (seq, o) consists of direct transformations (ϱ_i, o_i) as shown in Fig. 6.8b. If $(\hat{\varrho}, \hat{o})$ and (ϱ_i, o_i) are parallel independent, we get by the local Church-Rosser theorem that there exist direct transformations $(\hat{\varrho}, \hat{o}_1)$ and (ϱ_1, o'_1) such that the sequences $ANI_0 \overset{\varrho_1,o_1}{\Longrightarrow} ANI_1 \overset{\hat{\varrho},\hat{o}}{\Longrightarrow} \widehat{ANI}_1$ and $ANI_0 \overset{\hat{\varrho},\hat{o}}{\Longrightarrow} \widehat{ANI}_0 \overset{\varrho_1,o'_1}{\Longrightarrow} \widehat{ANI}_1$ are sequentially independent. If every pair of transformations $(\hat{\varrho}, \hat{o}_i)$ and (ϱ_{i+1}, o_{i+1}) is parallel independent leading to a chain of independency squares such as in Fig. 6.8b, it is natural to call $(\hat{\varrho}, \hat{o})$ and (seq_1, o_1) parallel independent. This construction can be adapted easily to stepwise sequential independence of a rule application and a transformation sequence, then to rule applications mixed with firing steps, and finally to independent sequences of transformations and rule applications so that we get a grid of independency squares rather than a chain.

We can always assume that a request handling transformation sequence depends on its request-triggering firing step. Thus, it is pointless to consider dependencies between parts of the same user action, e. g., whether (t_1, asg_1, S_1) and (seq_1, o_1) in Fig. 6.9 are sequentially independent. The complement (independent) user actions are constructed from the given ones, i. e., the co-span of user actions in the bottom of Fig. 6.3 from the span in the top (for parallel independent actions) or the right sequence from the left sequence (for sequentially independent actions), such that suitable properties of parallel and sequential independence for user actions are related as in the common Church-Rosser theorems such as Theorem 6.3.8. By assuming functional user actions (cf. Sect. 6.2), we eliminate the nondeterminism of possible transformation sequences such that the complexity is reduced considerably and we achieve a unique result for the complement user actions.

For the definition of parallel and sequential independence of user actions, we consider two functional user actions and give conditions that allow us to construct the complement functional user actions. The conditions have to be processed stepwise, because every condition yields an intermediate independency square with the construction of the complement firing steps and transformations (see Theorem 6.3.8), which is used by the consequent conditions.

Definition 6.3.9 (Parallel Independence of User Actions)
Given two functional user actions $ANI_0 \overset{t_1,asg_1,S_1}{\rightarrowtail} ANI'_0 \overset{seq_1,o_1}{\Longrightarrow} ANI'_1$ and $ANI_0 \overset{t_2,asg_2,S_2}{\rightarrowtail} ANI''_0 \overset{seq_2,o_2}{\Longrightarrow} ANI''_2$, we call these actions *parallel independent* if (as depicted in Fig. 6.9) consecutively

(P1): the firing steps (t_1, asg_1, S_1) and (t_2, asg_2, S_2) in ANI_0 are strongly parallel permutable according to Def. 6.3.1, i. e., by Theorem 6.3.3 (and the remark following it), we get complement firing steps leading to a net ANI'''_0 in square (1),

(P2): the firing step $ANI'_0 \overset{t_2,asg_2,\overline{S}'_2}{\rightarrowtail} ANI'''_0$ is elementwise parallel independent of the transformation sequence along (seq_1, o_1), leading to square (2),

(P3): the firing step $ANI''_0 \overset{t_1,asg_1,\overline{S}'_1}{\rightarrowtail} ANI'''_0$ is elementwise parallel independent of the transformation sequence along (seq_2, o_2), leading to square (3),

(P4): the transformation sequences along (seq_1, o'_1) and (seq_2, o'_2) are elementwise parallel independent, leading to square (4) such that (seq_2, o''_2) is functional on ANI'''_1 and (seq_1, o''_1) is functional on ANI'''_2. △

$$\begin{tikzcd}
 & & ANI_0 \arrow[dl, tail, "t_1{,}asg_1{,}S_1"'] \arrow[dr, tail, "t_2{,}asg_2{,}S_2"] & & \\
 & ANI'_0 \arrow[dl, Rightarrow, "seq_1{,}o_1"', "*" near end] \arrow[dr, tail, "t_2{,}asg_2{,}\overline{S}'_2"] & (1) & ANI''_0 \arrow[dl, tail, "t_1{,}asg_1{,}\overline{S}'_1"'] \arrow[dr, Rightarrow, "seq_2{,}o_2", "*"' near end] & \\
ANI'_1 \arrow[dr, tail, "t_2{,}asg_2{,}S'_2"'] & (2) & ANI'''_0 \arrow[dl, Rightarrow, "seq_1{,}o'_1"', "*" near end] \arrow[dr, Rightarrow, "seq_2{,}o'_2", "*"' near end] & (3) & ANI''_2 \arrow[dl, tail, "t_1{,}asg_1{,}S'_1"] \\
 & ANI'''_1 \arrow[dr, Rightarrow, "seq_2{,}o''_2"', "*" near end] & (4) & ANI'''_2 \arrow[dl, Rightarrow, "seq_1{,}o''_1", "*"' near end] & \\
 & & ANI'''_3 & &
\end{tikzcd}$$

Figure 6.9.: Intermediate steps for independent user actions

Definition 6.3.10 (Sequential Independence of User Actions)
Given two functional user actions $ANI_0 \overset{t_1,asg_1,S_1}{\rightarrowtail} ANI'_0 \overset{seq_1,o_1}{\Longrightarrow} ANI'_1$ and $ANI'_1 \overset{t_2,asg_2,S'_2}{\rightarrowtail} ANI'''_1 \overset{seq_2,o''_2}{\Longrightarrow} ANI'''_3$, we call these actions *sequentially independent* if (as depicted in Fig. 6.9) consecutively

(S1): the firing step $ANI'_1 \overset{t_2,asg_2,S'_2}{\rightarrowtail} ANI'''_1$ is sequentially independent of the complete transformation sequence along (seq_1, o_1) leading to square (2), (S2): the firing steps $ANI_0 \xrightarrow{t_1,asg_1,\overline{S}_1} ANI'_0 \xrightarrow{t_2,asg_2,S'_2} ANI'''_0$ are sequentially permutable according to Def. 6.3.2, i. e., by Theorem 6.3.3 (and remark following it), we get complement firing steps leading to square (1),

(S3): there exists a transformation sequence (seq_2, o_2) that is functional on ANI''_0 and elementwise parallel independent of the firing step $ANI''_0 \xrightarrow{t_1,asg_1,\overline{S}'_1} ANI'''_0$, leading to square (3),

(S4): the transformation sequences along (seq_1, o'_1) and (seq_2, o'_2) are elementwise parallel independent, leading to square (4) by construction of the complement actions such that (seq_1, o''_1) is functional on ANI'''_2. △

Remark (Well-definedness of independency diagrams for user actions). By definition, transformations in user actions are along inclusions. Therefore, the existence of the complement firing steps and transformation sequences for the squares in Def. 6.3.9 and Def. 6.3.10 follows from applying Theorem 6.3.8.

The functionality of transformation sequences in the squares does not follow from the square construction and is rather a required condition for independence to be shown in each case!

Remark (Application conditions and functionality of intermediate constructions). Note that if not stated explicitly as in (P4), (S2), and (S3), we do not require the complement transformations to be functional. Moreover, Theorem 6.3.8 does not regard application conditions and may construct complement transformation sequences in these squares where the match sequences possibly are not valid according to Def. 5.2.18 because they violate the application conditions. Actually, requiring functionality and satisfaction of application conditions for the intermediate constructions would result in a too strong notion of independence that classifies far more user actions to be dependent than necessary for Communication Platform models.

The justification for this simplification is that we assume user actions — i. e., the triggering and request handling — to be performed atomically in a Communication Platform model so that consequently only the two outermost paths in Fig. 6.9 — i. e., the ones corresponding to the user actions in Fig. 6.3 — are considered as proper user actions in the sense of Def. 5.3.1 at all. Clearly, the net ANI'''_0 in the center of Fig. 6.9 can only result from a firing of t_2 that "interrupts" the user action triggered by t_1 (or vice versa). This means that any constructed path other than the two outermost ones cannot occur by performing user actions in the platform net ANI_0, and we may need them just as formal constructions to construct the (parallel or sequentially) complement user action, as long as this leads to correct and independent complement user actions.

Note also that for verifying (S3) and (S4), there are equivalent conditions. We can alternatively require that

(S3'): the transformation sequences along (seq_1, o'_1) and (seq_2, o''_2) are elementwise sequentially independent, leading to square (4) such that (seq_1, o''_1) is functional on ANI'''_2, (S4'): the firing step $ANI''_0 \xrightarrow{t_1,asg_1,\overline{S}'_1} ANI'''_0$ is sequentially independent of the complete transformation sequence along (seq_2, o'_2), leading to square (3) such that (seq_2, o_2) is functional on ANI''_0.

But regarding the definitions of weak and trigger independence and the following compatibility theorem, the actual definition of sequential independence allows us to express them more concisely.

Weak and trigger independence The conditions (P4) and (S4) demand that the (intermediate) parallel independent transformation sequences (seq_1, o'_1) and (seq_2, o'_2), which are not necessarily functional on ANI'''_0, lead to the construction of transformation sequences that have to be functional on the corresponding nets. This is a rather strict requirement, which expresses the highest possible degree of independence between two user actions in the sense that performing one user action does not interfere with the functional matching for the request handling of the other one. We came up with handler expressions and valid matching sequences for the triggered reconfigurations, because there are many modeling cases where the request handlers have to be designed to catch exceptions for preventing errors and defective configurations in the platform model. This means that two functional user actions actually may lead to the same configuration ANI'''_3 without that the handlings in the lower square (4) result from a complement construction of the intermediate independent transformation sequences by the local Church-Rosser theorem. In other cases, there may not even exist functional handlings forming a confluent square (4).

To distinguish these degrees of user action independence, we additionally define weak and trigger independence, which loosen these restrictions.

Definition 6.3.11 (Weak Independence of User Actions)
We call two functional user actions *parallel weakly independent* if they comply to the properties given in Def. 6.3.9, except for condition (P4), which is replaced by

(wP4): there are transformation sequences $ANI'''_2 \xRightarrow{seq_1,o''_1} ANI'''_3$ functional on ANI'''_2 and $ANI'''_1 \xRightarrow{seq_2,o''_2} ANI'''_3$ functional on ANI'''_1 resulting in a confluent square (4).

We call two functional user actions *sequentially weakly independent* if they comply to the properties given in Def. 6.3.10, except for condition (S4), which is replaced by

(wS4): there is a transformation sequence $ANI'''_2 \xRightarrow{seq_1,o''_1} ANI'''_3$ functional on ANI'''_2 resulting in a confluent square (4). △

Definition 6.3.12 (Trigger Independence of User Actions)
We call two functional user actions *parallel trigger-independent* if they comply (at least) to the properties (P1)-(P3) given in Def. 6.3.9.

Analogously, two functional user actions are *sequentially trigger-independent* if they comply (at least) to the properties (S1)-(S3) given in Def. 6.3.10.

If any of these properties does not hold for two functional user actions, we call them *parallel* or *sequentially trigger-dependent*, respectively. △

The following theorem states that the notions of parallel and sequential user action independence of a certain independence degree are compatible to each other, analogously to the relation of parallel and sequential independent transformations and firing steps in the local Church-Rosser Theorem 6.3.8.

Theorem 6.3.13 (Compatibility of Parallel and Sequential Independence) *For a given Communication Platform model, the following holds:*

1. *Given two functional and parallel (weakly) independent user actions* $ua_1 = ANI_0 \overset{t_1,asg_1,S_1}{\rightarrowtail} ANI'_0 \xRightarrow{seq_1,o_1} ANI'_1$ *and* $ua_2 = ANI_0 \overset{t_2,asg_2,S_2}{\rightarrowtail} ANI''_0 \xRightarrow{seq_2,o_2} ANI''_2$*, then there are an AHLI net* ANI'''_3 *and functional user actions* $ua'_2 = ANI'_1 \overset{t_2,asg_2,S'_2}{\rightarrowtail} ANI'''_1 \xRightarrow{seq_2,o''_2} ANI'''_3$ *and* $ua'_1 = ANI'_2 \overset{t_1,asg_1,S'_1}{\rightarrowtail} ANI'''_1 \xRightarrow{seq_1,o''_1} ANI'''_3$ *— called* complement actions *— such that* ua_1 *and* ua'_2 *as well as* ua_2 *and* ua'_1 *are sequentially (weakly) independent.*
2. *Given two functional and sequentially (weakly) independent user actions* $ua_1 = ANI_0 \overset{t_1,asg_1,S_1}{\rightarrowtail} ANI'_0 \xRightarrow{seq_1,o_1} ANI'_1$ *and* $ua_2 = ANI'_1 \overset{t_2,asg_2,S'_2}{\rightarrowtail} ANI'''_1 \xRightarrow{seq_2,o''_2} ANI'''_3$*, then there are an AHLI net* ANI''_2 *and two functional user actions* $ua'_2 = ANI_0 \overset{t_2,asg_2,S_2}{\rightarrowtail} ANI''_0 \xRightarrow{seq_2,o_2} ANI''_2$ *and* $ua'_1 = ANI'_2 \overset{t_1,asg_1,S'_1}{\rightarrowtail} ANI'''_1 \xRightarrow{seq_1,o''_1} ANI'''_3$ *— called* complement actions *— such that* ua_1 *and* ua'_2 *are parallel (weakly) independent.*

Proof.

1. We first prove the case for general independence. The construction along (P1)-(P4) of Def. 6.3.9 yields the functional user actions ua'_1 and ua'_2 as in Fig. 6.9. It remains to show that ua_1 and ua'_2 as well as ua_2 and ua'_1 are sequentially independent. We prove the conditions for sequential independence of ua_1 and ua'_2.

   (S1): From (P2), we have that the sequence and the constructed firing step $ANI'_1 \xLeftarrow{seq_1,o_1} ANI'_0 \overset{t_2,asg_2,\overline{S}'_2}{\rightarrowtail} ANI'''_0$ are (elementwise) parallel independent. According to Theorem 6.3.8, there exists the sequentially independent firing step $ANI'_1 \overset{t_2,asg_2,S'_2}{\rightarrowtail} ANI'''_1$ in square (2).

   (S2): This follows from (P1), because by Cor. 6.3.5, the strong parallel permutability of $ANI_0 \overset{t_1,asg_1,S_1}{\rightarrowtail} ANI'_0$ and $ANI_0 \overset{t_2,asg_2,S_2}{\rightarrowtail} ANI''_0$ implies the sequentially permutable firing step $ANI'_0 \overset{t_2,asg_2,\overline{S}'_2}{\rightarrowtail} ANI'''_0$ in square (1).

   (S3): This follows directly from (P3) and that ua_2 is a functional user action.

   (S4): This follows directly from (P4).

   For the case of parallel weakly independent user actions ua_1 and ua_2, (wS4) follows directly from (wP4). The conditions for the sequential (weak/trigger) independence of ua_2 and ua'_1 can be shown analogously.
2. The construction along (S1)-(S4) of Def. 6.3.10 yields the functional user actions ua'_2 and ua'_1 as in Fig. 6.9. It remains to show that ua_1 and ua'_2 are parallel independent.

(P1): This follows from (S2), because by Cor. 6.3.5, the sequential permutability of the firing steps $ANI_0 \overset{t_1,asg_1,S_1}{\rightarrowtail} ANI'_0 \overset{t_2,asg_2,\overline{S}'_2}{\rightarrowtail} ANI'''_0$ implies the strongly parallel permutable firing step $ANI_0 \overset{t_2,asg_2,S_2}{\rightarrowtail} ANI''_0$ in square (1).

(P2): From (S1), we have that the sequence and the constructed firing step $ANI'_0 \overset{seq_1,o_1}{\Longrightarrow} ANI'_1 \overset{t_2,asg_2,S'_2}{\rightarrowtail} ANI'''_1$ are (elementwise) sequentially independent. According to Theorem 6.3.8, there exists the parallel independent firing step $ANI'_0 \overset{t_2,asg_2,\overline{S}'_2}{\rightarrowtail} ANI'''_0$ in square (2).

(P3): This follows from (S3), because we assume seq_2, o_2 in (S3) to be functional so that there is no other valid match sequence for seq_2 on ANI''_0.

(P4): This follows directly from (S4) and that ua'_2 is a user action.

For the case of sequentially weakly independent user actions ua_1 and ua_2, (wP4) follows directly from (wS4) and that ua'_2 is a functional user action. □

For trigger-independent actions, we formulate a separate corollary, which follows directly by the previous proof.

Corollary 6.3.14 (Compatibility of Parallel and Sequential Independence) *For a given Communication Platform model, the following holds:*

1. *Given two functional and parallel trigger-independent user actions $ua_1 = ANI_0 \overset{t_1,asg_1,S_1}{\rightarrowtail} ANI'_0 \overset{seq_1,o_1}{\Longrightarrow} ANI'_1$ and $ua_2 = ANI_0 \overset{t_2,asg_2,S_2}{\rightarrowtail} ANI''_0 \overset{seq_2,o_2}{\Longrightarrow} ANI''_2$, then there are user actions $ua'_2 = ANI'_1 \overset{t_2,asg_2,S'_2}{\rightarrowtail} ANI'''_1 \overset{seq_2,o''_2}{\Longrightarrow} ANI'''_{32}$ and $ua'_1 = ANI''_2 \overset{t_1,asg_1,S'_1}{\rightarrowtail} ANI'''_1 \overset{seq_1,o''_1}{\Longrightarrow} ANI'''_{31}$ — called* complement actions *— such that ua_1 and ua'_2 as well as ua_2 and ua'_1 are sequentially trigger-independent.*
2. *Given two functional and sequentially trigger-independent user actions $ua_1 = ANI_0 \overset{t_1,asg_1,S_1}{\rightarrowtail} ANI'_0 \overset{seq_1,o_1}{\Longrightarrow} ANI'_1$ and $ua_2 = ANI'_1 \overset{t_2,asg_2,S'_2}{\rightarrowtail} ANI'''_1 \overset{seq_2,o'_2}{\Longrightarrow} ANI'''_3$, then there exists a user action $ua'_2 = ANI_0 \overset{t_2,asg_2,S_2}{\rightarrowtail} ANI''_0 \overset{seq_2,o_2}{\Longrightarrow} ANI''_2$ such that ua_1 and ua'_2 are parallel trigger-independent.*

Proof. For item 1, where ua_1 and ua_2 are parallel trigger-independent, the arguments for (S1)-(S3) from the proof of Theorem 6.3.13 still hold because they only rely on (P1)-(P3). Note that in this case $ANI'''_1 \overset{seq_2,o''_2}{\Longrightarrow} ANI'''_{32}$ of the user action ua'_2 and $ANI'''_2 \overset{seq_1,o''_1}{\Longrightarrow} ANI'''_{31}$ of the user action ua'_1 may not be functional and just any transformation sequence, possibly even the empty sequence. For item 2, the argument is analogous that (P1)-(P3) can be shown by assuming (S1)-(S3). □

Remark (User actions without reconfigurations). For a pair of user actions of which one is a plain (nontriggering) firing step without a subsequent request handling, the definitions of independence for user actions can be reduced to the first two conditions. For two parallel user actions, we require (P1)-(P2) to be satisfied for parallel trigger independence. If there also exists any functional transformation sequence $ANI_0''' \xRightarrow{seq_1,o_1'} ANI_1'''$ leading to a commutative square (not necessarily the one resulting from (P2)), we call the user actions parallel weakly independent. Parallel independence of the user actions is given if moreover $ANI_0''' \xRightarrow{seq_1,o_1'} ANI_1'''$ is the very transformation resulting from the parallel independence construction of (P2).

For two consecutive user actions of which the second is the nonreconfiguring one, we require (S1) and (S2) for sequential trigger independence. If moreover there exists any functional transformation sequence $ANI_0''' \xRightarrow{seq_1,o_1'} ANI_1'''$ leading to a commutative square (2) (not necessarily the one resulting from the independency square of (S1)), we call these user actions sequentially weakly independent. It the transformation sequence $ANI_0''' \xRightarrow{seq_1,o_1'} ANI_1'''$ itself is the complement transformation resulting from the sequential independency square of (S1), the given user actions are sequentially independent.

For two sequential user actions of which the first is the nonreconfiguring one, we require (S2) and (S3) for sequential trigger independence, but without the condition that the functional transformation sequence $(seq_2, o2)$ has to lead to a commutative square (3) with $ANI_0''' \xRightarrow{seq_1,o_1'} ANI_1'''$ of the second given user action. If $(seq_2, o2)$ indeed leads to the commutative square (3), we call these user actions sequentially weakly independent. If moreover the transformation sequence $ANI_0''' \xRightarrow{seq_1,o_1'} ANI_1'''$ itself is the complement transformation sequence resulting from the parallel independency square of (S3), the given user actions are sequentially independent.

For these cases of nonreconfiguring user actions, analogous results to the ones in Theorem 6.3.13 can easily be shown. For two user actions without reconfigurations, the degrees of independence coincide with the permutability of the corresponding firing steps, i. e., the conditions (P1) and (S2).

(Weak/trigger) dependence of user actions By Theorem 6.3.13, parallel independence means that two concurring user actions can be performed in any order with the same resulting configuration and the same request-handling transformation sequence. The same holds for parallel weakly independent user actions, although their handling transformation sequences are not necessarily independent in the strict sense of transformation independence. In graph transformations, two rule applications are called dependent if they are not independent. According to this, we call two concurring user actions *parallel dependent* if they are not parallel weakly independent, i. e., we cannot construct sequentially weakly independent user actions as in item 1 of Theorem 6.3.13 that lead to the same result. A weaker form of dependency is given for two actions that are just parallel weakly independent but not parallel independent (which may be called "properly parallel weakly independent"). Therefore, we denote with *parallel weak dependence* the negation of "parallel independence".

For the result on trigger independence in Cor. 6.3.14, we consider solely whether the triggering firing step can occur and do not regard the complement user actions' transformation sequences. We denote with *trigger dependence* simply the counterpart of trigger independence to express that the triggering firing step of an action depends on another concurring or preceding action.

parallel	weak dep.	$\Longleftarrow$	dep.	$\Longleftarrow$	trigger dep.
	$\not\Updownarrow$		$\not\Updownarrow$		$\not\Updownarrow$
	indep.	$\Longrightarrow$	weak indep.	$\Longrightarrow$	trigger indep.
	(P1)-(P4)		(P1)-(P3),(wP4)		(P1)-(P3)
by Theorem 6.3.13	$\leftrightsquigarrow$		$\leftrightsquigarrow$		$\leftrightsquigarrow$
sequential	indep.	$\Longrightarrow$	weak indep.	$\Longrightarrow$	trigger indep.
	(S1)-(S4)		(S1)-(S3),(wS4)		(S1)-(S3)
	$\not\Updownarrow$		$\not\Updownarrow$		$\not\Updownarrow$
	weak dep.	$\Longleftarrow$	dep.	$\Longleftarrow$	trigger dep.

Table 6.1.: Overview: independence of user actions

In Table 6.1, we summarize the different degrees of independence and dependence and their relations. For every independence degree, the table enumerates the required conditions. Note that for every independence degree, the corresponding dependence degree (indicated by the antivalence $\not\Leftrightarrow$) requires just any of these conditions to be violated. A higher degree of independence implies a lower degree (indicated by $\Rightarrow$). The parallel and sequential independence degrees are related conceptually ($\leftrightsquigarrow$) by Theorem 6.3.13 (and Cor. 6.3.14).

$$\begin{tikzcd}
 & & ANI_0 \arrow[dl, tail, "t_1{,}asg_1{,}S_1"'] \arrow[dr, tail, "t_1{,}asg_1{,}S_1"] & & \\
 & ANI'_0 \arrow[dl, Rightarrow, "\varrho_1{,}o_1"'] \arrow[dr, tail, "t_1{,}asg_1{,}\overline{S}'_1"'] & \textbf{(1)} & ANI'_0 \arrow[dl, tail, "t_1{,}asg_1{,}\overline{S}'_1"] \arrow[dr, Rightarrow, "\varrho_1{,}o_1"] & \\
ANI_1 \arrow[dr, tail, "t_1{,}asg_1{,}S'_1"'] & \textbf{(2)} & ANI''_0 \arrow[dl, Rightarrow, "\varrho_1{,}o'_1"'] \arrow[dr, Rightarrow, "\varrho_1{,}o'_1"] & \textbf{(3)} & ANI_1 \arrow[dl, tail, "t_1{,}asg_1{,}S'_1"] \\
 & ANI'_1 \arrow[dr, Rightarrow, "\varrho_1{,}o''_1"'] & \textbf{(4)} & ANI'_1 \arrow[dl, Rightarrow, "\varrho_1{,}o''_1"] & \\
 & & ANI_1 & &
\end{tikzcd}$$

Figure 6.10.: Weak dependency induced by token selection in triggering firing steps

Remark (Dependencies induced by token selection). With the results of this section at hand, we justify why the triggering firing steps of user actions have to be strongly parallel permutable rather than just parallel permutable. Consider the example of a user action with minimal effect, where the triggering firing step $ANI_0 \overset{t_1,asg_1,S_1}{\rightarrowtail} ANI'_0$

does not consume any token and produces a request token i_1 in ANI'_0. The algebraic value and place of this request token is not important for this example. The functional handling $ANI'_0 \overset{\varrho_1,o_1}{\Longrightarrow} ANI_1$ simply deletes this request token. If we check this user action for parallel independence to itself (see the top of Fig. 6.10), intuitively, we expect that the action is parallel independent to itself, because it just creates a request and deletes it without any other precondition or side effects.

We can construct (1), because the firing step is parallel permutable and get the complement firing step $ANI'_0 \overset{t_1,asg_1,\overline{S}_1}{\rightarrowtail} ANI''_0$ that adds a token $\bar{i}_1$ in ANI''_0. Because $ANI'_0 \overset{\varrho_1,o_1}{\Longrightarrow} ANI_1$ consumes the token i_1 from ANI'_0 also the complement transformation $ANI''_0 \overset{\varrho_1,o'_1}{\Longrightarrow} ANI'_1$ matches and deletes the token i_1 from ANI''_0. Unfortunately, it turns out that the transformation $ANI''_0 \overset{\varrho_1,o'_1}{\Longrightarrow} ANI'_1$ is not parallel independent to itself. Therefore, we get only weak independence (and weak dependence) by the transformation $ANI'_1 \overset{\varrho_1,o''_1}{\Longrightarrow} ANI'_1$ that consumes the token $\bar{i}_1$, leading to the confluent square (4).

But, if we consider the parallel concurring user actions

$$ANI_1 \overset{\varrho_1,o_1}{\Longleftarrow} ANI'_0 \overset{t_1,asg_1,S_1}{\leftarrowtail} ANI_0 \overset{t_1,asg_1,S^*_1}{\rightarrowtail} ANI^{*\prime}_0 \overset{\varrho_1,o^*_1}{\Longrightarrow} ANI^*_1$$

where the selection S^*_1 creates a token i^*_1 that is different than i_1 created by the selection S, we get parallel independence for the user actions without weak dependency: $ANI^{*\prime}_0 \overset{\varrho_1,o^*_1}{\Longrightarrow} ANI^*_1$ consumes the token i^*_1 and therefore the span of complement transformations in the top of square (4) do not delete the same token as in the example in Fig. 6.10 so that they are parallel independent.

Evidently, the weak dependency in the first example is induced by the overlapping of the created token i_1 in the parallel concurring trigger firing step. This causes a dependency of the following functional handlings on the request tokens that leads to a conflict in ANI''_0. The requirement for strong parallel permutability of the firing steps avoids such dependencies so that the tokens involved in conflicts of the complement transformations in square (4) already are present in the first net ANI_0. The extension of Theorem 6.3.3 to Cor. 6.3.5 further justifies to consider strong parallel permutability.

In the following, if we consider the parallel independence of two user actions, we assume that the selections of their triggering firing steps (and the following nets and transformations) have been changed to meet the condition (Pc) for strong parallel permutability in Def. 6.3.4 (in addition to selection changes according to the remark on page 151). This is passable, because the intermediate nets of the user actions just have to be replaced by nets that behave equivalent regarding transformations and firing behavior (cf. Theorem 4.4.11).

We demonstrate and apply the concepts and degrees of independence of user actions on example activities of the Skype case study in Sect. 7.4.3.

7. Modeling Case Study: Skype

> Take, for example, the question of obedience. In the past, when I gave her [my dog] a command, I had absolutely no way of knowing how she would respond. Her responses seemed to me so varied that I could find no general law which would govern them all. Now I have such a law. But remember, I could never have found out this law without mathematical analysis. What is this law? I will tell you. Every time I have ever given her a command, she has *always* responded in exactly the same way, only I was not bright enough to recognize what the way is. Unaided by mathematics, I kept looking at the *differences* of the responses and was totally blind to the similarities. But now I know! Whenever I give her a command, there is only one thing she ever does, and every time it is the same thing. Either she obeys it or she doesn't.
>
> *(Raymond Smullyan)*

With the principles at hand on modeling Communication Platforms with reconfigurable Petri nets in Sect. 3.2 and the formal inventory of AHLI net transformations and Communication Platform models from the previous chapters, we are ready to apply these on a case study on modeling Skype. For this, we first examine the behavior of the Skype clients in more detail and with the modeling concepts in mind for delimiting the extent of the case study. In addition to defining a Communication Platform model for Skype, we also demonstrate how to apply the modeling concepts from Sect. 5.1. After that, we demonstrate the whole system specification at concrete use case scenarios, for which we also formulate and validate model properties.

This case study builds on and extends the case studies done in [BEE+09, Mod10, MEE+10, HM10].

7.1. Modeling Requirements for Skype as a Communication Platform Model

In this section, we examine the behavior of the Skype clients in more detail than in Sect. 3.1 to get an overview of the particular cases when we need reconfiguration by request handlers to realize Skype's features in a Communication Platform model of Skype.

As a first step, we collect all these cases of possible changes by observing the client's behavior and how the set of possible user actions changes, depending on the performed actions. In the discussion in Sect. 3.2.4, we worked out that particular communication and its related data are associated to a channel, i. e., a net structure connecting the client components in some way. In the following, to get an idea about how to manage such channels and the actions they provide, we treat the channels independently from each other by focusing on a single isolated channel and the user clients it connects. We take the point of view on a Skype AHLI net with components representing the user clients, keep track of how the sets of available transitions for a user change, and discuss which transitions are supposed to trigger a reconfiguration and for what reason. For this first survey, we assume all actions to be successful and ignore exceptions where these actions may fail such as calling a user that is offline and treat this issues later in detail in the next sections. These observations serve later on in this case study as an informal requirements specification of what the request handlers and rules of the Communication Platform model are supposed to do.

7.1.1. Client core actions

There are some actions that can be performed in the Skype client immediately after the first start or that are so universal in their purpose that we can consider their corresponding transitions to form the core component of a client without the need to ever reconfigure them with any transformations. This set of actions is our starting point for the reconfigurations and we call it the *client core*.

The first kind of actions is about setting the client's state, i. e., to *activate* and to *deactivate* the client. We ignore the other states in Skype such as "away", "do not disturb", and "invisible" because besides suppressing the ring tone or showing the user as offline in other clients, in fact they do not affect other actions, neither of the user's client nor of his contacts.[1]

A user can always *delete* and *undelete* the entries in his contact list, *change his privacy* settings, and *request a contact* from Skype's user registry to be added onto his contact list. We see later that *delete* and *undelete* need to trigger reconfigurations.

7.1.2. Direct channels

In Skype, one can open a communication window such as in Fig. 3.4 for contacts in the contact list as well as for contacts out of the Skype user registry. In this communication window, one can start a call or send messages. As a first step, we consider the case that two users have not been in contact before and that one of them opens such a window out of the user search tool. We assume that this action constitutes a channel between the two clients so that they can start communicating and that this communication is logged in the history of this window. In the following, we call this situation of two connected clients without mutual entries in their contact lists a "direct channel" between the clients.

For an overview, this is illustrated in the top left corner of Fig. 7.1 with the rounded boxes as configurations of the channel between two users and the arrows as reconfigurations that are triggered by the action that matches the arc inscription. In terms of user actions as in Def. 5.3.1, the triggering actions are transition firing steps with an optional change of configuration by applying the corresponding token request handler. For this

[1] It is indeed possible to try to call a contact in Skype that is displayed to be offline; he may just have set his status to "invisible" a be able to answer the call.

case of creating a direct channel, we have the configuration “Initial” and a triggering action *requestDirect*, which leads to the configuration “DirectChannel” next to it. In “DirectChannel”, the (new) actions above the division line belong to the user who triggers the reconfiguration with his user action, and the ones below belong to the passive user at the other end of the channel. Coincidentally, in this case the action sets of the active and the passive user are the same (up to *requestContact*), but we find differences in the next configuration.

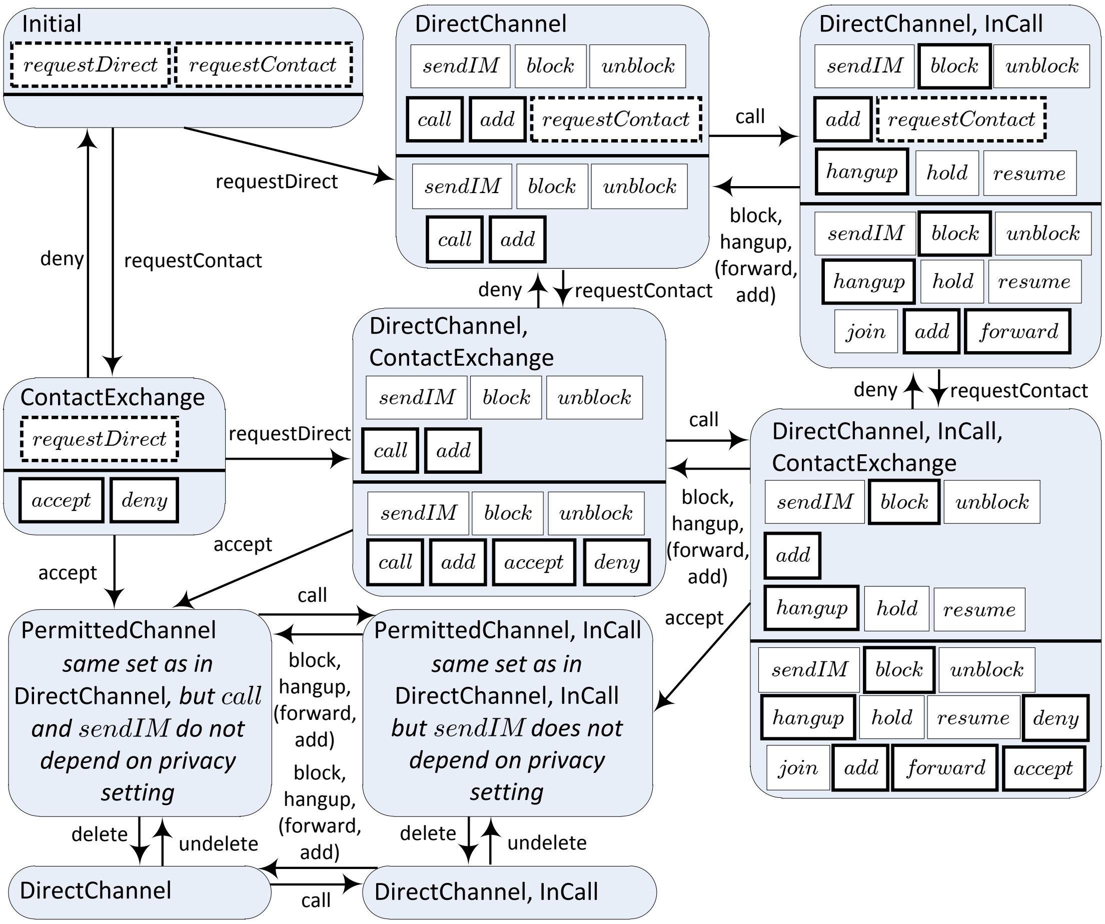


Figure 7.1.: Overview of actions and their effects in direct channels

We mark actions that trigger a reconfiguration with a bold border as for *requestDirect*. This action has to be available for the initiating user so that he can trigger reconfigurations for connecting his client to any other client component in the Skype net. But obviously, this action does not belong to the context of the channel itself that we are looking at. This is a strong indicator that *requestDirect* should belong to the client core and should not be affected by the reconfiguration, which we denote by drawing a dashed border for the action (in this case additionally to the bold border). For now, we ignore special cases such as that the direct channel already exists and how to handle such exception and assume that the reconfiguration by *requestDirect* can be invoked once for each pair of users.

In a direct channel, each client can *send instant messages*, initiate a *call*, and *block* or *unblock* the other user. Sending messages does not need reconfiguration because the sent message can simply be made available for the other user by logging it in his history for this channel.

Although blocking and unblocking of the other user affects the possibility to call him and send messages, these actions do not need reconfiguration in the channel because their formal preconditions do not change. More explicitly, before and after blocking, sending a message depends on the same condition, namely that neither of the users in this channel has blocked the other. This can simply be modeled with an appropriate AHLI net structure. We see later that there are actions like deletion of contacts that need reconfiguration because they do change preconditions.

Obviously, starting a call introduces new possible actions (see configuration “DirectChannel, InCall” in Fig. 7.1): The called (passive) user can choose to *join*, which starts the communication, or both can *hang up*, which ends the call and therefore is a reconfiguration-triggering action. Both users can *hold* this call to possibly join other calls or *resume* it, without the need for reconfiguration. Note that a call is ended also if a user blocks the other one so that during the call, the action for blocking triggers the same reconfiguration back to the configuration “DirectChannel” as the one for hanging up (denoted with the action’s bold border).

Finally, the second user can *forward* the call to a third user of his contact list, and both users can *add* users from their contact lists to extend this call to a group call. The effect of these actions for this particular channel between the two users is the reconfiguration back to the “DirectCall” configuration. We discuss the remaining effects such as building up the call channel between the first and the third user or the creation of a channel for a group call soon in more detail.

Contact exchange, permitted channels, and deletion of contacts In all the three channel configurations in the top row of Fig. 7.1, a user can ask for the permission of the other user to exchange contacts so that each user appears in the contact list of the other one.

Requesting a contact adds two actions to the set of possible actions for the asked user: He may *accept* or *deny*, which is illustrated by the reconfigurations along *accept* to the configurations in the second row in Fig. 7.1, of which the names of the first row’s configurations are extended by *ContactExchange*. Note that the horizontal reconfigurations for the other reconfigurations remain the same, which indicates that the net structure for contact exchange might be independent from the direct channel’s one. Note also that contact requests in Skype can exist crosswise between two users at the same time until one of them accepts.

If the asked user denies this request in either of the configurations in the second row, the channel falls back to the configurations in the first row by deleting the two actions related to contact exchange.

If the asked user accepts, the preconditions of sending messages and starting calls change. In Skype, every user can set his privacy settings to that either only contacts on the own contact list or all users (via direct channels) are allowed to send messages or to call. Therefore, if the users of a direct channel have agreed to exchange their contacts, the privacy settings do not longer have to be respected by *sendIM* and *call* because the settings can only restrict incoming communication of users that are not in the own contact list, anyway. Once a request for contact exchange has been accepted for a direct channel, we call it *permitted channel* as in the third row of Fig. 7.1.

This is the reason why we also have to reconfigure transitions such as the one for sending instant messages in the step from direct channels to permitted channels or if a contact is deleted or undeleted (cf. Fig. 7.1) and why we have to distinguish these configurations at all. For the configuration, if a user deletes another user from his contact list, we have to take care that the actions *sendIM* and *call* of the deleted user again have to respect the privacy settings of the deleting user. This means that the precondition for these transitions changes, which needs a reconfiguration (cf. Sect. 5.1.2.2). Intuitively, we could again fall back to a direct channel configuration with contact request. But as we mention in the description of contact management in Skype in Sect. 3.1.1, we assume that deleted users can always be restored back to the contact list without asking for permission again so that we now make the design choice of allowing to undelete deleted contacts freely (see bottom row in Fig. 7.1) and going back to the permitted channel configuration.

Note that we suppose *delete* and *undelete* to be actions in the context of the client core and not of this channel. In order to realize undeleting of contacts in direct channels that resulted from deleting a contact in permitted channels, the client core manages lists of actual and deleted contacts. So, even if we e. g., allow for the unique core transitions *delete* and *undelete* to trigger a reconfiguration of configuration “DirectCall”, it still depends on whether the other contact really has accepted before, as the bottom row in Fig. 7.1 suggests.[2]

Again, as for the contact exchange compared to the direct channel, the effects of starting and finishing calls seem not to depend on whether the channel is direct, permitted, or next to a contact exchange so that we might formulate a single reconfiguring rule for the channels with and without “InCall” in Fig. 7.1 that achieves the desired effect.

Forwarding calls Basically, forwarding a call is like hanging up the current one and calling a third contact. However, forwarding cannot be simulated by the reconfigurations in Fig. 7.1 because we assume that the action *call* that connects the forwarded user to the third user belongs to the forwarded user, and we restrict actions (represented by transitions in the client components or channels) to be invoked by their owning user, only. The forwarding user cannot hang up the call and then somehow order the forwarded client to initiate the call to the third contact. Therefore, we need a special reconfiguration to realize forwarding. Note that the reconfiguration depends not only on the channel status of the currently calling users but also whether there is already a direct channel between the forwarded and the third client, which otherwise has to be created by this

[2] Although the channel structure itself is sufficient for reconfiguration rules to decide the current relation and blocking status between the users, a separate place that represents the contact list in the client core is useful to access user names for other actions such as adding contacts to groups.

reconfiguration as well.

7.1.3. Managing groups

Although there is an intuitive understanding of groups in communication applications, we have to look carefully at how groups and related features are realized in Skype, i. e., how they are handled in the user interface, so that we can make appropriate design choices for our model.

The initial action for every group in Skype is to found it, either by selecting several contacts in the contact list and invoking one of the actions “New group conversation”, “Call the group”, or “Send IM” from the context menu (see Fig. 3.6) or by extending a running direct call with another user by adding users from the contact list (see Fig. 3.5). We can observe that in both cases a new window such as in Fig. 3.8 (similar to the ones for direct calls) is opened, providing actions to manage and communicate with this group. It is notable that one can be in a group together with other users without having any other relation to these users, to say, without a direct channel representing some interaction with one of these users that occurred in a communication window. From this, we can conclude that a group should be modeled as an own kind of channel, similarly but independent of the direct and permitted channels before.

For this case study, we simplify some of the complex requirements for conferences and groups from Skype in Sect. 3.1: Firstly, although groups can be founded for contact selections of arbitrary size in Skype, we restrict the founding action to three people (the group owner and two contacts from the contact list), which is the minimum for this action. Because the owner can afterward still add as many contacts as he wants, every group constellation can be created out of the initial group of size three. Then, we ignore the special rights of the “first” callee in Table 3.1 and only consider a group owner, a host of a conference call in this group, and the remaining group members. Finally, we do not regard that all group members get the right to remove other members as soon as the owner of the group left the group. Instead, we do not allow the owner to ever leave his group. These modifications to the original behavior of groups in Skype are moderate and still adequate for representing most of the concrete use cases, and we allow for all the other details on rights in Table 3.1.

From these assumptions and Table 3.1, we get the following dependencies between actions and configurations of the group channel structure (see the left of Fig. 7.2 in the same notation as the configuration diagram for direct channels before):

From his client core, a user *requests a group* with two of his contacts and is from now on the owner of this group with the two contacts as members of the group. The owner is the only one who can *remove* other group members but all members are allowed to *add* their contacts to the group or to leave. Note that we distinguish active members (including the owner) and former inactive members (whose role we denote as “Idle”). This is because in the Skype client, even if a particular user is removed from or leaves a group, he can still access his communication window for this group and read its history, even though his actions are not activated any more but are still visible in the client. Therefore, we need to preserve the client’s connection to this channel. The actions are activated again once the idle user gets readded to the group (which we indicate with the

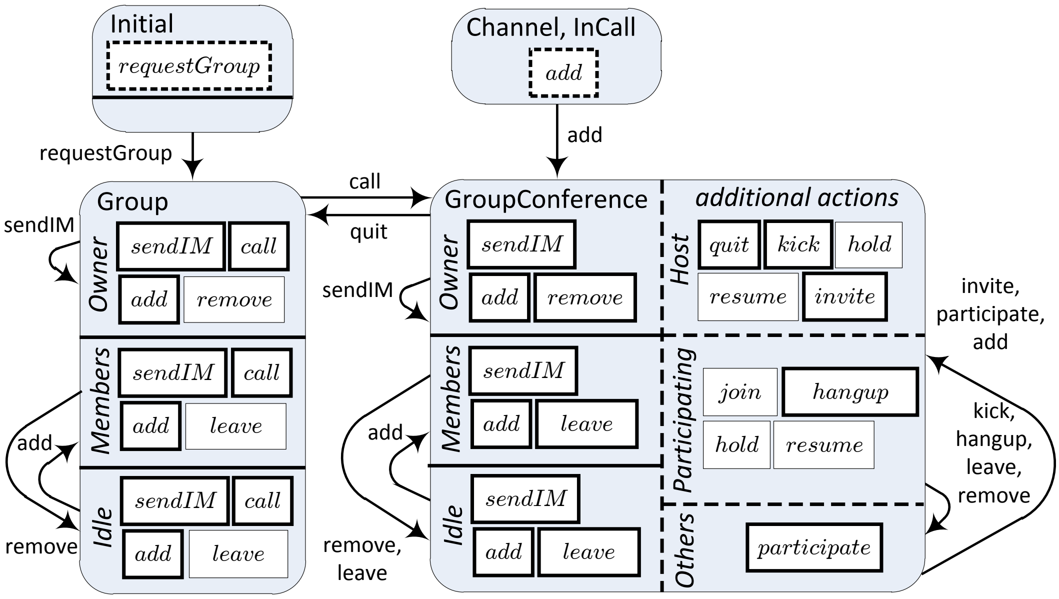


Figure 7.2.: Overview of actions and their effects in groups

separate arrow for *add* from the “Idle” to the “Member” role compartment, and similarly with *remove* the other way round).

As in direct channels, all active members can *send instant messages*, which are sent to all other active members, only. Note that for receiving group messages, it does not matter whether some user has blocked another user in the group. All these reconfigurations result in the group configuration, possibly of bigger size when a user is added that has not been a (active or inactive) member of this group. But we do not regard the concrete size of a group due to the multicasting problem (see Sect. 5.2.2) and handle sending of messages and other actions depending on the size of the group later by formulating appropriate reconfiguring interaction schemes.

Group calls As said before, a group can be founded by extending a running direct call, which results in a group where the owner and the first callee already joined a conference that is hosted by the owner. The added third user becomes a member of this new group and can join the running conference. In Fig. 3.8, this is indicated by the arrow to the configuration “GroupConference” from “Channel, InCall”, which stands for any running call in a direct or permitted channel. Besides constructing the group and the conference structure, this reconfiguration also has to remove the call structure in this direct channel.

When an active member of a group starts a call in the “Group” configuration, he will be the host, regardless whether he is the owner of the group. For this, we still distinguish the roles of owner, active and idle members in the conference. Note that not all active members automatically participate in a started conference. Besides the obvious case when an active member’s client is offline, also contacts that have blocked or are blocked by the host will not participate in the conference, i. e., their client will not be allowed to join the conference. Thus, we further distinguish the roles of the host, the participating members, and the other not participating members (which may be active or idle) and their corresponding additional actions that are possible in the conference.

The host may *quit* the conference, which simply leads back to the group configuration, and the active members still can *send instant messages*. The host can *kick* participating members, which has the same effect as if they *hangup*. Leaving or being removed from the group by the owner implies that participating members are kicked from the conference, too. But as in direct calls, all participants can *hold* and *resume* the conference. The host may *invite* nonparticipating active members, which changes their role to participating so that they can join the conference, or he may *add* a new member to the group who will also be participating instantly. Even if a user has been *kicked* or was not participating due to being blocked by the host, he can choose to *participate* in the conference, which will be successful if the host has not blocked this user at this moment in the direct channel between them. In this case the user will also automatically have joined the conference.

For *invite*, *participate*, *kick*, and *hangup*, it should be obvious that we need reconfigurations because the affected user's part of the actions in the context of this group conference changes. Moreover, *remove* and *leave* are reconfiguring actions as well in the conference, because they imply the effects of *kick* and *hangup*, respectively.

7.1.4. Model-Specific Simplifications

If the reader knows Skype from using it, he or she may argue that although we cover most actions a user can perform via the interface of a Skype client that are important for our case study, one can find some actions as buttons or menu entries in Skype that are a bit different in their effects than in this overview or that are (seemingly) missing although we have not explicitly chosen to exclude them in Sect. 3.1. We take a look at some examples and why we do not consider them in particular in this case study.

7.1.4.1. Omitting Shortcuts

A first difference that might be understood as a discrepancy to our description can be seen in the context menu entries for "Call the Group" in Fig. 3.6. Assume that there does not yet exist a group structure for this choice of two contacts in the contact lists. One could expect that we model this action with an own request for calling a group triggering a reconfiguration that creates the needed group structure with a running conference where the group owner is the host. But following the modeling principle in item 7 of Sect. 3.2.6, we can spare us the need for modeling this reconfiguration explicitly because we already have envisioned to model the single actions for founding a group and starting a call in a group. Obviously, the mentioned context menu entry is just a shortcut having exactly the same effect as the action entry "New Group Conversation", which opens a new group window such as in Fig. 3.8, followed by invoking the action of the button "Call group" in this group window. Instead of modeling another complex reconfiguration, we can simply assume that the action "Call group" in the context menu of the user interface is realized by this very sequence of actions that are represented in the client component of our model.

The same holds for the entry “SendIM” in the same context menu and (not depicted) for the blocking action in the context menu of a group window. The latter is a shortcut because we agreed to model the blocking action for a contact in the unique direct channel to this contact’s client, which may not exist yet, even if there is an existing group. Remember that all group members can add contacts from their contact list, which do not necessarily have to be on the contact lists of the other group members. Thus, we consider blocking an “unconnected” user in a group as a shortcut for requesting a direct channel to this user (even though the window for this direct channel does not pop up in the interface) and then performing the block in this direct channel.

An even more obvious example for an action in Skype that we do not need to model is sending contacts from the own contact list to another user. The effect of this action is that the user who receives the contacts gets a link via the direct chat that just opens a window for requesting exchange of contacts with the sent contact, exactly as if the receiving user would just look up the sent contact himself in the Skype user registry and requesting the contact, which is the action that we have agreed to model. This seems just to be a convenience function for sharing contact names so that users do not have to send messages such as “Hey X, why not asking user Y for his contact?”

By the same arguments, we can restrict ourselves to modeling the creation of groups for three users because there is the possibility of adding more contacts, which is equivalent to creating the group for all the added contacts at once.

We do not consider action sequences as shortcuts if some subactions are performed by other users than the invoking user of the composed action (cf. item 4 of the modeling principles in Sect. 3.2.6).

7.1.4.2. Feigned Shortcuts

We have to be careful whether a supposed shortcut really can be realized by the single actions. Consider for example adding a contact to an ongoing direct call, which immediately results in a running conference of a group containing the two users of the direct call and the added user. At first sight, one could think of this action as a shortcut of hanging up the direct call, opening a group channel with the three contacts and starting a conference. But if the two users of the original call have not yet exchanged their contacts (cf. first row of Fig. 7.1), the supposed shortcut action cannot be realized by a group creation because groups can only be created for contacts on the own contact list.

7.1.4.3. Avoiding inconsistent configurations

The previous example of creating a conference with a user that is not yet on the own contact list is not a big problem because in principle the mentioned conference can be realized by normal group creation as soon as the group owner gets the corresponding contacts on his contact lists. However, if we blindly consider it as a shortcut, the result is just that in some cases the intended behavior cannot be realized in our model as it is meant to be.

A more severe problem would arise if there existed a firing sequence in the net leading to a configuration that is inconsistent w. r. t. the functional requirements, e. g., that a user is in a conference of a group without being an active member of this group, which we have to avoid. When modeling the action for leaving (or kicking a user from) a group, we must take care that the corresponding reconfiguration forces a user who participates in a running conference within this group to hang up in this conference (cf. Sect. 7.2.5.6).

As a general guideline, we can assume actions to be realized as shortcuts as long as there cannot occur firing sequences in the net that result in dysfunctional configurations. We demand that the shortcut's effects can be realized by firing the composed transitions but we have no means to force them to fire immediately one after another (realizing the shortcut as an atomic action) nor do we want to achieve that; this would interfere too much in the firing behavior of the platform net, anyway.

7.1.4.4. Actions depending on other actions

In Sect. 7.1.1, we have chosen to treat the deactivation of the Skype client as an action in the client core. In the Skype software client, a user can always deactivate it, even if there are calls running or possibly put on hold. These calls are ended immediately when going offline. For the case study, we accept a simplification and do not allow the transition for deactivating to be fired as long as the client is involved in at least one call or conference, even if it is put on hold. Otherwise, the deactivation would need to trigger a complex reconfiguration that removes the deactivating client from these calls and conferences. Instead, we model the deactivating transition to depend on a marking that implies that the client is not busy as we demonstrate in Sect. 5.1.1.3. If the transition for deactivating fires, all calls have to be ended before. Although deactivating now depends on ending all calls before, we know that our model can represent and realize all possible actions of the Skype client software.

Nevertheless, we demonstrate a possibility to realize such actions implying optional effects at the example of a group member leaving this group. If the leaving member is in an ongoing conference in this group while leaving, he needs to hang up in the conference or even, if he is the conference's host, to quit it completely (also kicking all other participants). A request handler realizing and relating these reconfigurations is discussed in Sect. 7.2.5.6 on page 220.

7.2. A Communication Platform Model of Skype

In this section, we present a Communication Platform model (see Def. 5.3.3) *SCPM* for the requirements of a Skype model from the previous section. The parts of *SCPM* are described in detail the following subsections. We define

$$SCPM = (\Sigma_{Skype}, A^{Skype}, \emptyset, R, IS, S_{Req}, handler), \text{ where}$$

- Σ_{Skype} and A^{Skype} are the signature and algebra for Skype AHLI nets presented in Sect. 7.2.1, where also the set of request sorts S_{Req} of Σ_{Skype} is defined,
- the initial configuration of the Skype platform model is the empty AHLI net $\emptyset$ over Σ_{Skype} and A^{Skype},

- R and IS are the sets of all AHLI transformation rules and loose interaction schemes presented in this section,
- $handler\colon Constr(S_{Req}) \to PSEQ(R \cup IS)$ is the token request handler function (see Defs. 5.2.16 and 5.2.17) defined in this section, which we also give in a summarized form in Sect. 7.2.7.

7.2.1. Extended Communication Platform Signature and Algebra for the Skype Platform Model *SCPM*

Table 7.1 shows the signature Σ_{Skype} for our Skype Communication Platform model *SCPM*, which encompasses the general signature for Communication Platforms Σ_{CP} presented in Sect. 5.1.3. From the subsignature Σ_{CP}, we use the sorts *ID* for user identifiers, *Data* for an abstract representation of data, *Control* for simple "black" control tokens, and *Bool*, *Int*, *BoolInt* for switches and counters.

The purposes of the sorts and operations in Σ_{Skype} as they are used in the definition of the algebra A^{Skype} in Table A.4 are the following:

State represents the state of the client with two values *Online* and *Offline*. For the case study, we ignore further possible client states (cf. Sect. 7.1.1).

Setting values are used to represent a client's privacy settings. Every *Setting* value is a pair of the *Option* values *ContactsOnly* and *Anyone*. The first value in a *Setting* pair represents the option for incoming messages and the second one for incoming calls.

History terms represents lists of log entries. The value *Empty* is the empty list. Every operation yielding a *History* term — i. e., *sentIM*, *blocked*, *unblocked*, *called*, *participated*, *invited hungup*, *forwarded* — takes an *ID* value for the user name who performed an action to be logged, another *ID* or *Data* value for the target user or data of the logged action, and the *History* this logged action is to be appended to.

The set of token request sorts S_{Req} consists of the remaining sorts. The operations that yield these sorts form the set of token request constructors $Constr(S_{Req})$.

ComRequest are request tokens for reconfigurations that treat communication in general, i. e., creating a direct channel (*directReq*), starting and hanging up a call (*callReq*, *hangupReq*), forwarding a call (*forReq*), creating groups (*groupReq*, *exGReq*), starting and quitting conferences in groups (*callGReq*, *quitGReq*), adding people to groups and conferences (*addGReq*), participating in a conference (*partGReq*), and leaving or being kicked from conference or group (*leaveGReq*, *hangupGReq*).

ContactRequest tokens are for announcing reconfigurations that change the relation of two clients to each other, i. e., asking for contact exchange (*contactReq*), permitting or denying a contact exchange (*permReq*, *delCXReq*), and deleting or undeleting a contact (*delCReq*, *undelCReq*).

$\Sigma_{\mathit{Skype}} = \Sigma_{\mathit{CP}}+$	
sorts:	$\mathit{State}, \mathit{Option}, \mathit{Setting}, \mathit{History}$
request sorts:	$\mathit{ComRequest}, \mathit{ContactRequest}, \mathit{TransmitRequest}$
opns:	$\mathit{Online}, \mathit{Offline}\colon \to \mathit{State}$ $\mathit{ContactsOnly}, \mathit{Anyone}\colon \to \mathit{Option}$ $\mathit{Empty}\colon \to \mathit{History}$ $\mathit{makeSetting}\colon \mathit{Option}\ \mathit{Option} \to \mathit{Setting}$ $\mathit{sentIM}\colon \mathit{ID}\ \mathit{Data}\ \mathit{History} \to \mathit{History}$ $\mathit{blocked}\colon \mathit{ID}\ \mathit{History} \to \mathit{History}$ $\mathit{unblocked}\colon \mathit{ID}\ \mathit{History} \to \mathit{History}$ $\mathit{called}\colon \mathit{ID}\ \mathit{ID}\ \mathit{History} \to \mathit{History}$ $\mathit{part}\colon \mathit{ID}\ \mathit{History} \to \mathit{History}$ $\mathit{invited}\colon \mathit{ID}\ \mathit{ID}\ \mathit{History} \to \mathit{History}$ $\mathit{hungup}\colon \mathit{ID}\ \mathit{History} \to \mathit{History}$ $\mathit{forwarded}\colon \mathit{ID}\ \mathit{ID}\ \mathit{History} \to \mathit{History}$
request constructors:	$\mathit{directReq}\colon \mathit{ID}\ \mathit{ID} \to \mathit{ComRequest}$ $\mathit{contactReq}\colon \mathit{ID}\ \mathit{ID} \to \mathit{ContactRequest}$ $\mathit{permReq}, \mathit{delCXReq}\colon \mathit{ID}\ \mathit{ID} \to \mathit{ContactRequest}$ $\mathit{callReq}, \mathit{hangupReq}\colon \mathit{ID}\ \mathit{ID} \to \mathit{ComRequest}$ $\mathit{forReq}\colon \mathit{ID}\ \mathit{ID}\ \mathit{ID} \to \mathit{ComRequest}$ $\mathit{delCReq}, \mathit{undelCReq}\colon \mathit{ID}\ \mathit{ID} \to \mathit{ContactRequest}$ $\mathit{groupReq}, \mathit{exGReq}\colon \mathit{ID}\ \mathit{ID}\ \mathit{ID} \to \mathit{ComRequest}$ $\mathit{callGReq}, \mathit{quitGReq}\colon \mathit{ID} \to \mathit{ComRequest}$ $\mathit{addGReq}\colon \mathit{ID}\ \mathit{ID} \to \mathit{ComRequest}$ $\mathit{partGReq}\colon \mathit{ID} \to \mathit{ComRequest}$ $\mathit{leaveGReq}, \mathit{hangupGReq}\colon \mathit{ID} \to \mathit{ComRequest}$ $\mathit{transGReq}, \mathit{transReq}\colon \mathit{ID}\ \mathit{Data} \to \mathit{TransmitRequest}$
vars:	$p, p'\colon \mathit{Setting} \qquad o\colon \mathit{Option}$
tokenvars:	$o1, o2, o3\colon \mathit{Options} \qquad h, h1, h2, h3, \ldots\colon \mathit{History}$

Table 7.1.: Signature Σ_{Skype} for the Skype CP Model *SCMP*

TransmitRequest tokens encapsulate data to be transmitted or multicasted in exclusive communications (cf. Sect. 5.1.1.2) such as calls and conferences (*transReq*) or for multicasting messages in group chats (*transGReq*).

With this, we have a request token for every user action with reconfiguration that we describe in the modeling requirements (see Figs. 7.1 and 7.2). The usage of these operations and the handlings of the request tokens are described in detail in the remaining sections of the case study.

Note that we use unnumbered variables for the arc inscriptions (possibly with primes) and numbered variables for the token variables. This holds also for the inscriptions and token variables that are defined in Σ_{CP} in Table A.1.

7.2.2. Graphical Notations for the Case Study

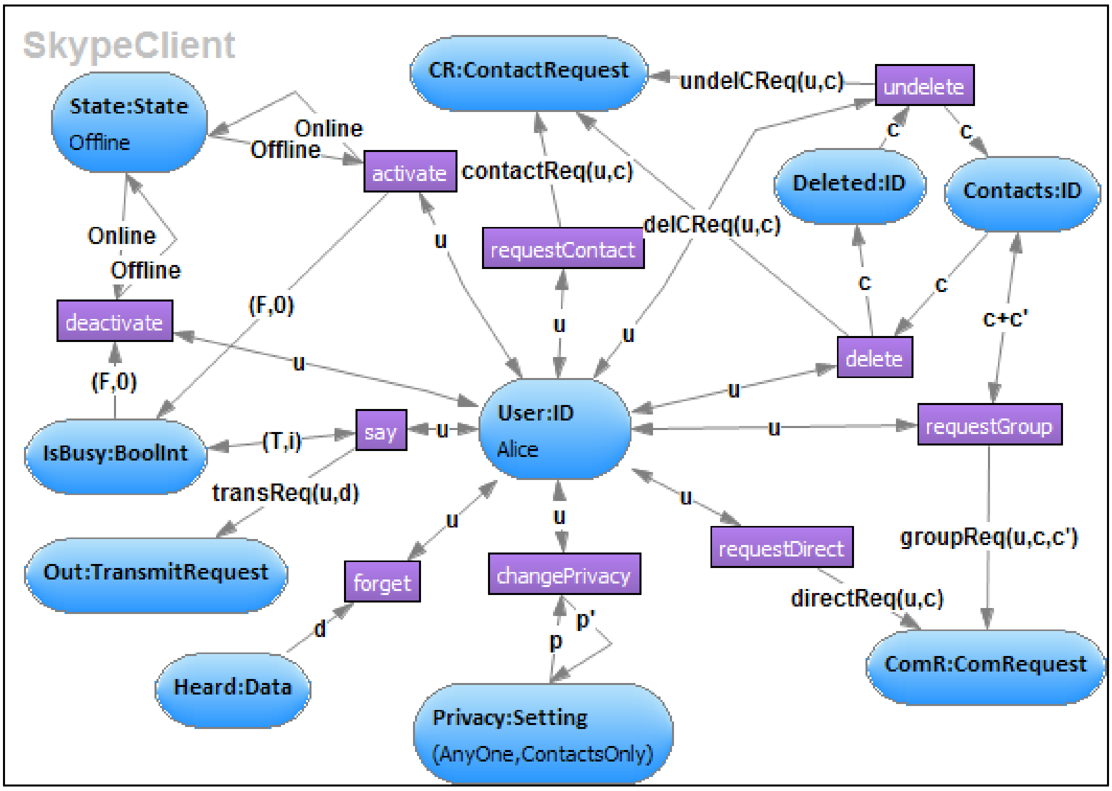


Figure 7.3.: Core component of a Skype client

For the nets and rules of this case study, we use a simplified graphical notation of AHLI nets as in Fig. 7.3. The rectangles are transitions and the ovals are places with inscriptions denoting the name and type of a place, e. g., the place `User` of type *ID* in the center. Below the place's name, we arrange the values of the tokens that mark this place. For example, place `User` is marked with one token with the algebraic value *Alice*.[3] Although we are dealing with AHLI nets and individual token markings, the actual individuals are not relevant, so we omit the token's individuals in the graphical notation and depict only their algebraic values on the corresponding places.

Furthermore, we do not explicitly state the set of token variables Y_ϱ for a rule ϱ). The set of token variables of a rule îs always the least subset of the set of tokens variables Y from the signature Σ_{Skype} in Table 7.1 that contains all variables used as tokens in the LHS, the RHS, and the application conditions' nets of this rule. For the kernel and

[3] See the carrier set A_{ID} for identifiers in the common algebra for Communication Platform Models in Table A.2, which we reuse in the algebra A^{Skype} in Table A.4 for the Skype platform model *SCPM*.

multi rule of an interaction scheme, the set of token variables contains all token variables occurring in any LHS, RHS, or application condition of the kernel or a multi rule in this scheme.

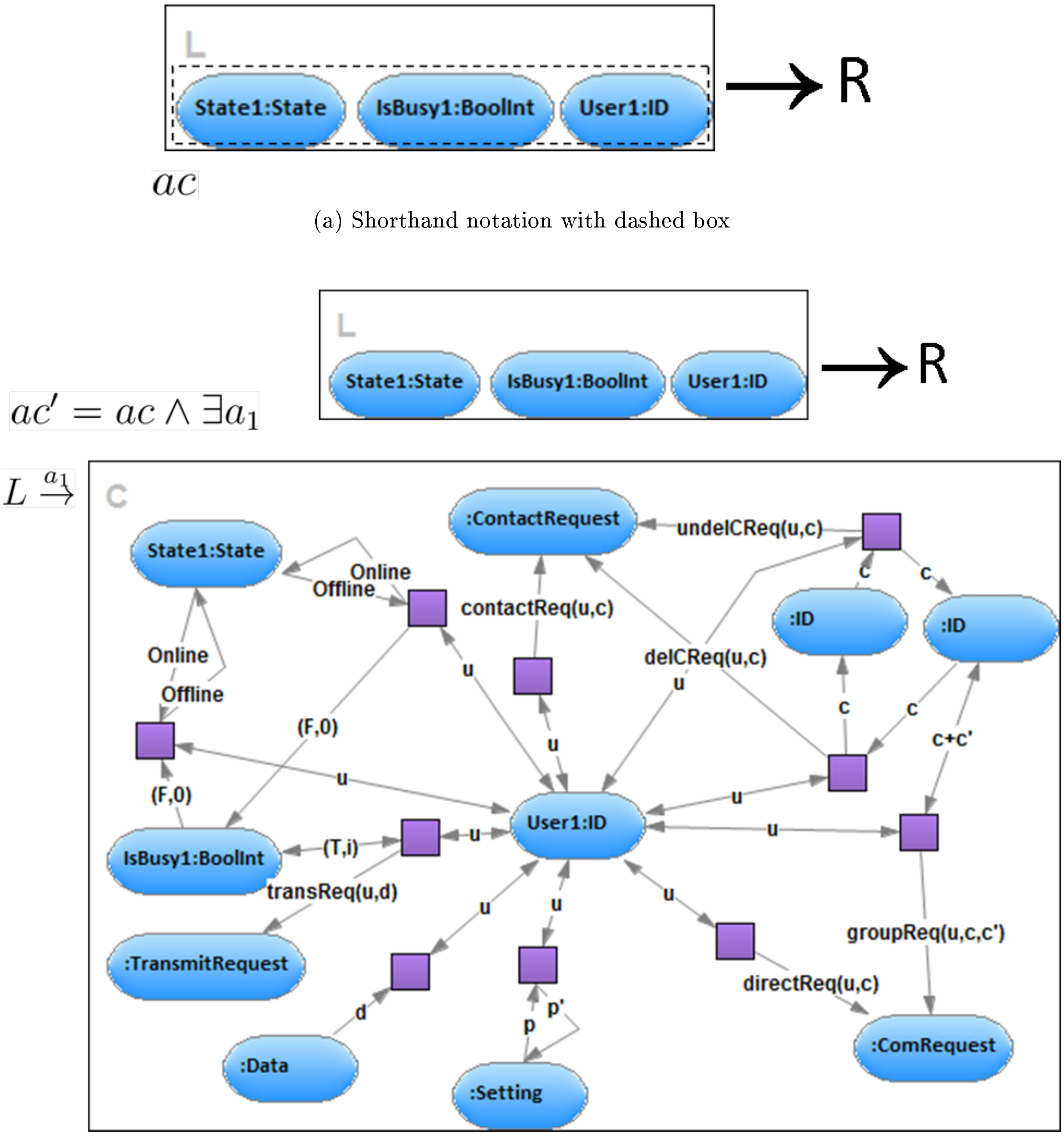
The arcs are inscribed with Σ_{Skype} terms over variables. A special short notation are inscriptions such as $u \mathrel{!=} c$ on the arcs going out from the place `State`, which formally means that the arc has the actual inscription u and the adjacent transition has a firing condition $eq(u, c) = F$. We assume all other transitions (without any arc that has such an inscription) not to have any firing condition.

In the following, we depict rules with their LHS and RHS (cf. the representation of rules in Sect. 5.3), possibly arranged vertically for complex rules. Additionally, we use further notational elements and abbreviations that shall help to visualize the nets and reconfiguration rules in a more compact way and to help to grasp the deleted and created net elements and thus the effect of complex rules more quickly. As in the rule given in Fig. 7.21, we surround net elements by colored boxed if they are manipulated by the rule: We use green boxes for created element in the RHS (e. g., transition *resume1* and place `OnHold1`), red boxes for elements in the LHS that are not preserved and therefore do not occur in the RHS (e. g., transitions *call1* and *call2*), and blue boxes exclusively for transitions that are preserved but whose environment — i. e., the incoming and outgoing arcs — changes (e. g., transitions *block1* and *block2*)[4] and for tokens that are replaced by other tokens (e. g., the token with the variable term $h$is replaced by a token with the term $called(u1, u2, h)$).

Implicit application conditions for client matching A recurrent pattern in the rules is that places and transitions of a specific core Skype client should be matched. For example, if we want to express that a rule $L \to R$ with an application condition ac should match certain places of a core Skype client (see Fig. 7.3), we use a dashed black box as shown in Fig. 7.4a to surround the elements belonging to one client core. This is a shorthand notation for the rule shown in Fig. 7.4b, whose application condition is a conjunction of ac and the condition $\exists a_1$, which demands that the places of the LHS match the corresponding places of a client core structure as given in the condition net C. We may use multiple dashed boxed in the LHS, which means that for each boxed area in the LHS, we have an implicit application condition such as $\exists a_1$ in Fig. 7.4b. For an example of such a rule, see Fig. 7.7, where the places in the top row in the LHS are supposed to match a client core and places in the lower row to another client core, respectively.

Compact notation for simple interaction schemes We only use simple interaction schemes with only one multi rule (cf. Sect. 5.3) in this case study. Moreover, we choose kernel morphisms as inclusions so that the multi rule contains the kernel rule. With these assumptions, we can depict interaction schemes in a compact way: In the following, the figures for simple interaction schemes show the multi rule $L \to R$ where the kernel rule $L_K \to R_K$ is indicated by a dotted blue box that contains the parts in the image

[4] Formally, these transitions are deleted and created again with a new environment, due to the properties of AHLI net morphisms.



(a) Shorthand notation with dashed box

(b) Implicit rule with application condition

Figure 7.4.: Short notation for matching places and transition of a client core

of the kernel morphism (e. g., see Fig. 7.28). If there are tokens on kernel places that are not intended to occur in the kernel, these tokens are surrounded by a dotted box. Application conditions for interaction schemes are named ac for the multi rule (starting with L) and ac_K for the kernel rule (starting with L_K).

7.2.3. Client Core Components

The AHLI net shown in Fig. 7.3 represents the Skype client core component of an example user named Alice. The owner of a client is determined by the *ID* token on the place named `User`, to which every transition representing an action of Alice is connected in the uniform way that they just read the user identifier token via a bidirectional arc inscribed with the reserved user variable u (cf. Sect. 5.1.1.1 and Sect. 5.3). Moreover, the client is currently in the state *Offline* as one can tell from the marking of the place `State` in the top-left corner. The state can change from *Offline* to *Online* by firing *activate* and back to *Offline* by firing *deactivate*. On the one hand, activating provides a combined switch–counter token $(F, \mathit{0})$ to `IsBusy`, which means that immediately after activating the client is not busy and there is not a call ongoing in any channel (cf. Sects. 5.1.1.3 and 7.1.4.4). On the other hand, *deactivate* demands and consumes this token value on firing. The current privacy settings are determined by a *Setting* token on the place `Privacy`: The first element of the tuple denotes the setting for instant messages and the other element the setting for incoming calls. Hence, in this configuration only users that are present on Alice's contact list (currently none) are allowed to send messages to her, and everyone is allowed to call her. All the other transitions in this figure have effects that are realized by handling the request tokens they produce (cf. Sects. 5.1.2.1 and 5.2.3).

In Skype, chats are a nonexclusive form of communication because a user can always participate in several chats at the same time — technically in the client as well as consciously —, i. e., he can be writing and receiving messages from different independent sources (cf. Sect. 5.1.1.2). All speech-related communication forms such as calls or conferences are exclusive because a user can only be listening or talking in at most one of those contexts so that in this case he is considered as busy. Therefore, it seems natural to put the action for talking and the "memory" of what the user has heard so far in the client core because these are fundamental actions that are independent from the actual channels the data is distributed through. For representing the sudden appearance of speech data out of the black box of the user's brain, we use a simple transition that generates this data freely with a variable on its outgoing arc that is not bound by a variable on an incoming arc (see transition "say" in Fig. 7.3). For chats in Skype, there are histories logging all messages in the context of the channel in which they have been received (see Figs. 3.4 and 3.8) and there is no possibility to write a message in the client without an open channel, which suggests the chat actions to be part of the respective channel. Following the idea of regarding calls as exclusive communication (cf. Sect. 5.1.1.2), we allow a user to *say* something if he is busy in any call or conference so that its participants can receive the produced *Data* value that is wrapped in a request token of type *TransmitRequest* on place `Out`. Every data that another user "said" and that has been transmitted to Alice's client is put on the place `Heard`, from which a user can simply *forget* what he has heard. This represents the cognitive skills of the client's user in a simple realistic way that is sufficient for our purposes.

The transition *requestGroup* corresponds to the action in Fig. 7.2 and produces a request token for the purpose of founding a group of three users for two of the contacts on the place `Contacts`, which represents the contact list of the client. The remaining transitions *requestDirect*, *requestContact*, *delete*, and *undelete* correspond to the actions in Fig. 7.1 and likewise produce request tokens to be handled besides moving the deleted or undeleted contact token back and forth to the places `Contacts` and `Deleted`. In the following, we explain the structures that are created or removed by these reconfigurations and present the reconfiguration rules and request handler expressions that realize them and further ones till we cover all reconfigurations sketched in Sect. 7.1.

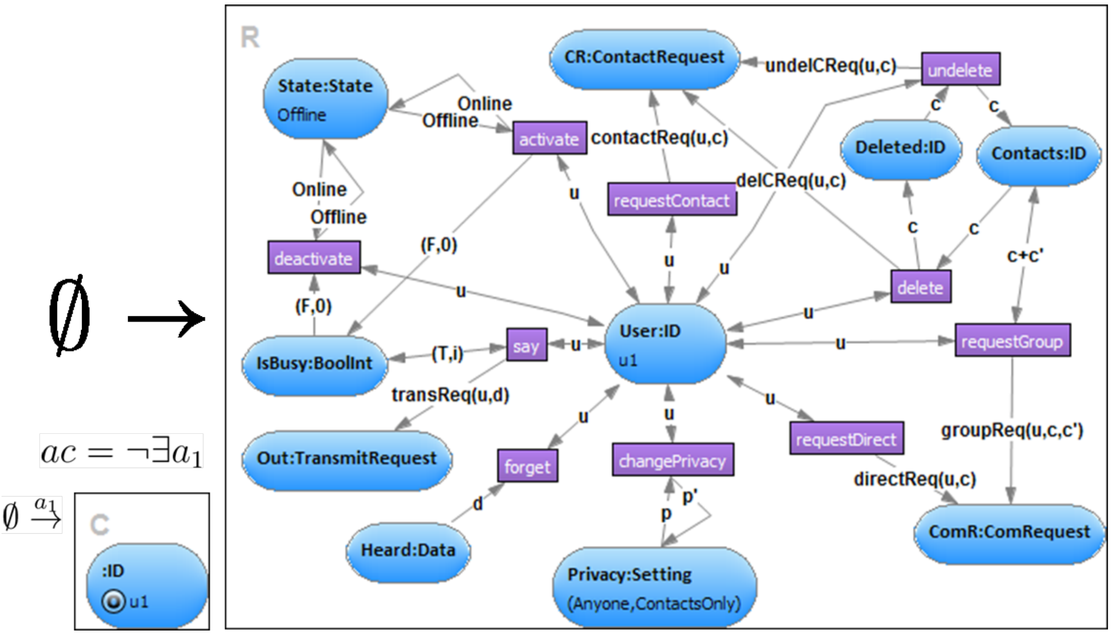


Figure 7.5.: External rule *CreateSkypeClient* with application condition *ac*

Creating Skype clients In Skype, new users can register via the software client application. In our platform model, there is no representation of this action as a transition because prior to registering the user is unknown to the Skype platform. Nevertheless, we can model the creation of a new user client component by the external rule *CreateSkypeClient* in Fig. 7.5, which is applied without a preceding token request (cf. Sect. 5.3). The application condition ensures that there does not yet exist any other client that has an *ID* token with the same identifier as the one that is created by the rule.

7.2.4. Direct and Permitted Channels

In this section, we present and discuss in detail transformation rules and request handlers (see Def. 5.2.17) for the reconfigurations of direct channels that we gathered in Fig. 7.1. For this, we consider fragments of this diagram and describe how every reconfiguration arrow is represented by a user action for the corresponding user and how possible created request tokens are treated by request handlers in a way that realizes the change of possible actions according to the diagram.

7.2.4.1. Requesting a Direct Channel

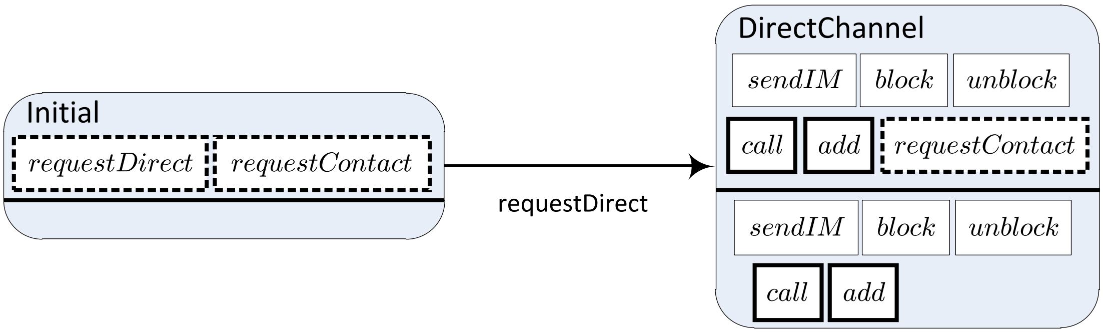


Figure 7.6.: Reconfiguration for requesting a direct channel

We start with the user action in Fig. 7.6, whose reconfiguration is triggered by firing the transition *requestDirect* in the client core component (see Fig. 7.3) that produces a token constructed with the signature operation *directReq* on the place `ComRequest`. The value of this request token wraps the names of the user who requests the direct channel (assigned to the reserved user variable u while firing) and of the user he wants to be connected to (assigned to the contact variable c).

Before formulating the request handler for the *directReq* request, we have to think of the exceptional cases in which the handler has to ignore and discard the request without creating a direct channel. For a direct channel request there is clearly only one exception, i. e., when the channel already exists. With this, we can define the request handler expression for this request constructor:

$$\mathit{handler}(\mathit{directReq}) := \mathit{DiscardDirectReqExisting} \parallel \mathit{CreateDirectChannel}$$

This handler consists of a single priority list (whose elements are separated by $\parallel$) and does not contain further sequenced elements (separated by ;). This means that at most one of the two rules is executed because from such a list, the first applicable rule (in read direction) is applied to the platform net.

CreateDirectChannel We first describe the effect of the rule depicted in Fig. 7.7 that realizes the successful reconfiguration of Fig. 7.6 by creating the channel structure: In the LHS, some places of the requesting user client (in the top) and some places of the requested contact's client (in the bottom) are matched. Note the dashed line expressing an appropriate application condition that states that these places can only be matched to corresponding places of a client core component as in Fig. 7.3; we also hint at that with the number suffixes of the places' (and in the RHS also of the transitions') names. Moreover, the matched identifier token on the identity place in the bottom has to correspond to the concrete name that is assigned by the match to the token variable $u2$, which is also wrapped in the request token on the requester's place `CR1`. Note that this matched request token is deleted by this rule, as it does not occur in its RHS (also denoted for convenience by the red box around the token in the LHS). The rule creates

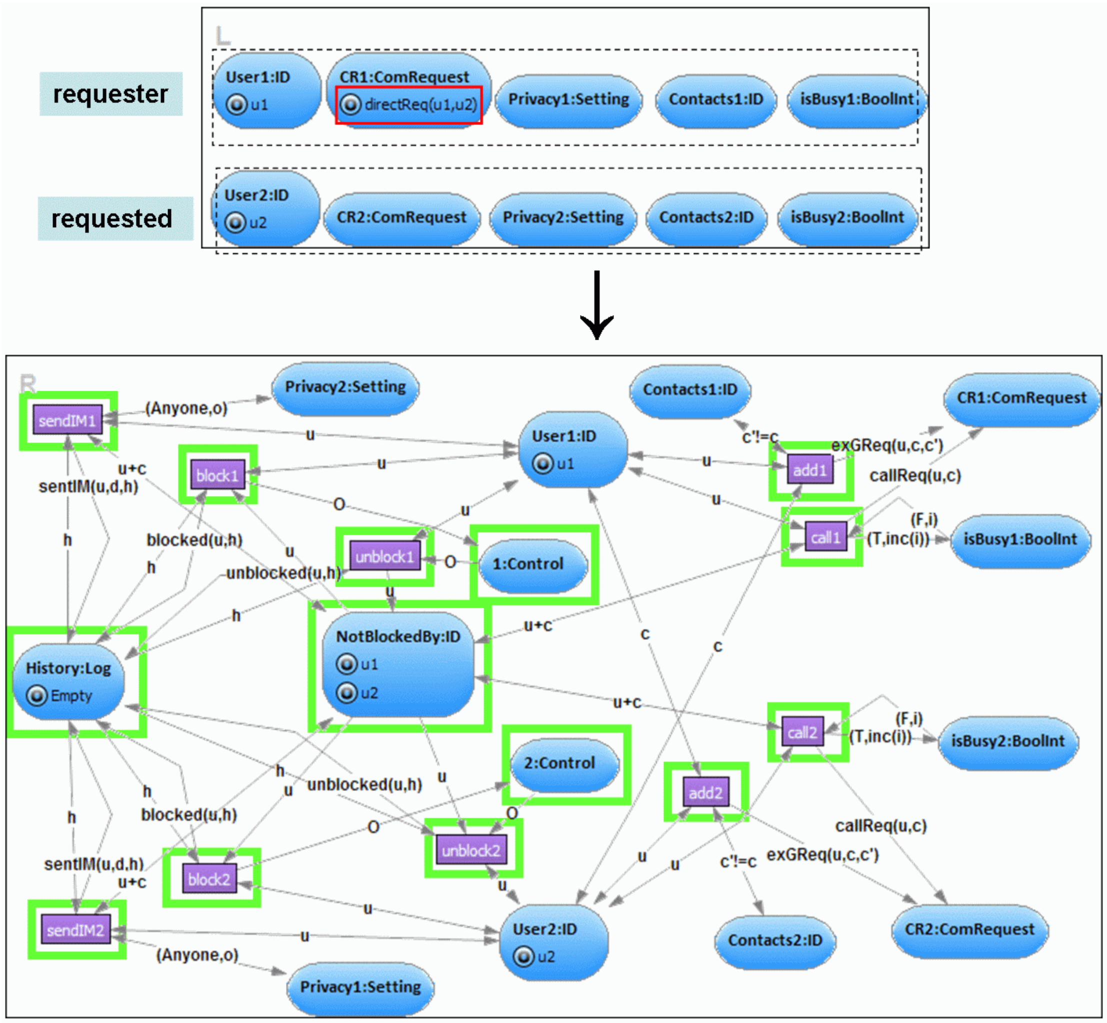


Figure 7.7.: Rule *CreateDirectChannel*

the places and transitions that are surrounded by a green box in the RHS, the other elements are just preserved.

The channel structure in the RHS is symmetric for both connected clients, so we only look at the top part (the places and transitions with names ending in “1”) and the two common places `History` and `NotBlockedBy` on the imaginary horizontal center line. From left to right, the new transitions have the following purposes: *sendIM1* realizes the sending of an instant message by the user with name *u1* (assigned to arc variable u) by appending to the current history token (assigned to variable h) some message (assigned to variable d) wrapped in the algebraic evaluation of the term $\mathit{sentIM}(u, d, h)$. The place `History` is shared by both clients of this channel and carries a history token considered accessible for both clients. Initially, after being created by the rule, this token is an empty log to which the subsequent events in this channel are appended. Further, this transition depends on the setting *Anyone* for messages on the place `Privacy2` belonging to the client of user $u2$ (put in top just for graphical reasons). The second *Option* element in this setting tuple is for calls and thus irrelevant for messages. Therefore, *sendIM1* accepts any value to be assigned to arc variable o. Finally, we allow *sendIM1* to fire only if neither of the users has blocked the other, which is realized by the shared central place `NotBlockedBy` from which two tokens must be available to be assigned to the variables of the arc sum $u + c$.

The transition *block1* removes the token with the value $u1$ from the place `NotBlockedBy` because on firing, the arc variable u is assigned to the value of token $u1$ on place `User1`. Additionally, the blocking action is logged in the history; more precisely, the name of the user who blocked this channel is noted.

Transition *unblock1* can only fire after *block1* because only the latter marks control place `1` with a black token that is required by *unblock1* for firing (cf. Sect. 5.1.1.3). The effect of firing *unblock1* is that the token removed by the firing of *block1* is put back to the place `NotBlockedBy` and the action is logged in the history, again.

The last two transitions announce requests for reconfiguration to the place `CR1` of user $u1$. Firing of *call1* produces the request token $\mathit{callReq}(u, c)$, with the concrete names $u1$ and $u2$ assigned to the arc variables u and c, respectively. This transition also changes the client status on the place `IsBusy1` of user $u1$ from false to true and increases the number of calls this user can participate in (cf. Sect. 7.2.3).

The request $\mathit{exGReq}(u, c, c')$ created by *add1* intends that a possible ongoing call (after firing *call1*) between the users $u1$ and $u2$, whose names are assigned to the arc variables u and c, is "extended to a conference in a group" with an additional user whose name is taken from the contact list place `Contacts1` of user $u1$. Note that we use the simplified notation $c' \mathrel{!}= c$ as arc inscription for expressing a firing condition $eq(c', c) = F$ for the transition *add1*, stating that the third participant for the conference must not be $u2$, who is already in the ongoing call. If *add1* is fired with an ongoing call, the effect of this action has to be the same as founding a group and starting a conference. In the other case that $u1$ and $u2$ are currently not in a mutual call, we regard this action to have the effect of founding a group.

We explain handling of call requests $\mathit{callReq}(u, c)$ in Sect. 7.2.4.4. The effect for the request $\mathit{exGReq}(u, c, c')$ for creating a group (possibly with ongoing conference), depends on how we model groups and conferences. Therefore we give the handler for kind of requests in Sect. 7.2.5.6 after modeling the management of conference calls and groups.

Note that the set of the transitions that are matched, preserved, and deleted by this rule corresponds to the intended configuration change in Fig. 7.6. This goes without saying for all "successful" rules in the following handlers and is easy to check so that we do not mention this correspondence explicitly for the other rules.

DiscardDirectReqExisting The discarding rule is much simpler and just needs to match a structure indicating that there already exists a direct channel and to delete the request token. The rule *DiscardDirectReqExisting* in Fig. 7.8 matches the transition for adding a third contact, which can be used to identify a direct channel between the matched clients. It would be a bad choice to match the transitions for sending instant messages instead, because in Fig. 7.1 we can see that these transitions are subject to reconfigurations and

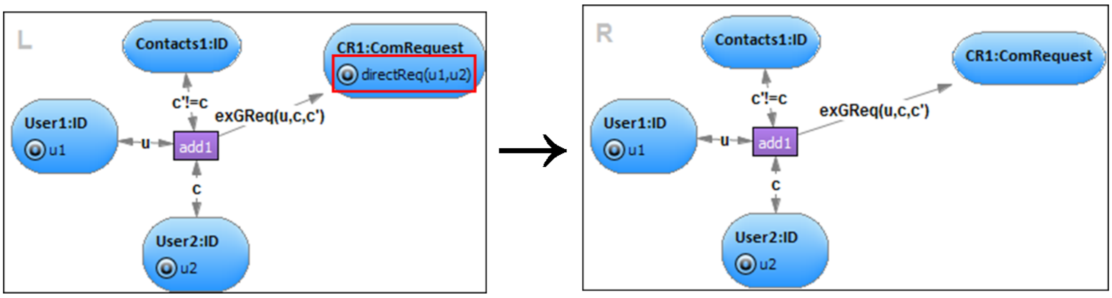


Figure 7.8.: Rule *DiscardDirectReqExisting*

we would have to formulate several discarding rules for all possible structural cases. But the adding transitions will never change in a channel and are a sufficient and good choice for structure identifying a channel. Alternatively, we could also have used the transitions for blocking or unblocking.

Now, we see that it is important that the discarding rule has a higher priority in the handler, i. e., this rule occurs to the left of and is tried to be applied before the creating rule. The creating rule is applicable after firing *requestDirect* even if there already has been created a channel between the two matched clients, so we have to formulate the handler for this request such that the discarding rule is applied in this exceptional case instead of the rule for the successful case.

7.2.4.2. Requesting a Contact

The reconfigurations in Fig. 7.9 describe how a user requests a contact exchange from another user so that each of them appears on the other user's contact list. Although at first sight it seems that we require many rules to realize all the reconfigurations between the different configurations, in fact, the action for requesting a contact exchange does not depend functionally on whether there exists a channel or if there is a running call in this channel. This independence has two consequences for our model. First, we have to provide the transition *requestContact* for creating the request token $\mathit{contactReq}(u, c)$ on place `CR` in the client core as we do in Fig. 7.3. Moreover, whether a user can invoke the actions *accept* and *deny* (as the main actions for modeling this use case) does not depend on any condition, besides that a user can invoke at most one of them. From this functional independence, we can conclude as a second consequence that it is sufficient to handle the contact exchange request for all horizontal configurations "requestContact" in Fig. 7.9 with a single rule creating a common contact exchange structure with transitions *accept* and *deny*. Of course, this single rule realizes just the successful case where a new contact exchange structure is created; we still have to formulate another discarding rule for the cases that the contact exchange structure already exists or if a contact exchange procedure already has been completed successfully between these particular two clients. We define the handler expression for a contact exchange request as

$$\mathit{handler}(\mathit{contactReq}) := \mathit{DiscardContactExchange} \parallel \mathit{CreateContactExchange}$$

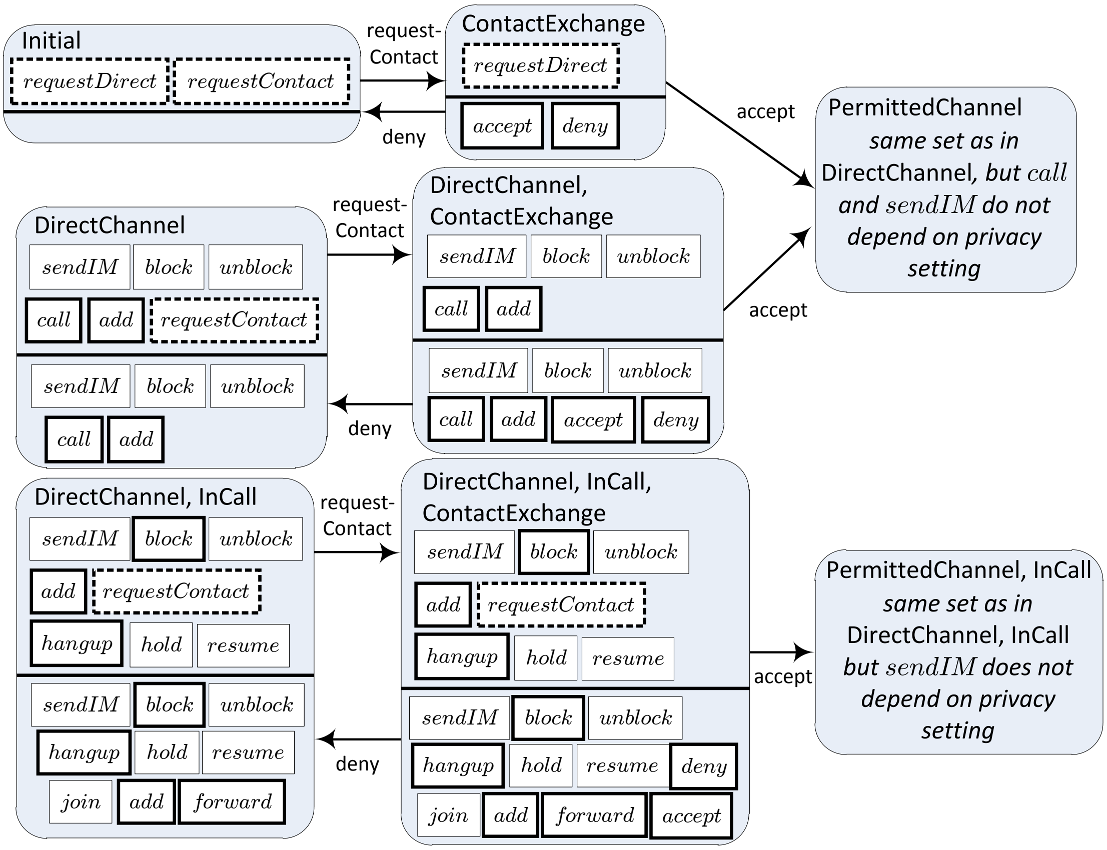


Figure 7.9.: Reconfigurations for contact exchange

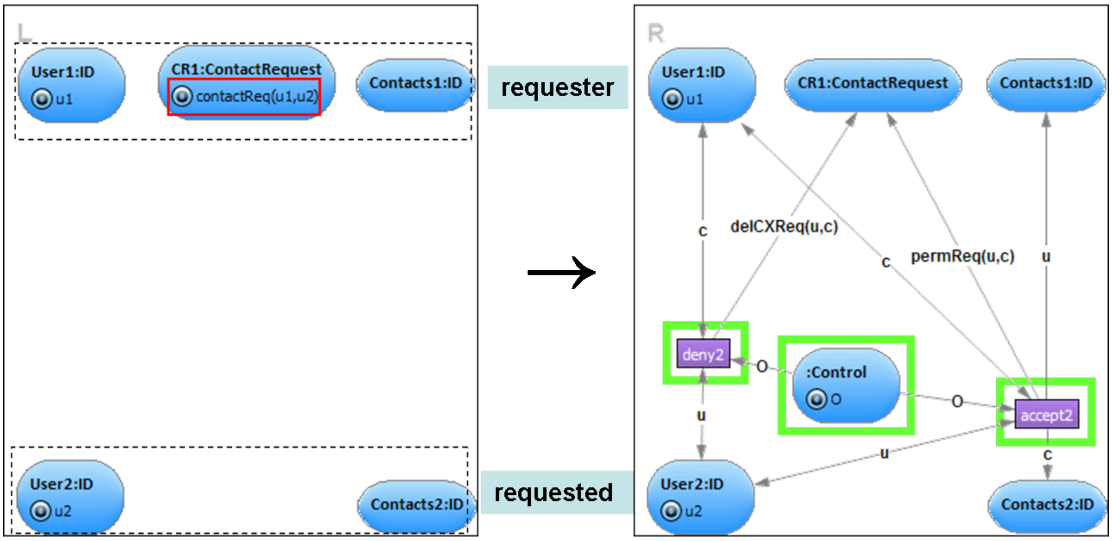


Figure 7.10.: Rule *CreateContactExchange*

CreateContactExchange The rule for successfully creating the contact exchange structure is depicted in Fig. 7.10. It simply creates the two transitions *accept2* and *deny2* for the asked user (note the arcs with the reserved user variable u to the identifier place of the asked user). A marked control place ensures that at most one of these transitions can fire and produces either a request token for accepting or denying the request. For these request tokens, we have to define further handlers.

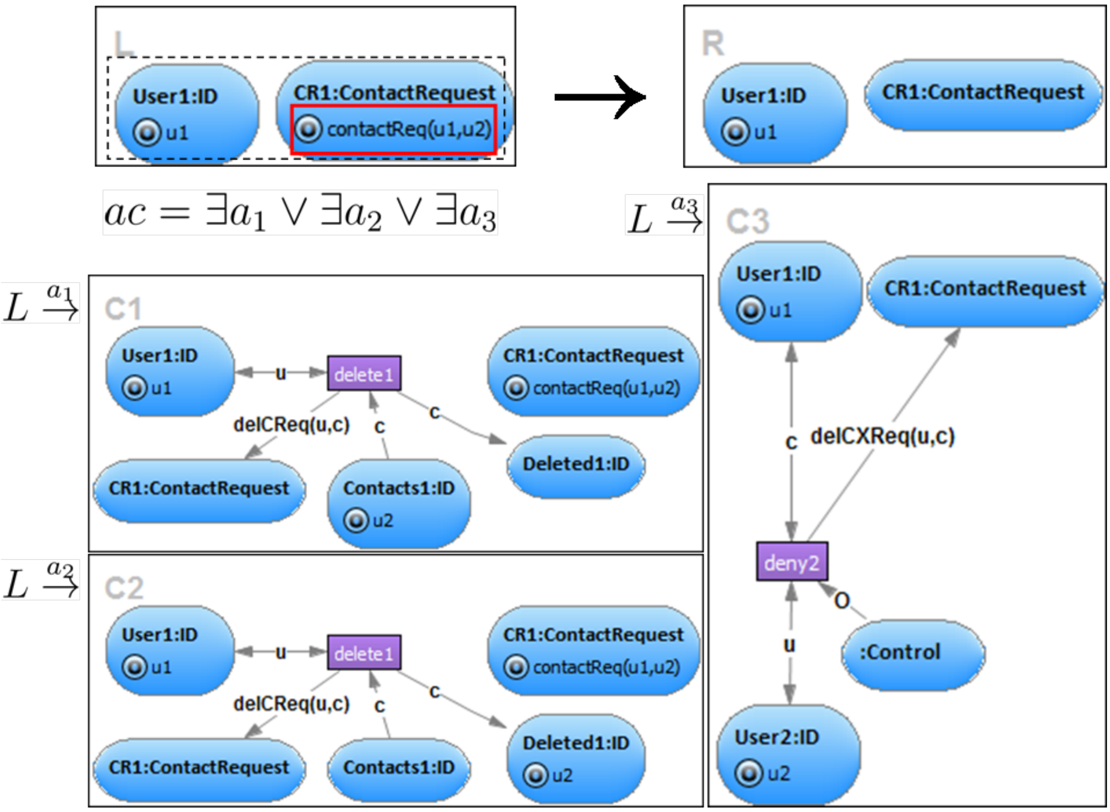


Figure 7.11.: Rule *DiscardContactExchange* with application condition *ac*

DiscardContactExchange The sole effect of the discarding rule in Fig. 7.11 is to delete the *contactReq* request token. This is the first rule for which we use a disjunctive application condition describing the two exceptional cases that this rule has to handle. One of the first two conditions $\exists a_1$ and $\exists a_2$ is satisfied if the requesting user already has exchanged contacts with the target user of the current request, which is the case if the user $u1$ who is posing the request $contactReq(u1, u2)$ already has a token $u2$ either on his place `Contacts1` representing his contact list or on his place `Deleted1` that carries the identifiers of the users that $u1$ has deleted from his contact list (cf. Fig. 7.3). The third condition $\exists a_3$ is satisfied if between the clients of users $u1$ and $u2$ there exists a contact exchange structure as rule *CreateContactExchange* creates. We can be sure that such a structure exists if a transition as *deny2* is matched.

The following handler for the *delCXReq* request realizes all the horizontal “deny” actions in Fig. 7.3, which we model by the transition *deny2* created by *CreateContactExchange*. For this, a single rule is sufficient that removes the whole contact exchange structure. We do not have to regard any exceptional cases.

$$\mathit{handler}(\mathit{delCXReq}) := \mathit{DeleteContactExchange}$$

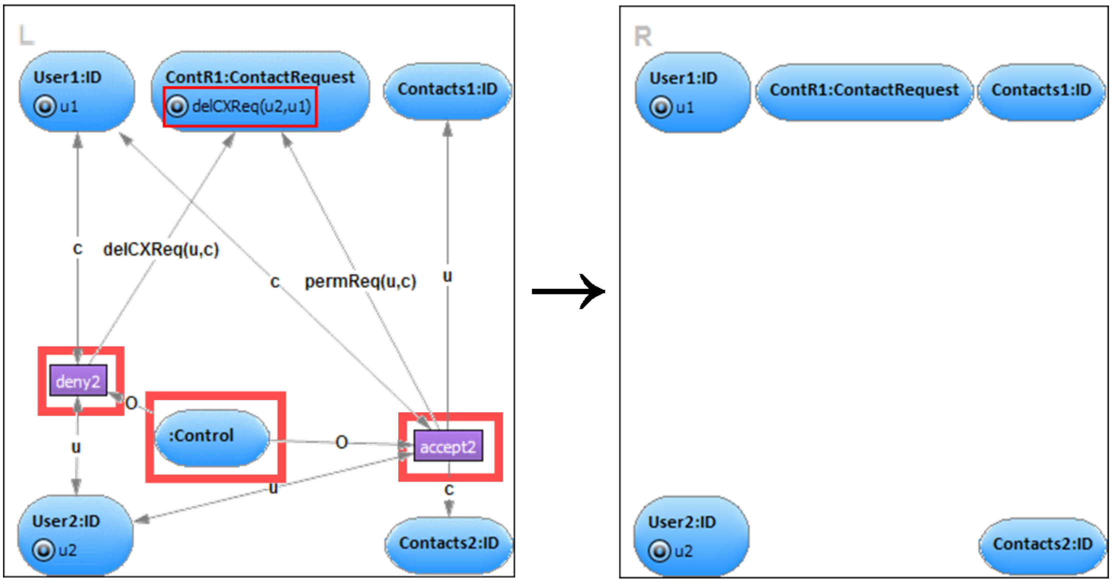


Figure 7.12.: Rule *DeleteContactExchange*

DeleteContactExchange The rule in Fig. 7.12 for deleting the contact exchange structure (after the asked user has denied) is just the opposite of rule *CreateContactExchange*, i. e., with switched LHS and RHS, but with appropriate changes in the markings. For example, the control token obviously is consumed by the firing of *deny2* that creates the request token on the place `ContR1`.

The handler for the *permReq* request for accepting the contact exchange, which realizes the remaining reconfiguration “accept” in Fig. 7.9, is more complex. First, the accepting user may also have requested a contact exchange with the requesting user the other way round so that the accepting user is also still waiting for a decision. The handler should take care of removing this opposite structure as well. Then, we see that the two lower horizontal reconfigurations “accept” outgoing from the direct channel configurations can be realized with a single rule; both just have to make each user’s *sendIM* action independent from the privacy settings of the other user. This might seem wrong, because in the description of the top permitted channel configuration (that is not in a call), we also explicitly assume the action *call* to be independent from the privacy settings. But we model calling in a channel by reconfiguration in Sect. 7.2.4.4, where we take care of distinguishing these cases, anyway. Thus, when a user accepts a contact exchange in a direct channel, we only adapt *sendIM* so that this action can be invoked always without the need to respect privacy settings. Finally, there is the case for accepting when there does not yet exist a direct channel (the reconfiguration "accept" in the top right of Fig. 7.9). For this case, instead of extending the direct channel to a permitted channel, the permitted channel structure has to be created. We combine all these considerations to the following handler for the request token $permReq(u2, u1)$, i. e., for accepting a contact exchange and creating a permitted channel between two users $u1$ and $u2$:

$$\begin{aligned} handler(permReq) :=& DeleteOppositeExchange; \\ & (ExtendToPermittedChannel \parallel CreatePermittedChannel) \end{aligned}$$

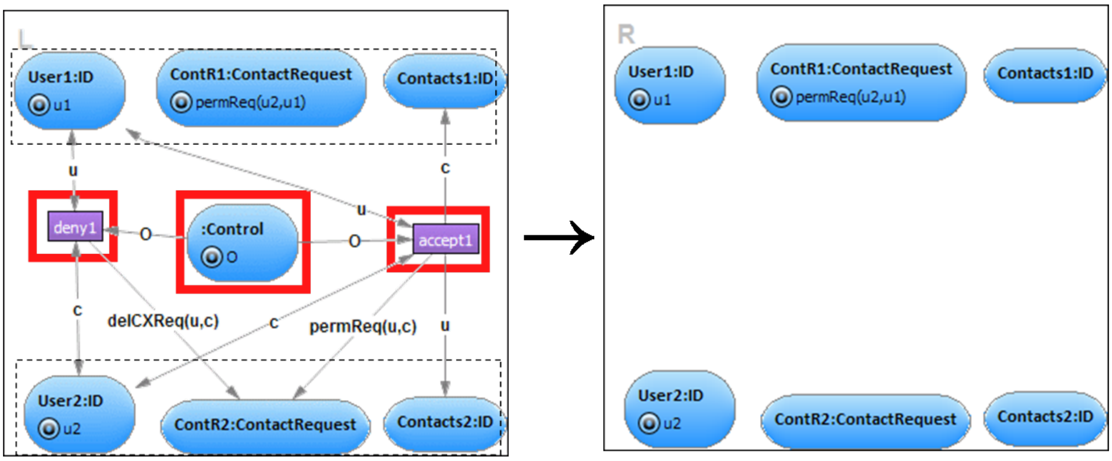


Figure 7.13.: Rule *DeleteOppositeExchange*

DeleteOppositeExchange The first handler rule shown in Fig. 7.13 deletes a contact exchange structure if there exists one in the opposite direction between user $u2$ and user $u1$. In this case, the second contact exchange structure is not needed anymore because the two remaining rules of the last priority list of the handler for the *permReq* token create a permitted channel between these two users.

ExtendToPermittedChannel If there exists a direct channel between the clients of the users $u1$ and $u2$, we only need to delete the contact exchange structure and disconnect the actions for sending instant messages from the opposite user's place carrying his privacy settings. This effect is the result of applying the rule in Fig. 7.14.[5] Note that environment of the two transitions for sending messages are changed (denoted by the cyan box).

[5] Do not be confused that we "extend" a direct channel by essentially disconnecting two transitions from two places. Instead of extending the net structure by further elements, we understand the extension as related to the fact that a permitted channel supersedes a direct channel.

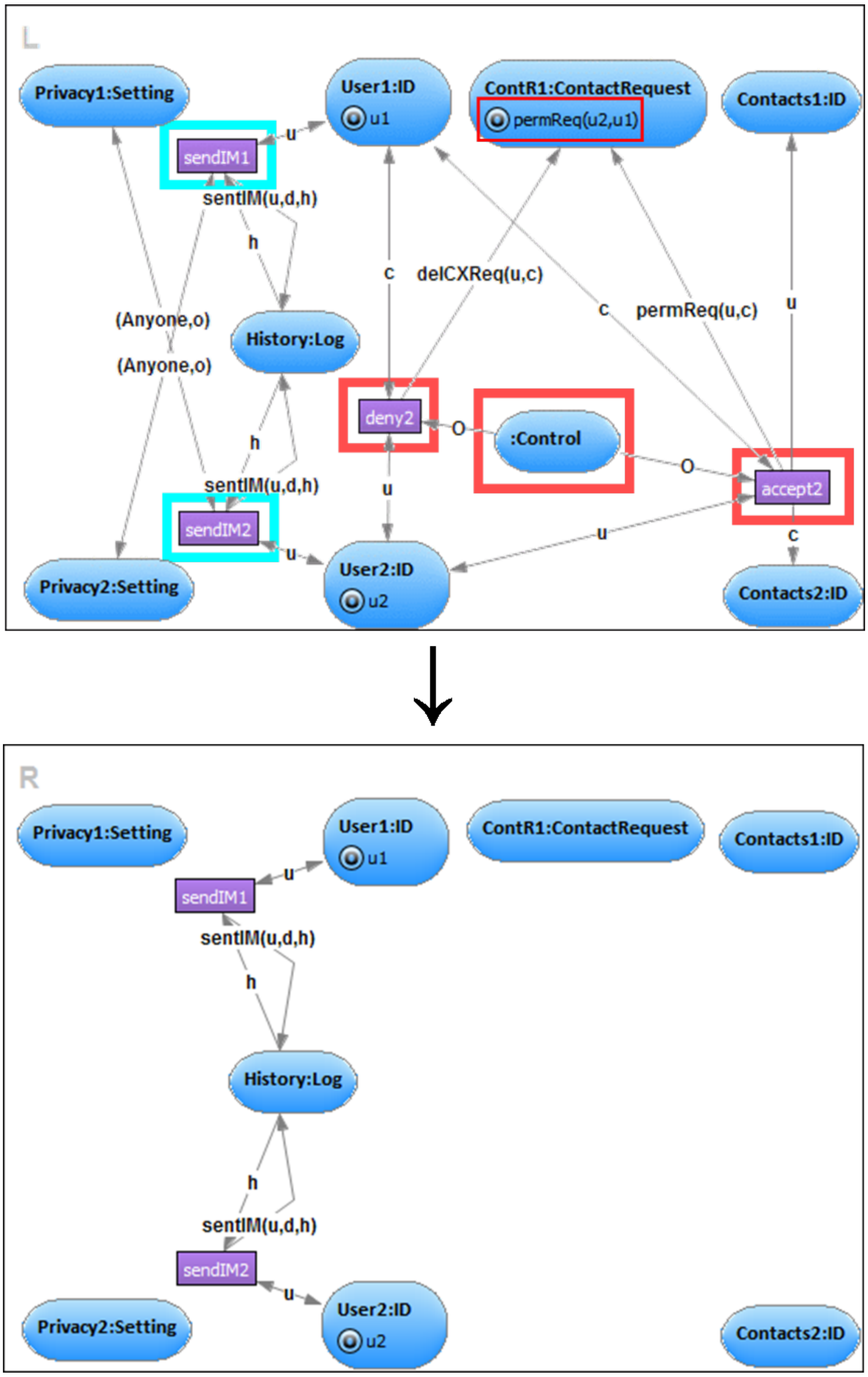


Figure 7.14.: Rule *ExtendToPermittedChannel*

CreatePermittedChannel If the contact exchange has been accepted by $u2$ without that a direct channel has been created before, the application of the rule in Fig. 7.15

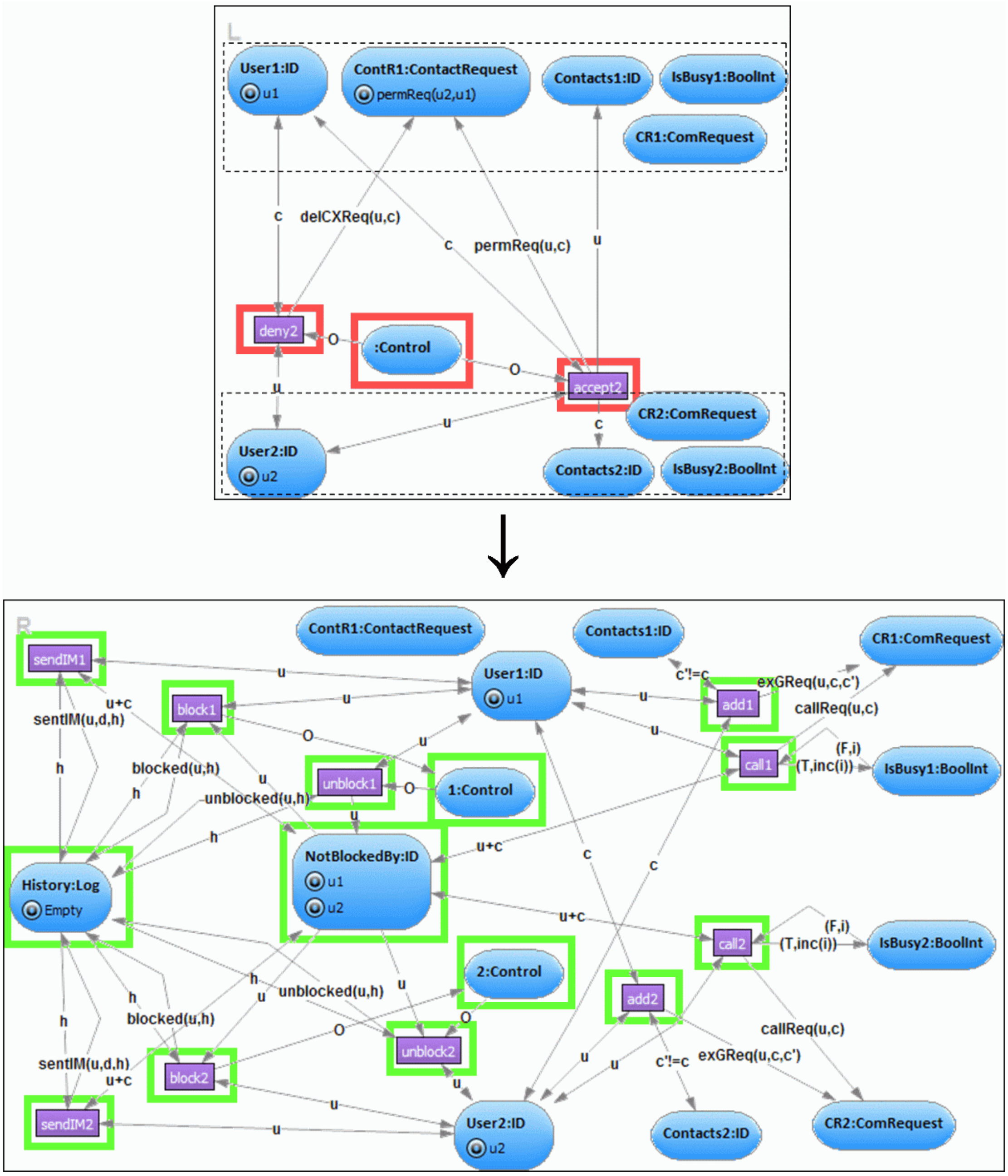


Figure 7.15.: Rule *CreatePermittedChannel*

results in a combination of the structural effects of the rules *ExtendToPermittedChannel* (see Fig. 7.14) and *RuleCreateDirectChannel* (see Fig. 7.7).

With this, we have covered all the possible reconfigurations for the actions *requestContact*, *deny*, and *accept* between all the configurations from Fig. 7.9.

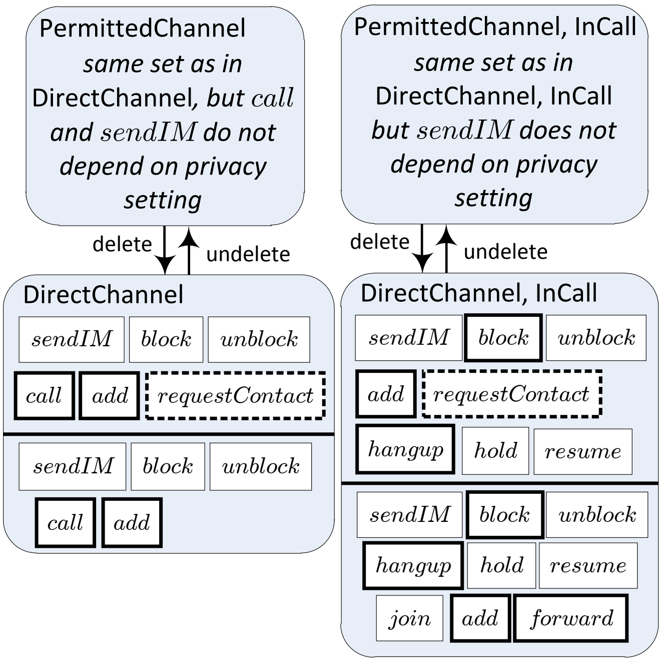


Figure 7.16.: Reconfigurations for contact management

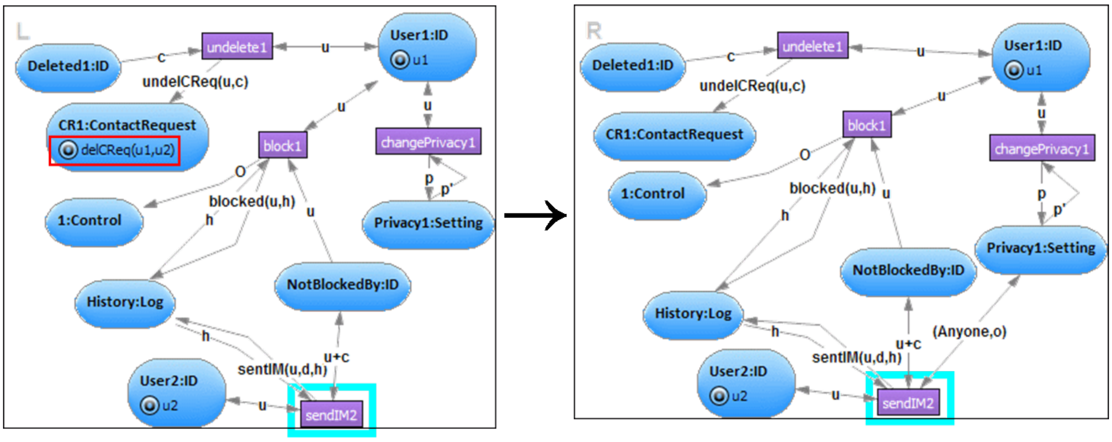


Figure 7.17.: Rule *DeleteContact*

7.2.4.3. Deleting and Undeleting Contacts

Once two users have agreed to a permitted channel between them and each has the other's identifier token on his contact list place, each of them can invoke the actions in Fig. 7.16 for first deleting and then undeleting the other's contact, which we model by the two corresponding transitions in the client core (see Fig. 7.3). Deleting and undeleting a contact affect message sending and calling in permitted channels but otherwise do not change the set of available actions.

Intentionally, we model the transition for starting a *call* in direct and permitted chan-

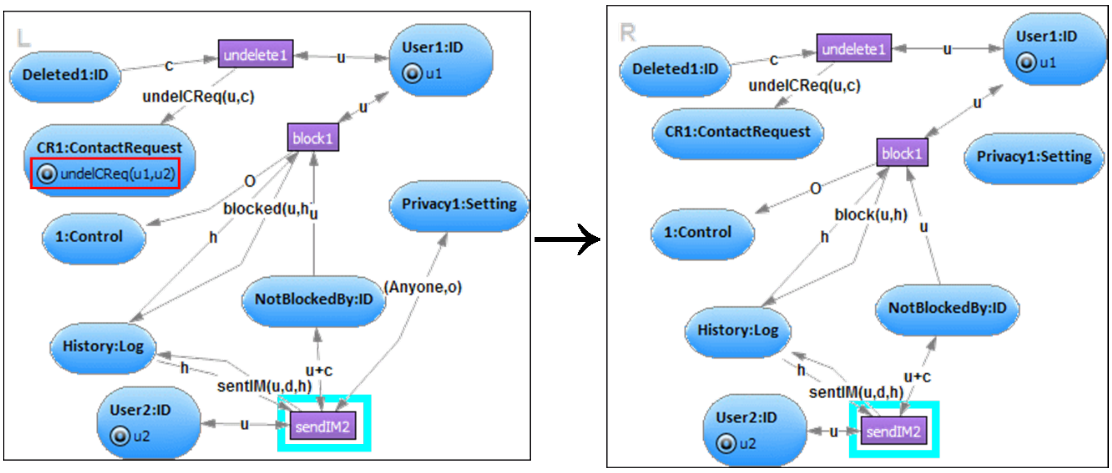


Figure 7.18.: Rule *UndeleteContact*

nels to depend on the caller being neither blocked nor busy, only. As discussed in Sect. 5.1.2.3, in Sect. 7.2.4.4 we care about whether the callee allows this incoming call by formulating an appropriate reconfiguration handler for the call request that considers all exceptional cases. For the delete and undelete request handler, it remains to reconfigure the channel transition for sending messages, which leads to the following handler definitions.

$$handler(delCReq) := DeleteContact$$

$$handler(undelCReq) := UndeleteContact$$

The rule *DeleteContact* in Fig. 7.17 matches the client core part with the delete request token $delCReq(u1, u2)$ of the deleting user $u1$ and the permitted channel (to user $u2$) to be manipulated. Because $u1$ deletes $u2$ from his contact list, from now on $u2$ may call $u1$ only if its privacy settings allow incoming calls from all users. The rule realizes this by reconfiguring the transition *sendIM2* of user $u2$ so that it depends on the corresponding privacy setting on place `Privacy1` of $u1$. Rule *UndeleteContact* in Fig. 7.18 reverses this effect if $u1$ later creates a token $undelCReq(u1, u2)$ by firing his transition *undelete* for the same user $u2$.

7.2.4.4. Initiating and Hanging up Calls

If a direct channel exists between two clients, either of them can request a call to the other one, hang up an ongoing call, and block the other user as illustrated in Fig. 7.19. We can see in Fig. 7.19 that the action set differences for these configuration changes are the same for all three vertical cases and in fact there is no dependency from these actions and their effects to the contact exchange status between the two regarded users. This indicates that we do not have to regard this in the handler for all three vertical reconfigurations of *call*, as well, and similarly for *block* and *hangup*.

For starting a call to the other user $u2$, a user $u1$ can fire his corresponding transition *call1* (or the other way round with *call2*) that has been created by the rule

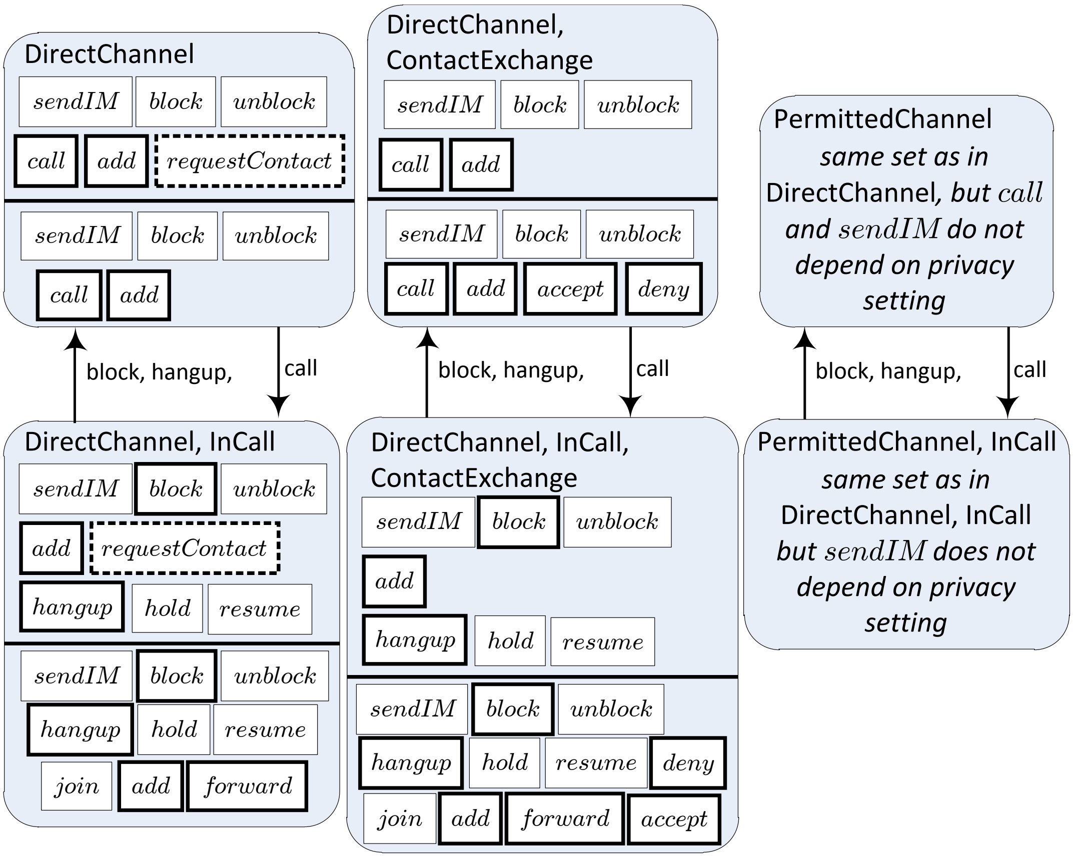


Figure 7.19.: Reconfigurations for direct calls

CreatePermittedChannel in Fig. 7.15 and which produces a request token $callReq(u1, u2)$. The attempt to call another user can simply be successful or fail, which results in the following simple request handler:

$$handler(callReq) := DiscardCallDirect \parallel CallDirect$$

DiscardCallDirect As before, the discarding rule for exceptional failing cases is the first in the handler expression. It deletes the request token and decreases the call counter of $u1$. A call attempt fails if the called client either is offline or does not allow to be called by the particular calling user. In Skype, the latter can have two causes: The first is that $u2$ has blocked $u1$. The second is that $u2$ has set its privacy settings to “only contacts may call” and does not have $u1$ on his contact list, which is the case if either there exists only a nonpermitted channel (no contact exchange has been performed) or if $u2$ has deleted $u1$ from his contact list.[6] In this particular situation of a call attempt

[6] Without the notion of “inhibitor arcs” in the Petri net formalism, the absence of a token cannot be formulated as a condition for the firing of a transition. We choose to treat this exception by reconfiguration rather than by complicating the firing behavior formally. But as we discuss in

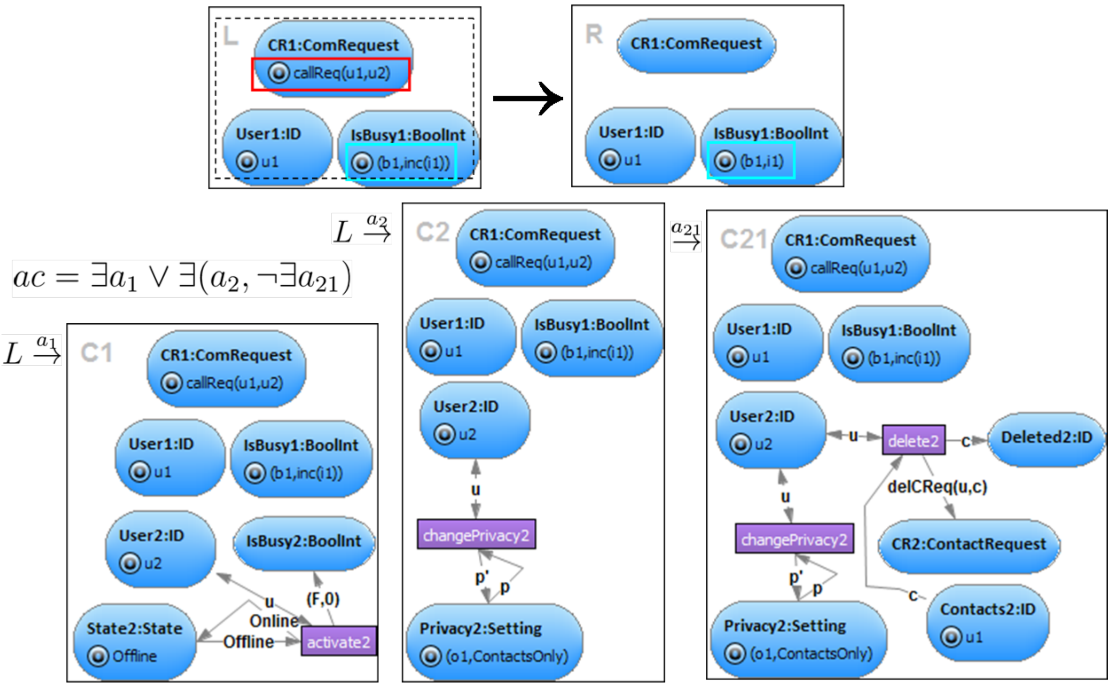


Figure 7.20.: Rule *DiscardCallDirect* with application condition *ac*

by $u1$, we can exclude the first case because $u1$ being blocked by $u2$ would not allow $u1$ to fire his calling transition (because of the required tokens on the place `NotBlockedBy` in the channel) in the first place.

The other exceptions are recognized by the application condition ac of rule *DiscardCallDirect* in Fig. 7.20. This condition is satisfied if $u2$'s client is offline (then $\exists a_1$ is satisfied) or if $u2$ has set his privacy settings as said above, which satisfies $\exists a_2$, and $u1$ does not appear on $u2$'s contact place, which satisfies the nested condition $\neg\exists a_{21}$ and hence the whole second clause $\exists(a_2, \neg\exists a_{21})$.

CallDirect The rule *CallDirect* in Fig. 7.21 manipulates a direct channel so that it represents an ongoing call. First, the transitions for requesting a call and the request token are deleted. On this channel's history place, the started call is logged in the history token. The channel's transitions for blocking the opposing user are reconfigured such that they additionally create a request token for hanging up this call, which we treat next. The same request token is produced by the new transitions *hangup1* and *hangup2*.

The called user $u2$ can *join* the call (until then we assume that his client is ringing or announcing the incoming call in some other way), which sets his busy status and the token on the place `HasJoined` to true. After that, each user can put this call on *hold* so that he can participate in another call and later, when he is idle again, he can resume this call if neither $u1$ nor $u2$ have decided to hangup this call before. The Boolean token

Sect. 5.1.2.3 for call attempts, this does not have to be necessarily a disadvantage or inaccurate modeling.

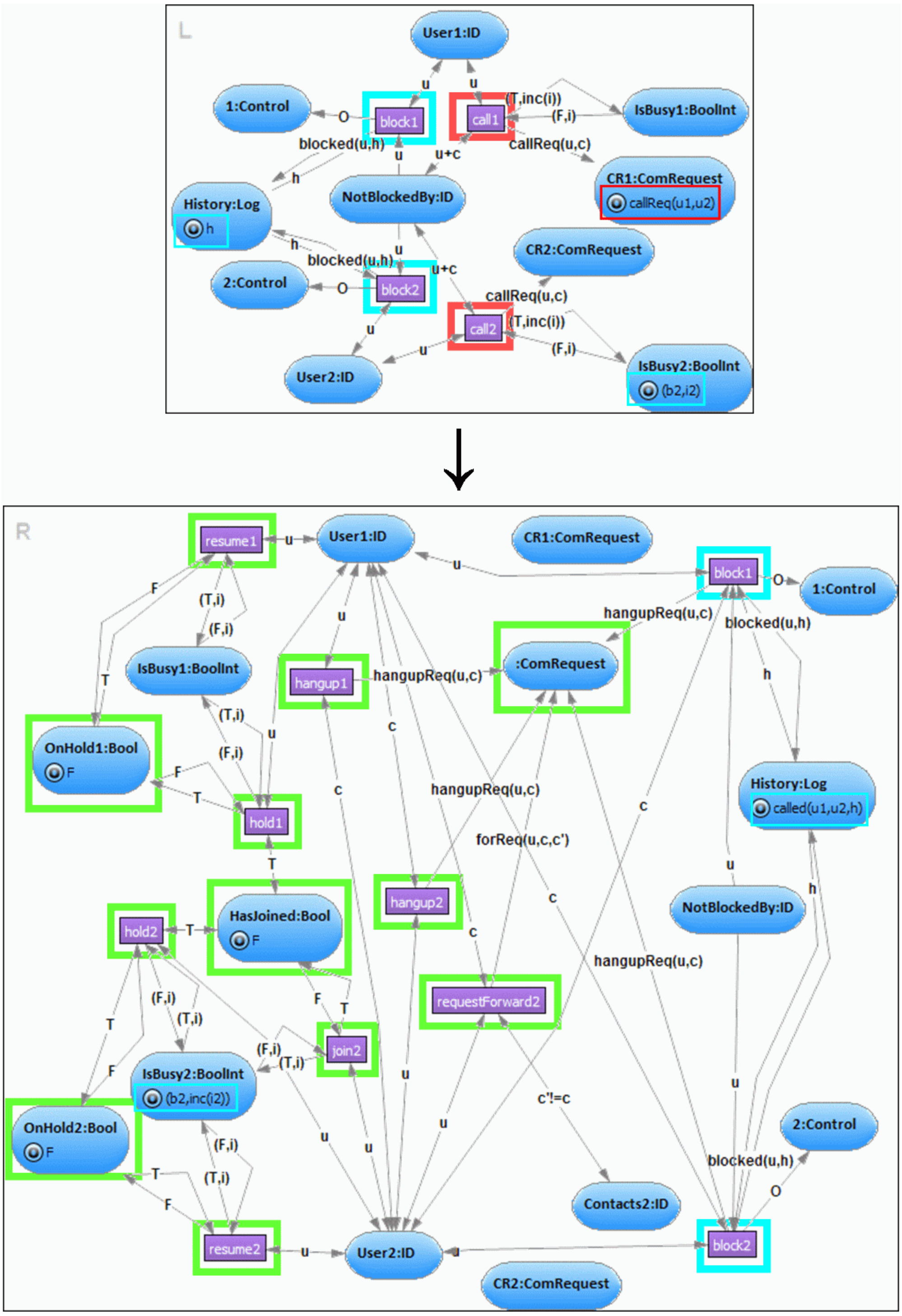


Figure 7.21.: Rule *CallDirect*

is busy	put on hold	new busy state
T	T	T
T	F	F
F	T	F
(F	F	F)

Table 7.2.: Determining the new busy state after hanging up a call

values on the places `OnHold1` and `OnHold2` ensure that this call can be put on hold (or be resumed) consecutively at most once by each participant and help to decide whether a participant of this call can still "hear" what the other says (cf. Sect. 7.2.6).

The second user can fire *requestForward2* to try to forward the call to a third user from his contact place, for which we define a request handler in Sect. 7.2.4.5.

The request token $\mathit{hangupReq}(u, c)$ (produced by the transitions for hanging up and for blocking) is handled by the following expression.

$$\mathit{handler}(\mathit{hangupReq}) := \mathit{HangupDirectCall}$$

HangupDirectCall This rule in Fig. 7.22 is basically the inverse of rule *CallDirect* with some modifications the tokens. It decreases the call counters and calculates the new busy status of each participant after hanging up by considering the current busy status and if the call to be ended has been put on hold by this user.

For determining the new busy state of each user, consider Table 7.2. Clearly, if a user has put this call on hold and is busy (in another call), he is still busy in the other call after this call has ended (first row). If he is busy and has not put the call on hold, the call to be ended is the currently active one and the user is idle after hanging up (second row). If the user is not busy at all when this call is ended, he neither is afterward (third row). The case in the fourth row cannot occur (if the call is not put on hold, the user must be busy), and the other cases justify the new busy state to be modeled as the conjunction of these two Boolean values.

7.2.4.5. Forwarding Calls

If the called $u2$ user in a direct call (initiated by $u1$) produces a request token $\mathit{forReq}(u2, u1, u3)$ in order to forward $u1$ to $u3$, the following handler takes care of the forwarding.

$$\begin{aligned}\mathit{handler}(\mathit{forReq}) :=& \mathit{PrepareDirectChannel};\\ & \mathit{ForwardHangup}; (\mathit{DiscardForwardCall} \parallel \mathit{ForwardCall})\end{aligned}$$

Forwarding a call consists of the actions of terminating the current call and starting a new call between $u1$ and $u3$. We already described handlers and rules for these manipulations but we cannot directly use them for this handler because they presume a

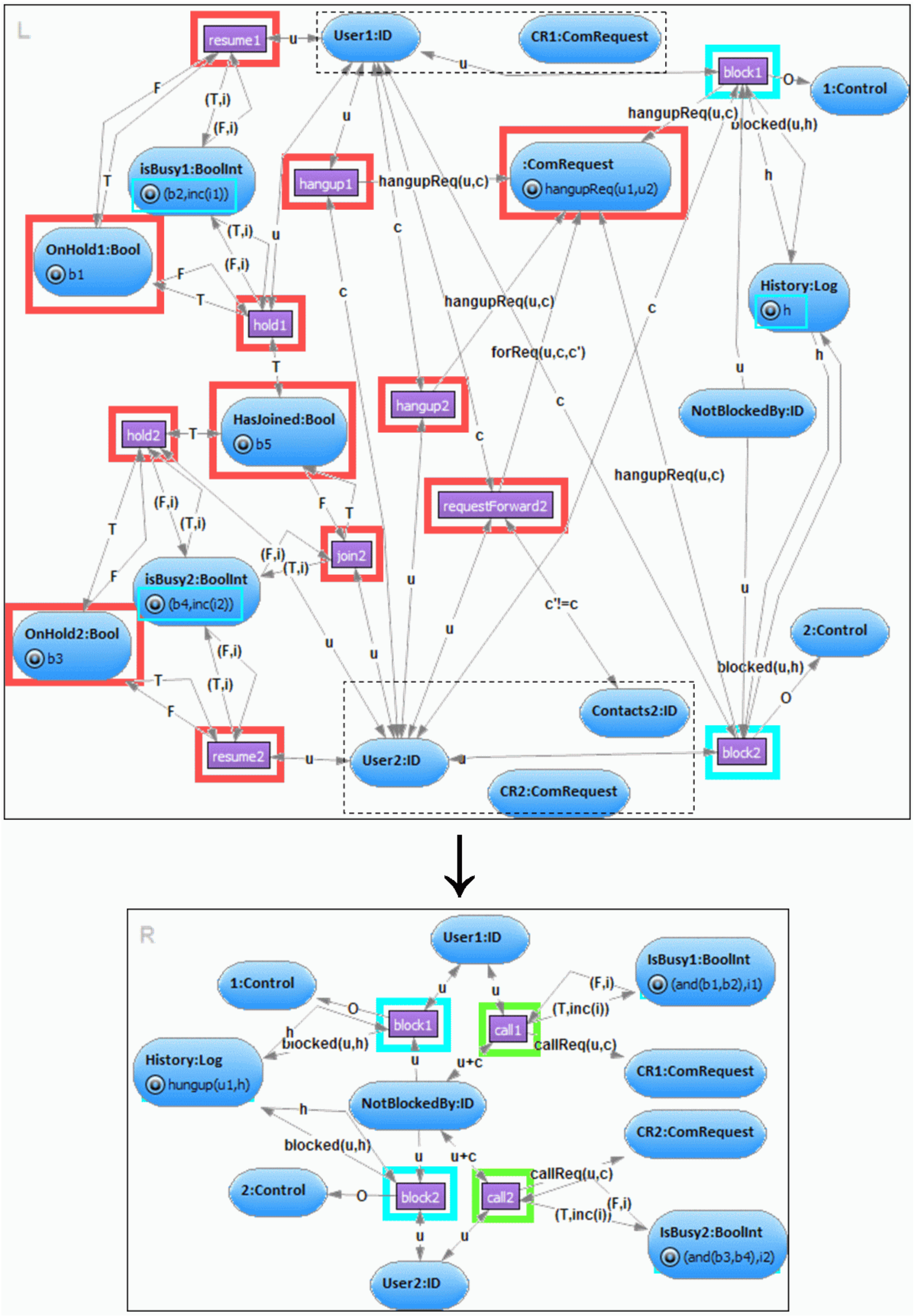


Figure 7.22.: Rule *HangupDirectCall*

different request token. Nevertheless, the following rules are very similar to the previous rules. We show these explicitly, but for later rules we may refer to previous rules and application conditions if the differences are not too extensive.

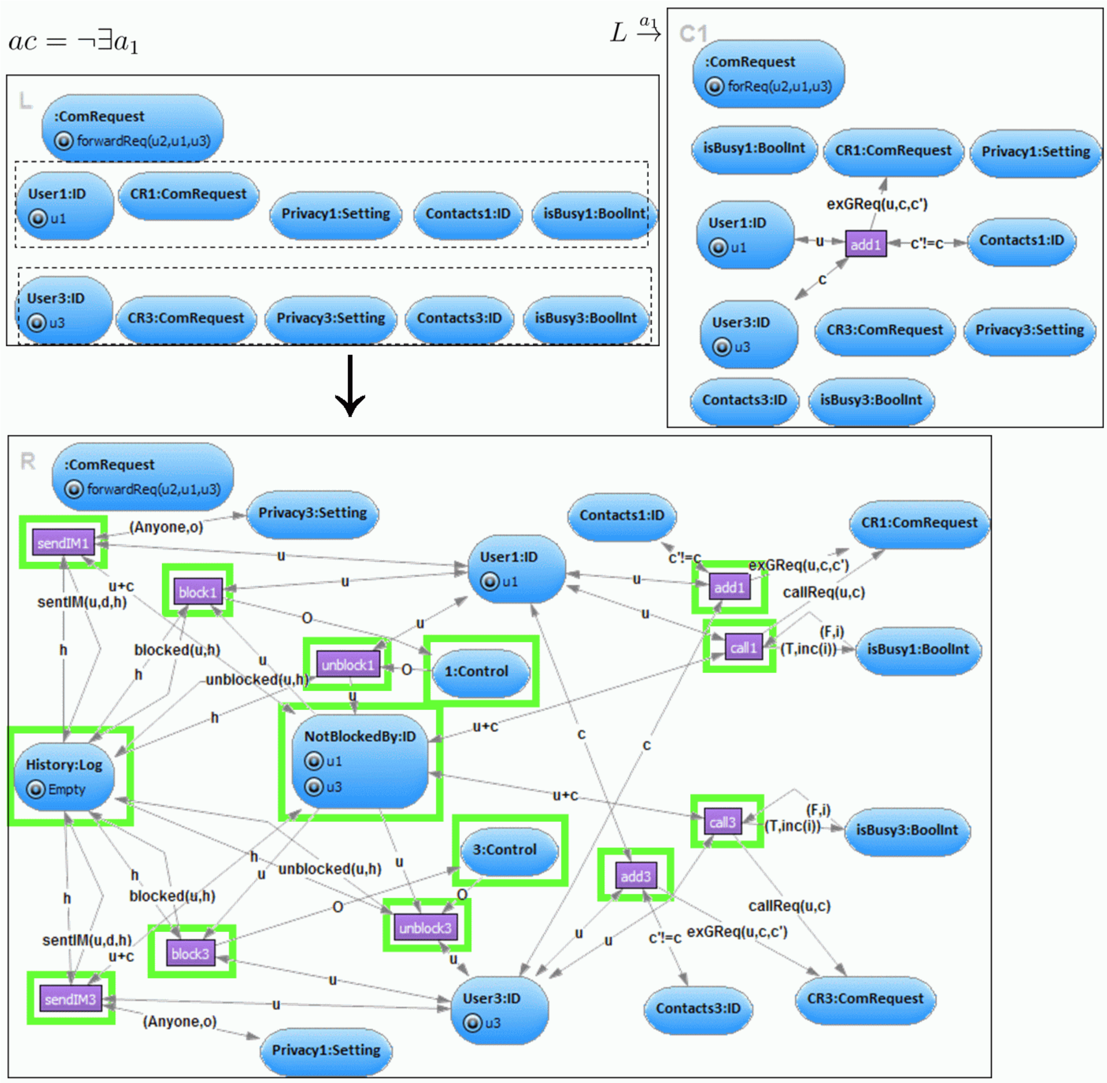


Figure 7.23.: Rule *PrepareDirectChannel* with application condition *ac*

PrepareDirectChannel There may already exist a direct channel between $u1$ and $u3$, but in order to ensure this, this rule in Fig. 7.23 creates a direct channel between these users exactly as rule *CreateDirectChannel* in Fig. 7.7 would do if $u1$ requested it. The rule is only applied if there does not yet exist a channel between $u1$ and $u3$, which is ensured by this rule's application condition, which is satisfied if there does not yet exist the structure in $C1$ indicating an existing channel (cf. the LHS of rule *DiscardDirectReqExisting* in Fig. 7.8). Note that the request token is not deleted by this rule.

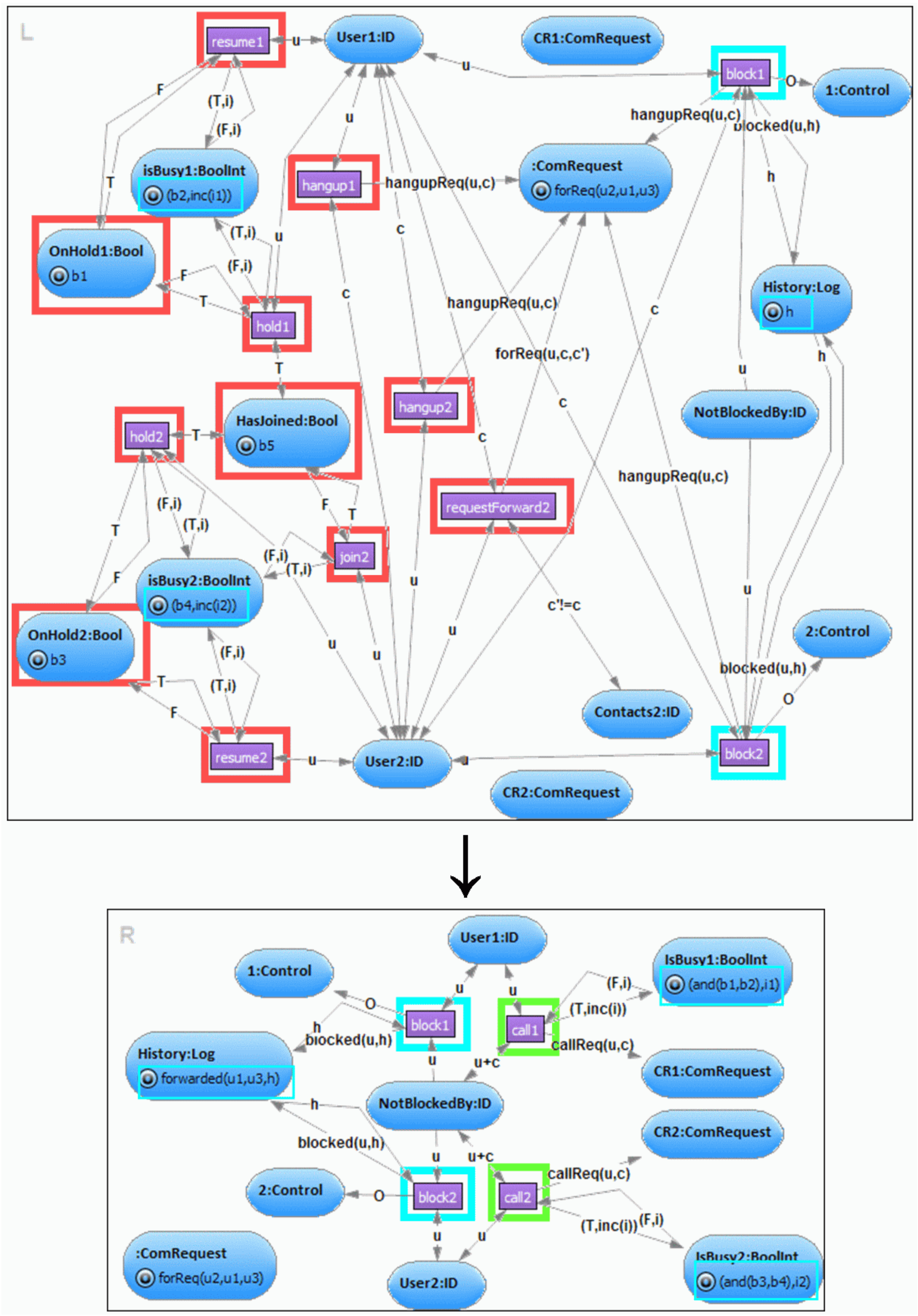


Figure 7.24.: Rule *ForwardHangup*

ForwardHangup This rule in Fig. 7.24 has the same effect as rule *HangupCall* in Fig. 7.22 and terminates the call between $u1$ and $u2$. As a slight variance, it logs to the history that the call has been forwarded instead of simply been hung up and the place carrying the forward request token is not deleted but preserved for the remaining rules of this handler.

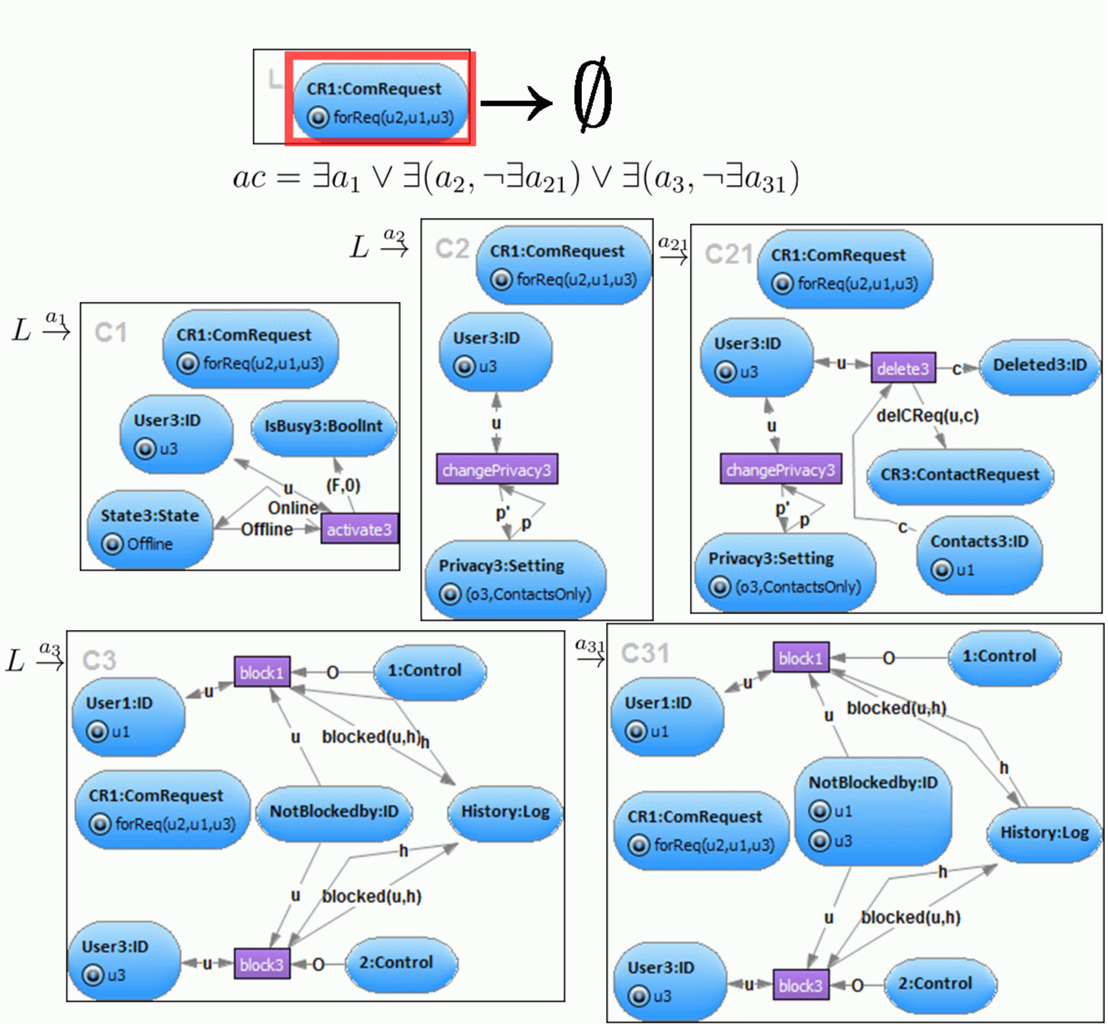


Figure 7.25.: Rule *DiscardForwardCall* with application condition *ac*

DiscardForwardCall Forwarding fails if $u1$ is not able or allowed to call $u3$ directly. The application condition of this rule in Fig. 7.25 checks if there is a reason that $u1$ cannot call $u3$. The rule *DiscardForwardCall* is applicable in these cases and discards the request token together with the place `CR1`, which is a leftover from the direct call structure between $u1$ and $u2$. The first two parts of the application condition are the same that we treated for rule *DiscardCallDirect* in Fig. 7.20, which check if $u3$ is offline (then $\exists a_1$ is satisfied) and if $u3$ has set his privacy settings to allow only incoming calls from his contacts and $u1$ is not on $u3$'s contact list (then $\exists(a_2, \neg\exists a_{21})$ is satisfied). The condition $\exists(a_3, \neg\exists a_{31})$ is satisfied if there exists a direct channel between the clients of $u1$ and $u3$ and at least one of them has blocked the other one. Note that the call counter in the client core of $u1$ does not have to be decreased in this case, because he did not request this call himself by firing the calling transition, which would otherwise have increased the counter as for regular calls.

ForwardCall Finally, if rule *DiscardForwardCall* is not applicable and $u3$ can be called, the rule *ForwardCall* in Fig. 7.26 establishes the forwarded call by causing the same effect on the direct channel between $u1$ and $u3$ (which may have been prepared just before by rule *PrepareDirectChannel*) as rule *CallDirect* in Fig. 7.21 would do.

7.2.5. Groups and Conferences

In this section, we describe the remaining user actions with reconfigurations from Fig. 7.2, which regard creating and managing groups and conferences.

7.2.5.1. Requesting a Group

If a user has fired the transition *requestGroup* to create a group, a request token $\mathit{groupReq}(u, c, c')$ has been produced, where u is the name of the user who performs the creating action, and c and c' are names from u's contact list. This request is handled by the following handler expression:

$$\mathit{handler}(\mathit{groupReq}) := \mathit{CreateGroup}$$

CreateGroup This rule in Fig. 7.27 creates a group channel between the clients of the users $u1$, $u2$, and $u3$ such that $u1$ becomes the owner of the group (the only member who can *remove* other members from the group). This channel is similar to the direct channels that are created by the rule *CreateDirectChannel* in Fig. 7.7. The group members can send instant messages to the other group members (via the transition *sendIM1*, *sendIM2*, and *sendIM3*), initiate a conference (via *call1* etc.), and add other users from their contact lists (with *add1* etc.). The user $u1$, as the owner of the group, can *remove* other members. All other members can *leave* the group. The currently active members of this group are indicated by the *ID* tokens on the place `Members`. Note that all actions in groups demand that the corresponding user's name lie on the place `Members`, i. e., he is currently an active member. The transitions *leave* and *remove* consume a name token from this place and trigger a reconfiguration (see Sect. 7.2.5.6). Every member has its own history for the messages that have been sent in this group while he was an active member. In contrast to the direct channel, sending instant messages in groups triggers a reconfiguration by producing request tokens $\mathit{transGReq}(u, d)$, because group messages require multicasting to all active members of this group.

7.2.5.2. Sending Group Messages

Sending messages in groups is always possible for active group members. The sent message has to be distributed to all other active members, i. e., to be appended to their history log. Because the number of active group members can change, the following

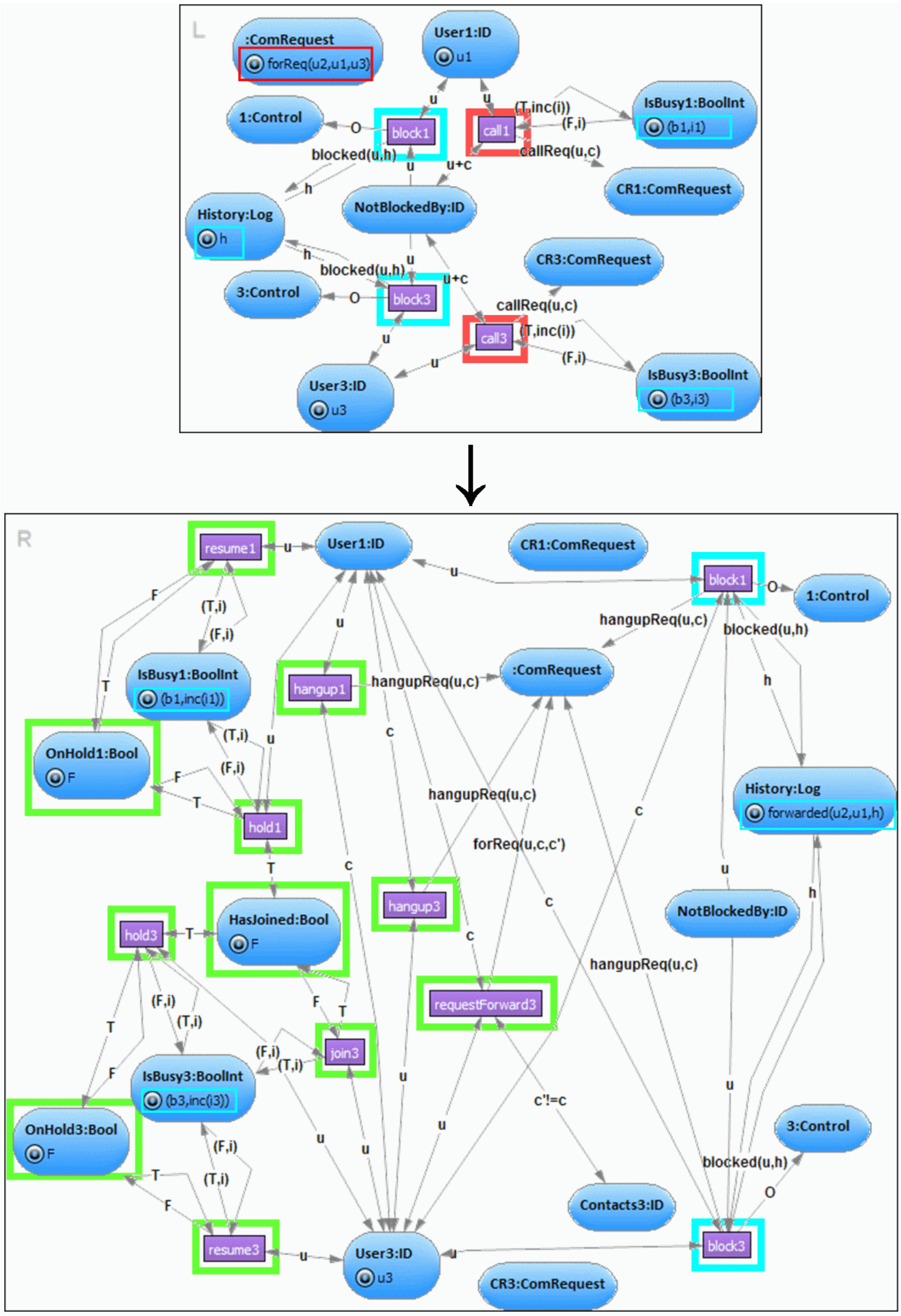


Figure 7.26.: Rule *ForwardCall*

handler for sending messages in groups consist of an interaction scheme that realizes

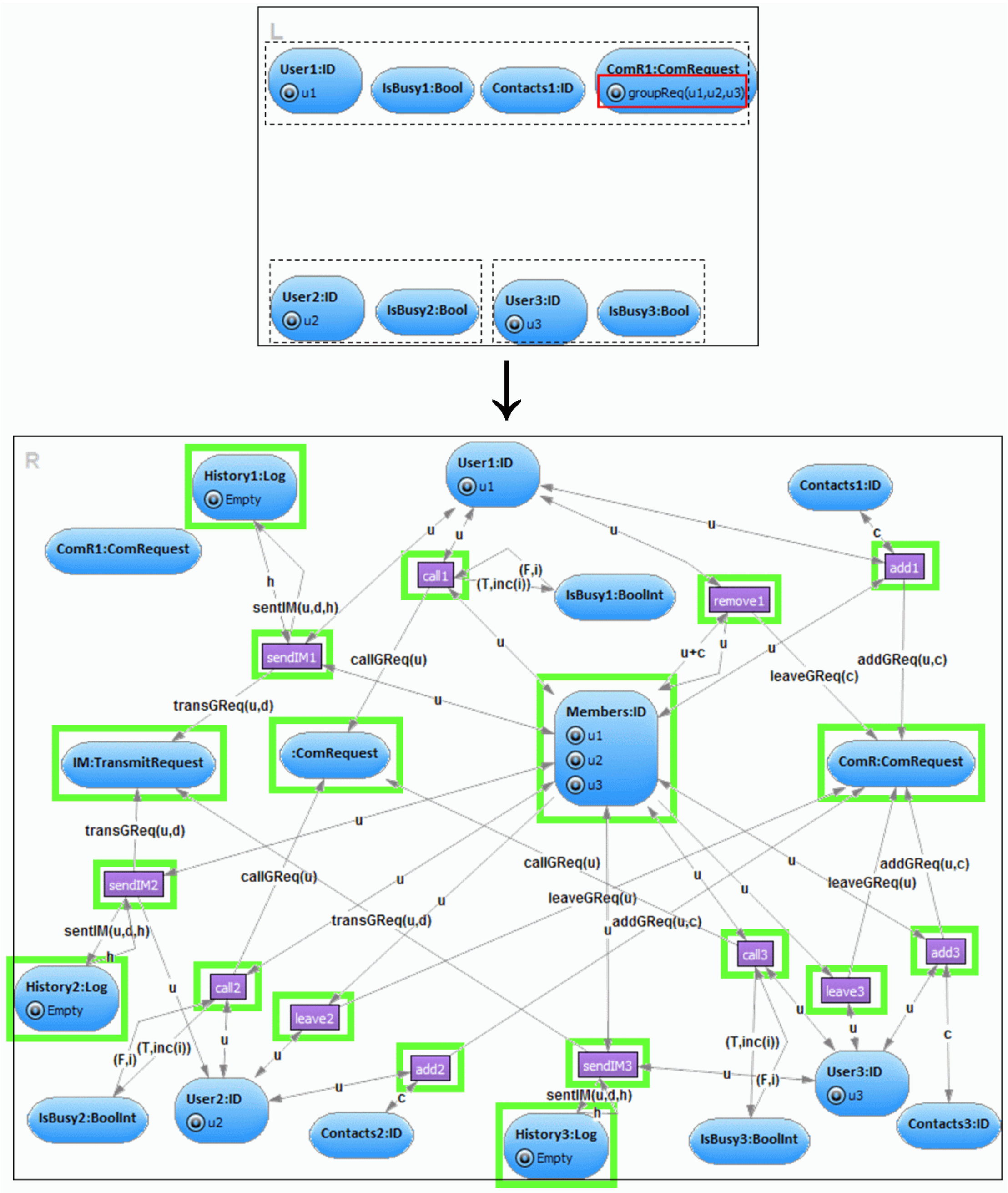


Figure 7.27.: Rule *CreateGroup*

selective multicasting of the sent data (cf. Sects. 3.2.5.1 and 5.2.2).

$$handler(transGReq) := SendGroupMessage$$

SendGroupMessage In Fig. 7.28, the simple interaction scheme for sending group messages is shown in compact notation. The kernel rule matches the group member place and

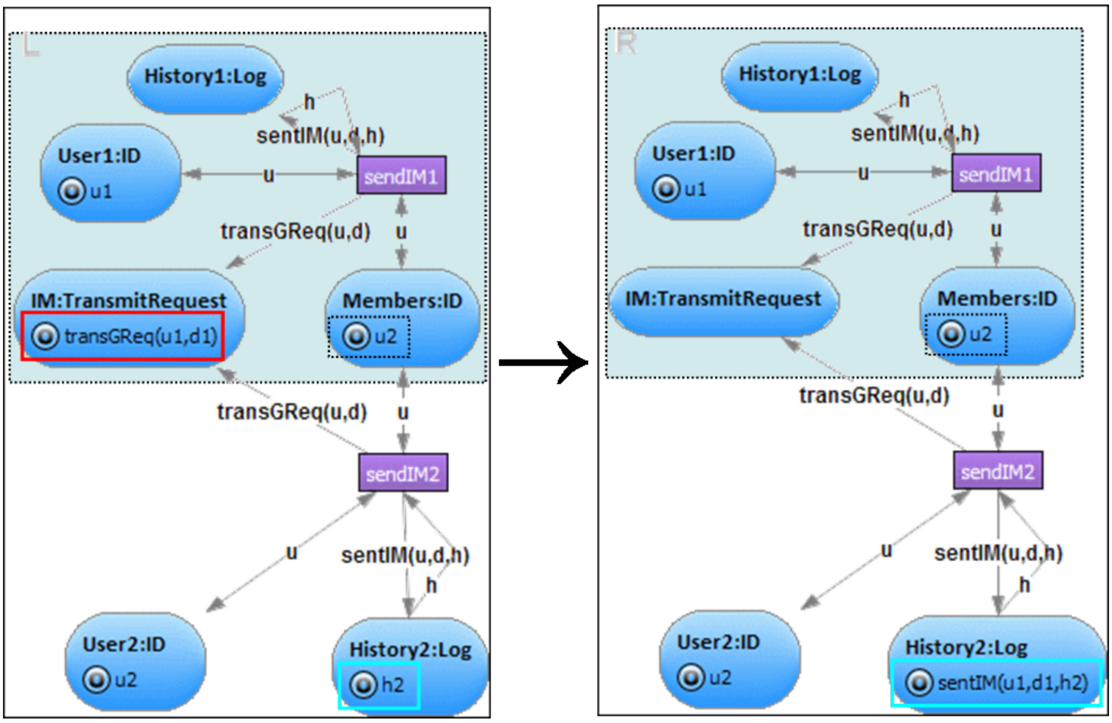


Figure 7.28.: Interaction scheme *SendGroupMessage*

the place with the *TransmitRequest* token $\mathit{transGReq}(u1, d1)$. The transition *sendIM1* ensures that the matched places belong to the same group. A matching of the multi rule selects a user $u2$ whose name can be found on the place `Members`. Note that the token $u2$ (surrounded by a dotted box) is not a part of the kernel rule and is matched by the multi rule, only! The effect of the multi rule is that the message is appended to the history $h2$ of $u2$. The kernel rule just deletes the *TransmitRequest* token. Maximal matching of the multi rule distributes $d1$ to all active members in this group, i. e., the users whose names are present on the place `Members`.

7.2.5.3. Initiating Conferences

If an active member u of a group starts a conference by firing his corresponding transition *call* in the group structure, the following expression handles the reconfiguration that is triggered by the produced request token $\mathit{callGReq}(u)$.

$$\mathit{handler}(\mathit{callGReq}) := \mathit{InitConference}; \mathit{ParticipateNewConference}$$

InitConference As a first step, the interaction scheme shown in Fig. 7.29 initializes the new conference. Its kernel rule matches the user who requested to start a conference, deletes his *call1* transition and creates transitions for the actions of a conference's host (cf. the "Host" compartment in the configuration "GroupConference" in Fig. 7.2). A new shared conference place `GroupCall` keeps track of the group members who have joined

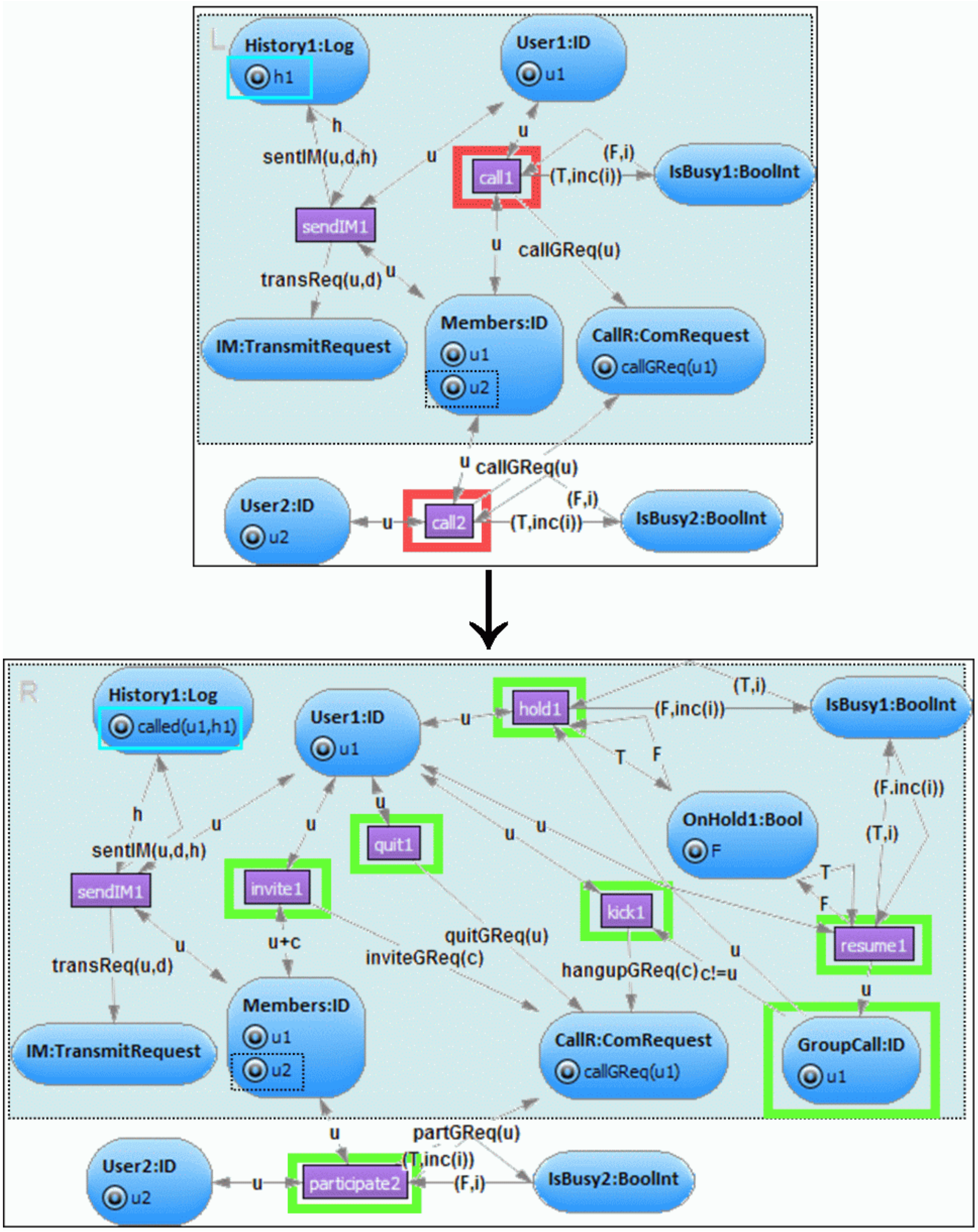
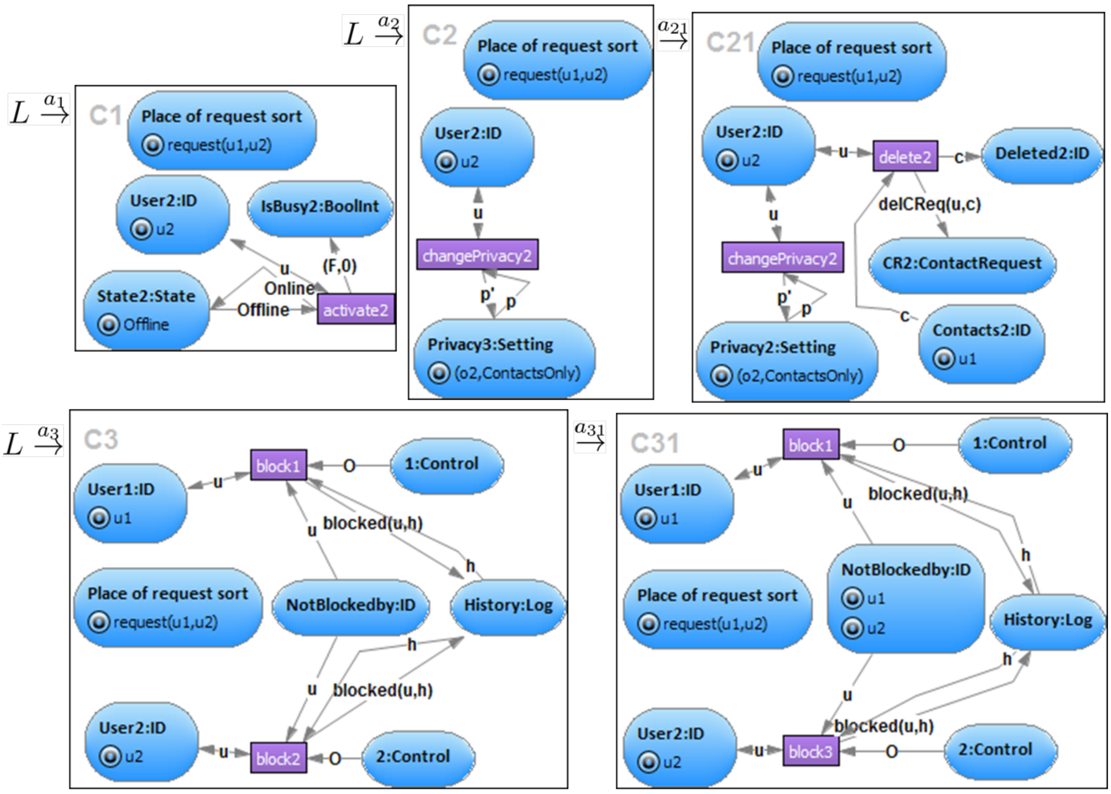


Figure 7.29.: Interaction scheme *InitConference*

the conference. The functionality of the transitions *hold1* and *resume1* is the same as for the transitions with the same name in the rules for calls in direct channels (see rule *CallDirect* in Fig. 7.21). The transitions *invite1*, *kick1*, and *quit1* produce corresponding token requests.

The scheme's multi rule matches another active member by the identifier token *u2* on the group place `Member` and replaces the transition *call2* with *participate2*. In a group, there can only be one ongoing conference and active members can decide to participate in this conference. Note that participating in a conference does not necessarily mean that the participating user has joined the conference. Participating in a conference can also mean that a user is trying to get into a conference or that the host has invited him, which the invited user can deny by hanging up. If a client is “ringing” w. r. t. a conference, then he is already considered to be participating (passively, still without getting the messages that are communicated in the conference).



$$ac = \exists a_1 \vee \exists(a_2, \neg\exists a_{21}) \vee \exists(a_3, \neg\exists a_{31})$$

$$\neg ac = \neg\exists a_1 \wedge \forall(a_2, \exists a_{21}) \wedge \forall(a_3, \exists a_{31})$$

Figure 7.30.: Abstract application condition template ac: $u1$ cannot call $u2$

Abstract application condition template for “$u1$ cannot call $u2$” There is a general template of application conditions for expressing whether a user can call another one or not, which we explain in the following. Although this is not a formal construction, we employ this adapting and reusing of application conditions if possible for further rules in the following without explicitly giving the corresponding application conditions. For the rule *DiscardForwardCall* in Fig. 7.25, we used the condition $\exists(a_3, \neg\exists a_{31})$ to express the condition that one user has blocked another one for discarding a forward request. Looking back, also the explicit application conditions of the rules *DiscardForwardCall* in Fig. 7.25 and *DiscardCallDirect* in Fig. 7.20 could be derived from this template.

Consider the application condition ac in Fig. 7.30 over the given schematic morphisms for some rule with LHS L. We assume that in L there is some place of a request sort carrying a request token that determines the names of $u1$ and $u2$. Alternatively, L also may contain one or both of the identifier places `User1` or `User2` of the client core of $u1$ and $u2$. The condition ac is satisfied if $u1$ cannot call $u2$, which is expressed by the disjunction of conditions that express a reason for this, each:

$\exists a_1$ This is satisfied if and only if the user $u2$ is offline so that the token *Offline* on $u2$'s *State* place can match the platform net.

$\exists(a_2, \neg\exists a_{21})$ This is satisfied if and only if $u1$ cannot call $u2$ due to his privacy settings. If $u2$ has restricted his privacy settings to incoming calls from contacts, only, then $\exists a_2$ is satisfied. If moreover $u1$ is not on $u2$'s contact list, then also the nested condition $\neg\exists a_{21}$ is satisfied and therefore $\exists(a_2, \neg\exists a_{21})$ is satisfied. If on the other hand $\exists(a_2, \neg\exists a_{21})$ is not satisfied, then $u2$ has $u1$ on his contact list or he has set his privacy settings to allow all incoming calls.

$\exists(a_3, \neg\exists a_{31})$ This is satisfied if and only if either $u1$ or $u2$ has blocked the other one. If $\exists a_3$ is satisfied, then a direct channel exists between $u1$ and $u2$. If moreover either $u1$ or $u2$ has blocked the other one in this channel, then $\neg\exists a_{31}$ is satisfied because the tokens on place `NotBlockedBy` cannot match. If on the other hand $\exists(a_3, \neg\exists a_{31})$ is not satisfied, then either nobody is blocked in the channel between $u1$ and $u2$ or there does no exists such a channel at all, i. e., $u1$ and $u2$ do not block each other.

The given nets $C1, C2, C21, C3, C31$ are minimal templates that we can reuse for any LHS L by adding the elements from the particular L to those that do not yet occur. With this, we get proper condition morphisms mapping identically on the parts of the LHS and like the shown schematic morphism on the additional parts.

If we negate an instantiated application condition ac for a particular LHS L, we get

$$\begin{aligned}\neg ac &= \neg\left(\exists a_1 \vee \exists(a_2, \neg\exists a_{21}) \vee \exists(a_3, \neg\exists a_{31})\right)\\ &= \neg\exists a_1 \wedge \neg\exists(a_2, \neg\exists a_{21}) \wedge \neg\exists(a_3, \neg\exists a_{31})\\ &= \neg\exists a_1 \wedge \forall(a_2, \neg\neg\exists a_{21}) \wedge \forall(a_3, \neg\neg\exists a_{31})\\ &= \neg\exists a_1 \wedge \forall(a_2, \exists a_{21}) \wedge \forall(a_3, \exists a_{31})\end{aligned}$$

as shown in Fig. 7.30, which expresses the condition that $u1$ is able to call $u2$.

The construction of the condition ac that $u1$ cannot call $u2$ is especially useful for discarding rules, which need to be applicable if a calling action is not possible. The construction of $\neg ac$ is useful especially for multi rules of interaction schemes such as the following.

ParticipateNewConference This interaction scheme in Fig. 7.31 reconfigures the conference that has been initialized by the interaction scheme *InitConference*, such that all active members of this group who are not blocked by the host and who do not block the host participate in the new conference. The kernel rule just deletes the request token.

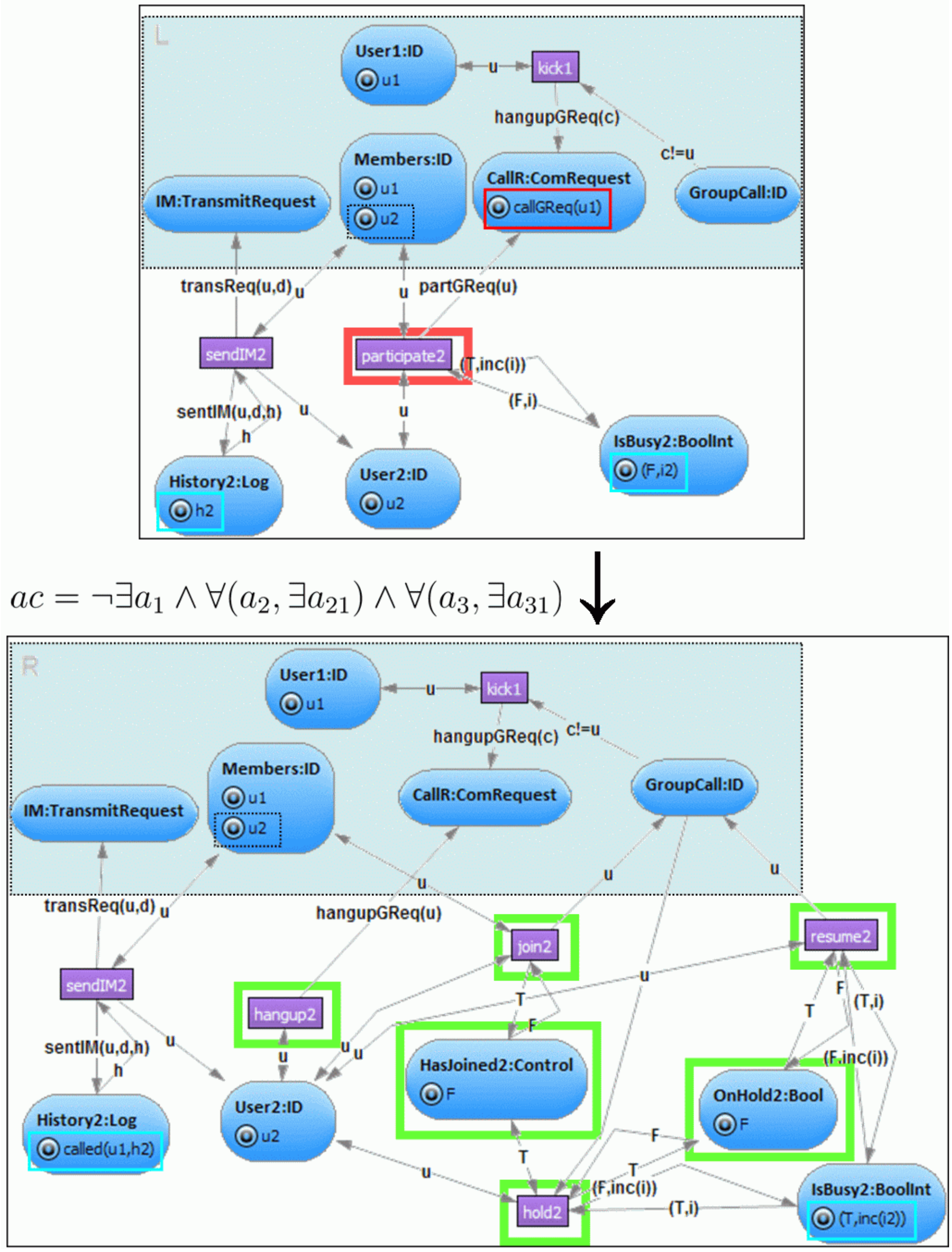


Figure 7.31.: Interaction scheme *ParticipateNewConference* with application condition *ac* for its multi rule

The multi rule matches an active group member $u2$ who is online and not busy in another call (because of the tokens *Online* and *(F,i2)*). The transition *participate* is deleted and transitions for acting in the conference call are created. The new transitions *join2*, *hold2*, and *resume2* work as in direct calls (cf. Sect. 7.2.4.4). The transition *hangup2* produces a special request token for hanging up in conferences. To ensure that the matched user

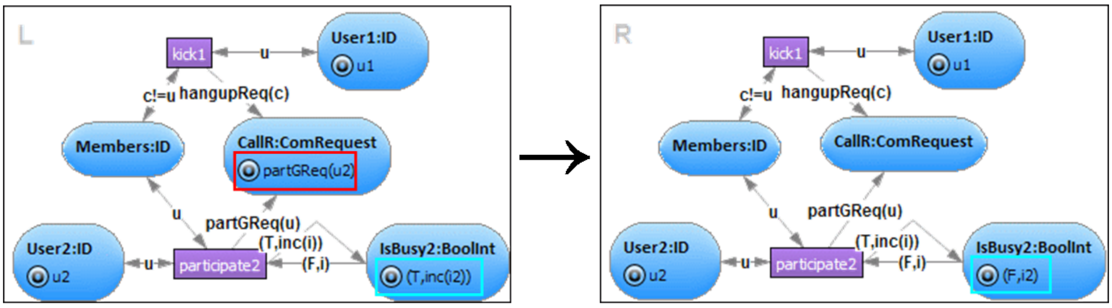
$u2$ in the multi rule of *ParticipateNewConference* is online, not blocked by $u1$ or vice versa, and allowing incoming calls from $u1$, we instantiate the negated template application condition from Fig. 7.30 appropriately for the multi rule. This is possible because the LHS contains the identifier places for the users $u1$ and $u2$. The instantiation yields the condition $ac = \neg\exists a_1 \wedge \forall(a_2, \exists a_{21}) \wedge \forall(a_3, \exists a_{31})$ for *ParticipateNewConference*.

7.2.5.4. Managing Conferences

During a running conference, active nonparticipating members can participate or be invited by the host if the host is allowed to call them (cf. the template application conditions in Fig. 7.30). Participating members can be kicked from the conference by the host or can hangup, which changes them to nonparticipating members.

If a nonparticipating active member $u2$ fires his transition *participate*, this produces a request token *partGReq*$(u2)$ on the group's *CallRequest* place. This request is handled by the following expression:

$$\mathit{handler}(\mathit{partGReq}) := \mathit{DiscardParticipate} \parallel \mathit{ParticipateConference}$$



$$ac = \exists a_1 \vee \exists(a_2, \neg\exists a_{21}) \vee \exists(a_3, \neg\exists a_{31})$$

Figure 7.32.: Rule *DiscardParticipate* with application condition *ac*

DiscardParticipate This rule shown in Fig. 7.32 catches the exceptional cases that an active group member cannot participate in a running conference hosted by $u1$, which is the case if the invited member is not online or if he cannot be called by the host. In this cases, this rule deletes the request token and decreases the call counter in the client client of $u2$, which has been increased on firing *participate2* before. To ensure that the matched user $u2$ is either offline, blocked by $u1$ or vice versa, or not allowing incoming calls from $u1$, we instantiate the template application condition from Fig. 7.30 appropriately for *DiscardParticipate*. This is possible because the LHS contains the identifier places for the users $u1$ and $u2$. The instantiation yields the condition $ac = \exists a_1 \vee \exists(a_2, \neg\exists a_{21}) \vee \exists(a_3, \neg\exists a_{31})$ for *DiscardParticipate*.

ParticipateConference This rule is the same as the multi rule of the interaction scheme *ParticipateNewConference* in Fig. 7.31, but without the application condition and with some differences in the markings. We only describe the differences without showing the rule again. First, the request token on the place `CallR` in the LHS is not the term $callGReq(u1)$ but $partGReq(u2)$. In the RHS, the history h is replaced by $part(u2, h)$ rather than $called(u1, h)$. Finally, the *Bool* token on the place `HasJoined2` is T in this rule instead of F, because if a member wants to participate, he is connected immediately to the conference and does not need to join explicitly. Because of the higher prioritized discarding rule *DiscardParticipate* in the handler expression, we do not need an application condition ensuring that $u2$ is allowed to participate.

If the conference host $u1$ invites an active member $u2$ from the group place `Members` by firing *invite*, the produced request token $invite(u2)$ is handled by the following expression:

$$handler(inviteGReq) := DiscardInvite \parallel InviteConference$$

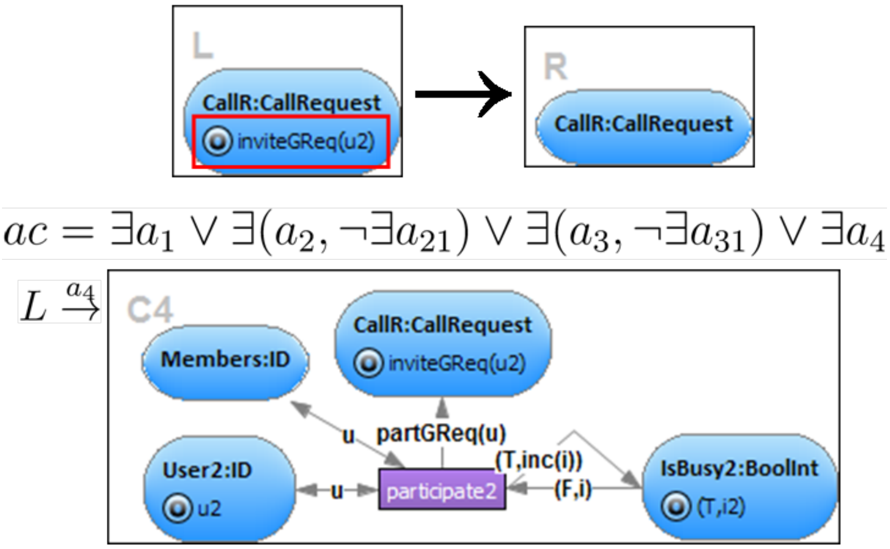


Figure 7.33.: Rule *DiscardInvite* with application condition *ac*

DiscardInvite This rule shown in Fig. 7.33 catches the exceptional cases that an active group member $u2$ cannot be invited by the host $u1$, which is the case if the invited member $u2$ is not online, if he is busy in another call, or if he cannot be called by the host. In this cases, this rule just deletes the request token. Otherwise the next rule realizes that $u2$ participates in the conference. To ensure that the matched user $u2$ is either offline, blocked by $u1$ or vice versa, or not allowing incoming calls from $u1$, we instantiate the template application condition from Fig. 7.30 appropriately for *DiscardInvite*. This is possible because the LHS contains the identifier places for the users $u1$ and $u2$. The instantiation yields the subcondition part $\exists a_1 \vee \exists(a_2, \neg\exists a_{21}) \vee \exists(a_3, \neg\exists a_{31})$ for *DiscardParticipate*. The remaining subcondition $\exists a_4$ that we give explicitly in Fig. 7.33

is satisfied if $u2$ is already busy in another call, because in this case the token $(T, i2)$ on place `IsBusy2` can match in the platform net.

InviteConference As rule *ParticipateConference* on page 211, this rule is almost the same as the multi rule of the interaction scheme *ParticipateNewConference* in Fig. 7.31. Here, the request token on the place `CallR` in the LHS is not the term $callGReq(u)$ but $inviteGReq(u)$. In the RHS, the history h is replaced by $invited(u1, u2, h)$ rather than $called(u1, h)$. In contrast to rule *ParticipateConference*, we leave the *Bool* token F on the place `HasJoined2` as it is in rule *ParticipateNewConference*, because the invited user $u2$ has to decide whether he wants to join the conference upon this invitation. Because of the higher prioritized discarding rule *DiscardInvite* in the handler expression, we do not need an application condition ensuring that $u1$ is allowed to invite $u2$ or that $u2$ is allowed to participate.

Both actions for a user leaving a conference or being kicked by the conference host have the same effect that a participating member is excluded from the conference. Therefore, we treat both actions with the same request and handler expression. A request token $hangup(u2)$ for a participating user $u2$ of a conference can be produced either by $u2$ firing his transition *hangup*, which may occur before or after joining the conference, or by the host firing his transition *kick*.

$$handler(hangupGReq) := RemoveJoined; HangupConference$$

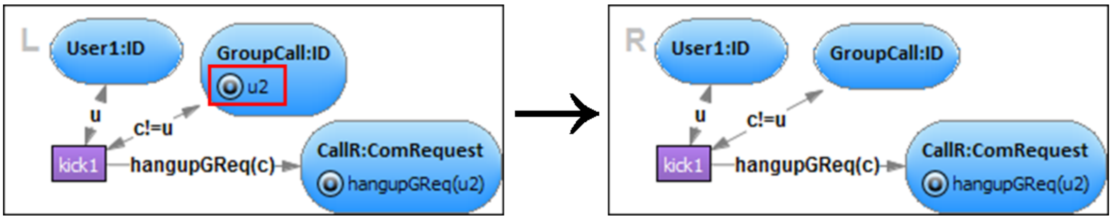


Figure 7.34.: Rule *RemoveJoined*

RemoveJoined This rule shown in Fig. 7.34 removes the identifier token of a user $u2$ who is going to hang up or be kicked from a conference from the conference place `GroupCall` containing all the group members who have joined the conference. Note that user $u2$ can hang up even if he has not joined the conference, i. e., in the case that the host invited $u2$ and his client is still “ringing”. In this case, *RemoveJoined* is not applicable and the handler just continues with the next rule.

HangupConference This rule shown in Fig. 7.35 has the inverse structural effect of the rule *ParticipateConference* on page 211 (and the interaction scheme *ParticipateNewConference* in Fig. 7.31). It decreases the call counter of $u2$’s client and sets his busy state according to Table 7.2, which we discuss for hanging up direct calls with rule *HangupDirectCall* (see Fig. 7.22).

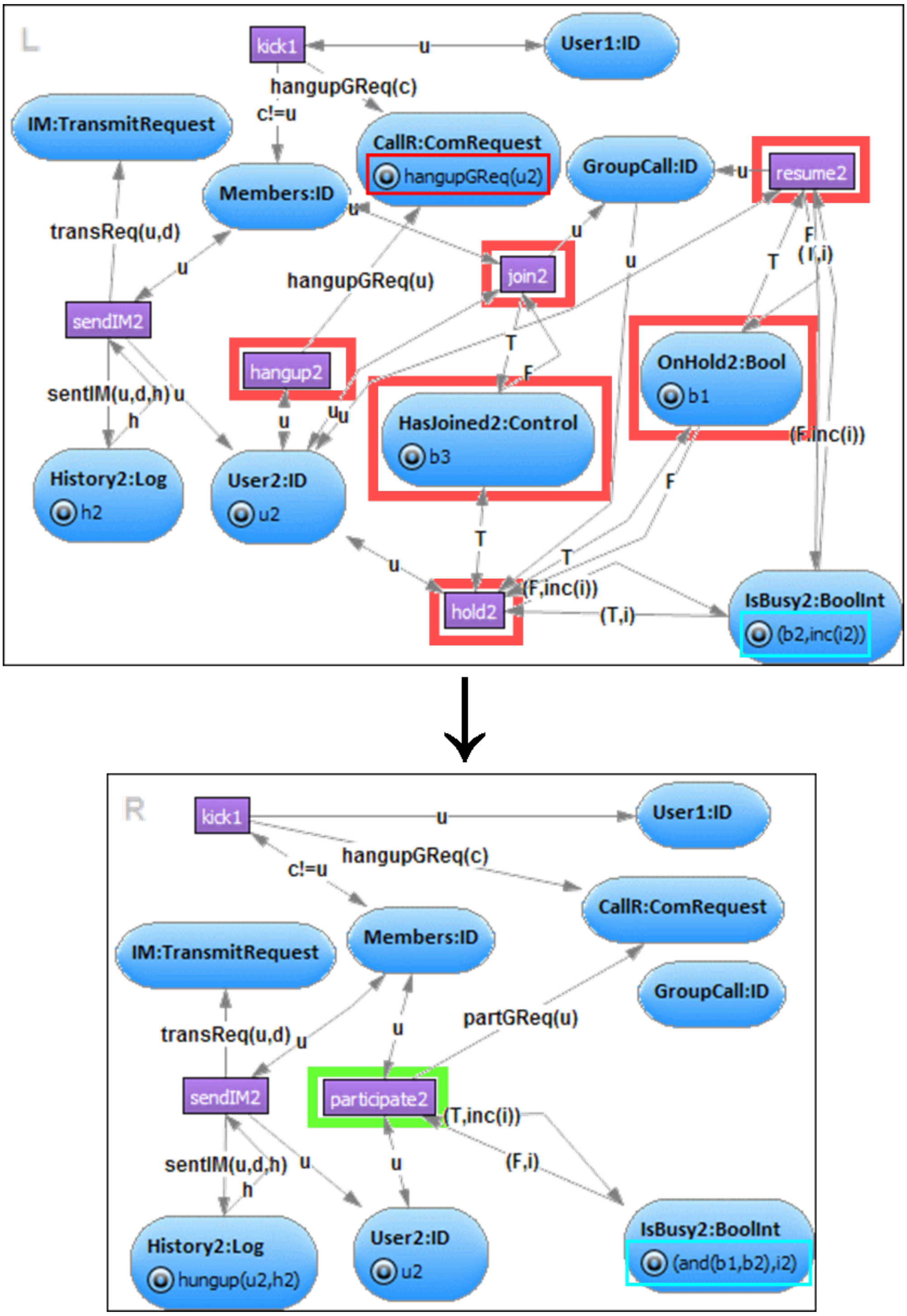


Figure 7.35.: Rule *HangupConference*

7.2.5.5. Quitting Conferences

If the host leaves his conference, then the conference is closed and all participants are kicked, too. The following handler expression realizes this reconfiguration for a request token $\mathit{quitGReq}(u1)$ that is produced if the host $u1$ fires his transition quit. The multi rules of the first two interaction schemes remove the participating clients similar to the rules for hanging up in a conference.

$$\mathit{handler}(\mathit{quitGReq}) := \mathit{RemoveAllJoined}; \mathit{KickParticipants}; \mathit{CloseConference}$$

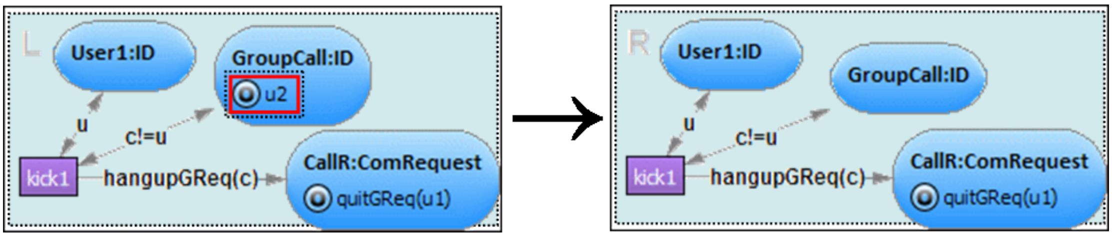


Figure 7.36.: Interaction Scheme *RemoveAllJoined*

RemoveAllJoined This interaction scheme shown in Fig. 7.36 deletes all identifier tokens from the conference place `GroupCall`. The kernel rule and the multi rule are identical except to the token $u2$ on this place. The purpose of this interaction scheme is the same as for the rule *RemoveJoined* in Fig. 7.34 for hanging up in a conference.

KickParticipants This interaction scheme shown in Fig. 7.37 reconfigures all participating members to nonparticipating members. For this, the multi rule matches and reconfigures a participating member as rule *HangupConference* in Fig. 7.35. The kernel rule matches only the preserved conference places and the host $u1$, which are removed by the next interaction scheme.

CloseConference To finally close the conference after removing all participants with the previous interaction scheme, the interaction scheme in Fig. 7.38 has the inverse effect of the interaction scheme *InitConference* in Fig. 7.29. Note that the new busy status for the former host is determined as in Table 7.2, which we discuss for hanging up direct calls with rule *HangupDirectCall* (see Fig. 7.22).

7.2.5.6. Managing Groups

All active members of a group can add other users from their contact lists to this group. If a user $u1$ wants to add a user $u2$ from his contact list, the request token $\mathit{addGReq}(u1, u2)$ is handled by the following expression:

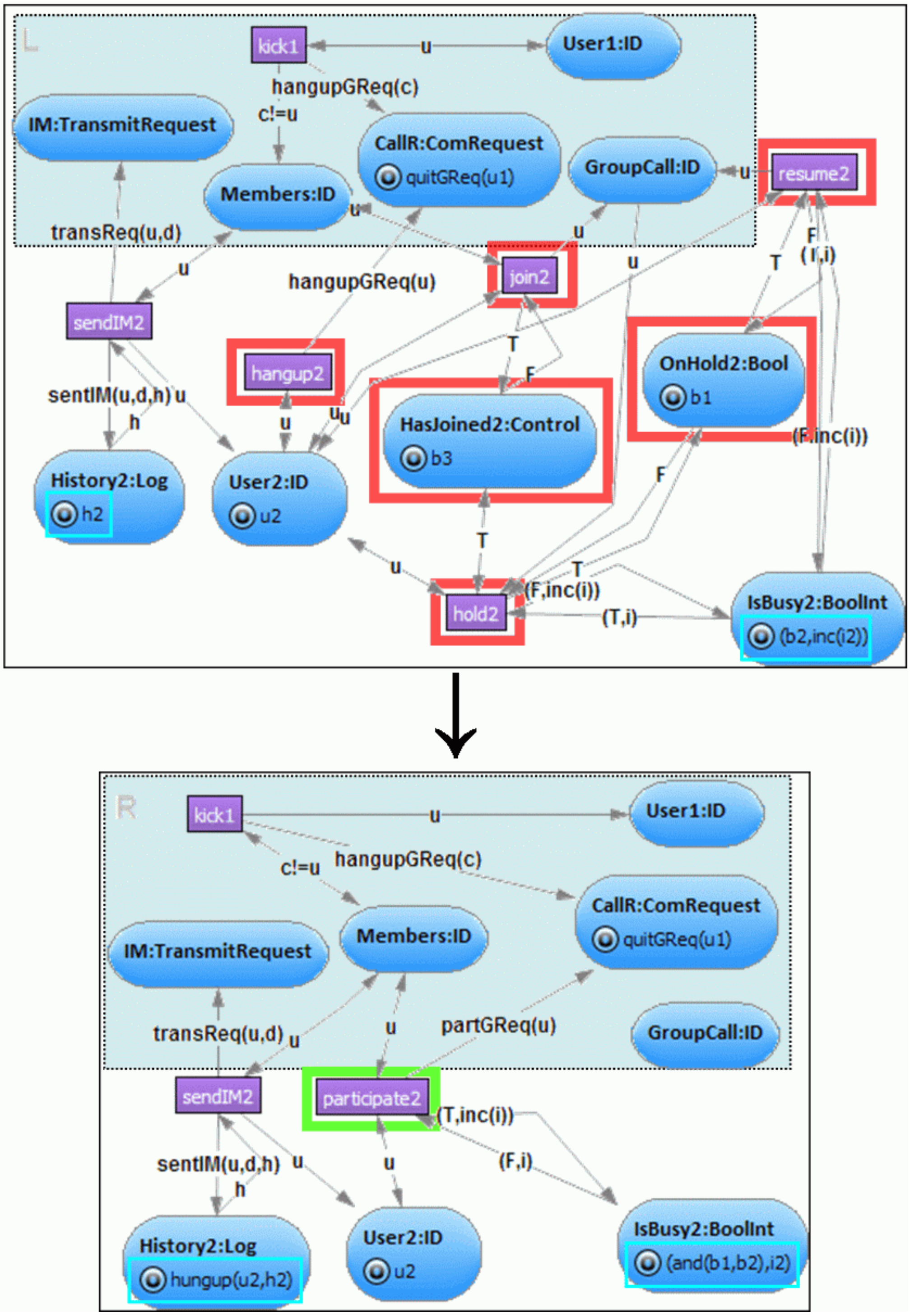


Figure 7.37.: Interaction Scheme *KickParticipants*

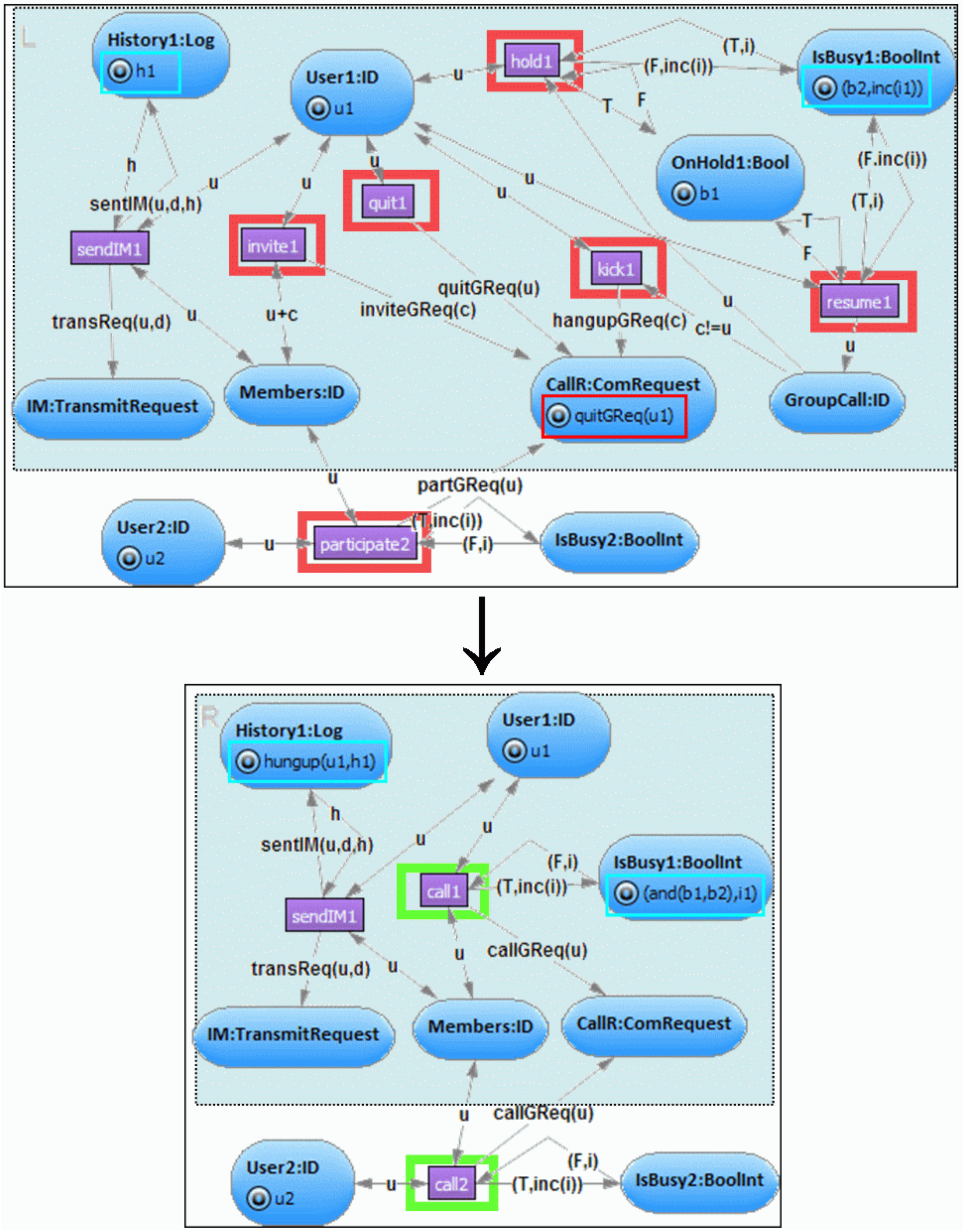


Figure 7.38.: Interaction Scheme *CloseConference*

$$
\begin{aligned}
handler(addGReq) :=\, & (DiscardAddGroup \parallel ReaddToGroup \parallel AddToGroup);\\
& AdaptToConference;\\
& (DiscardAddGroup2 \parallel ParticipateAdded)
\end{aligned}
$$

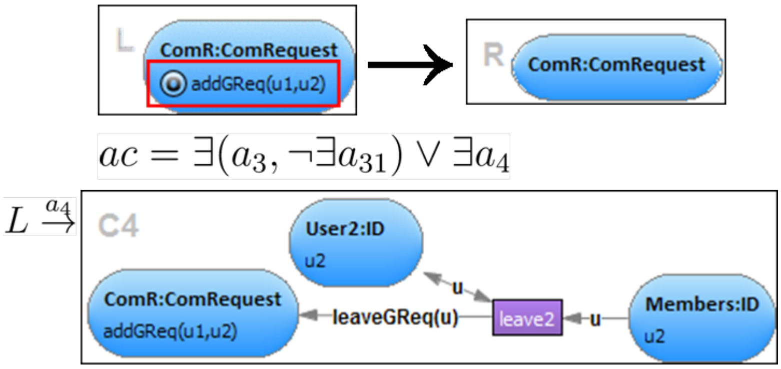


Figure 7.39.: Rule *DiscardAddGroup* with application condition *ac*

DiscardAddGroup Fig. 7.39 This discarding rule shown in Fig. 7.39 is applicable if $u2$ cannot be added to the group, which is the case if $u2$ is already an active member of the group or if $u1$ or $u2$ block the other one. The application condition $\exists(a_3, \neg\exists a_{31})$ is a part of the template in Fig. 7.30 with the same morphism names instantiated for the LHS of rule *DiscardAddGroup*, where the request token determines the names of the user $u1$ and $u2$. If $u1$ is currently blocking $u2$ or vice versa, then $\exists(a_3, \neg\exists a_{31})$ is satisfied. If $u2$ is an active member of the considered group, then $\exists a_4$ is satisfied.

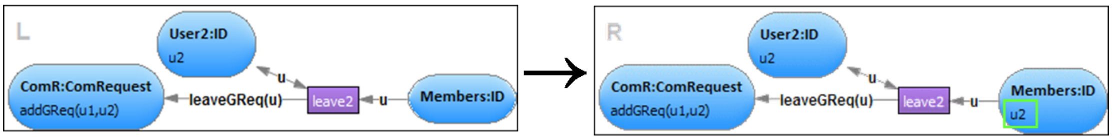


Figure 7.40.: Rule *ReaddToGroup*

ReaddToGroup This rule in Fig. 7.40 is applicable if $u2$ is an inactive member of the respective group who gets readded by $u1$. In this case, this rule adds an identifier token for $u2$ to the group place Members so that $u2$ becomes an active member again. We do not need a negative application condition forbidding that this identifier token already exists on place `Members`, because in this case the higher prioritized discarding rule *DiscardAddGroup* would have been applicable (cf. its application condition $\exists a_4$). The rule *ReaddToGroup* does not delete the request token so that the following rules in the sequence can let $u2$ participate in a possibly running conference in this group.

AddToGroup If the higher prioritized rules *DiscardAddGroup* and *ReaddToGroup* are not applicable, then this rule shown in Fig. 7.41 connects $u2$'s client with the corresponding group as the rule *CreateGroup* in Fig. 7.27 does initially for each of the three clients in a newly created group. Note that this rule does not delete the request token, because the following rules should be able to add $u2$ to a running conference if possible.

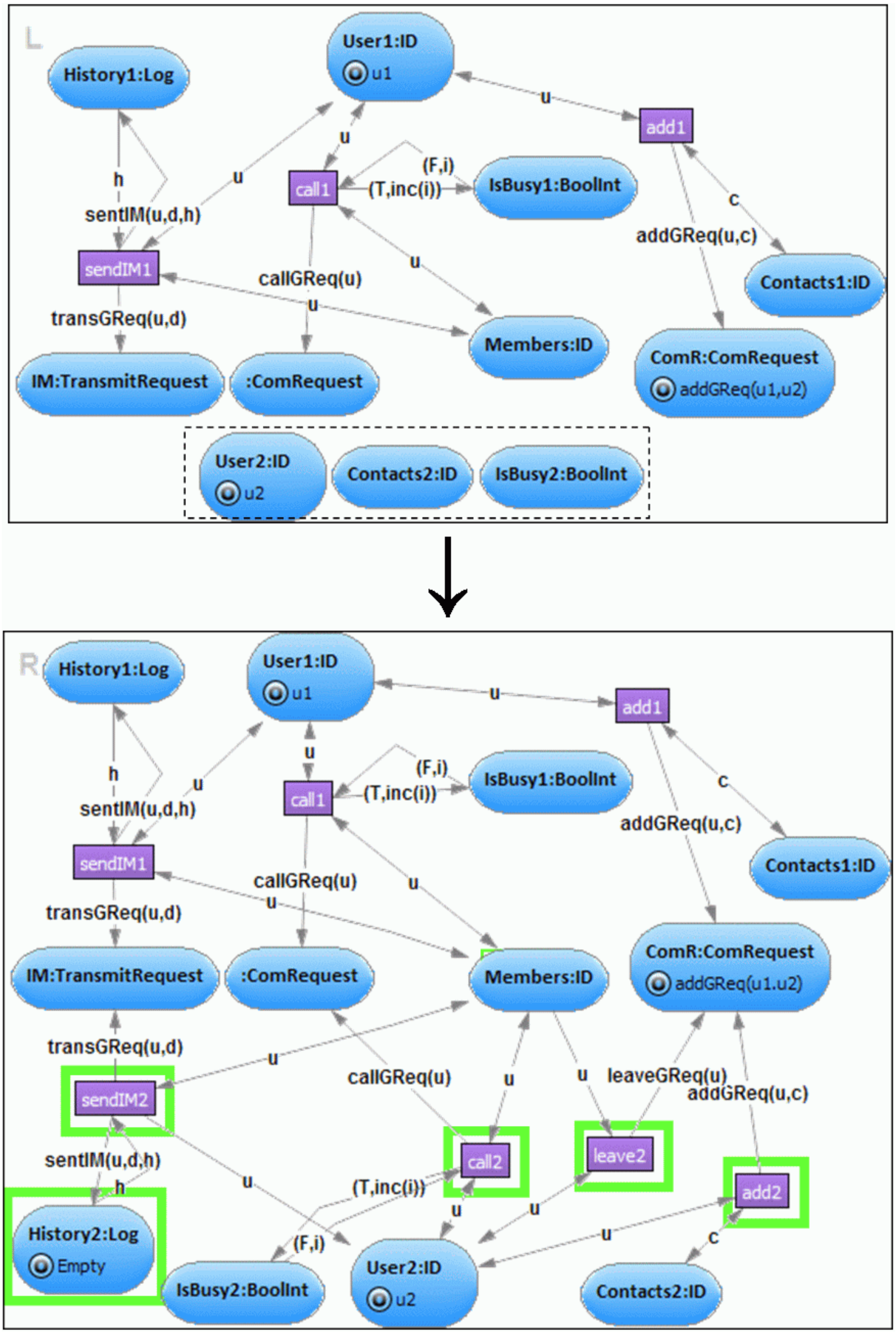


Figure 7.41.: Rule *AddToGroup*

AdaptToConference If there is a running conference in this group, the rule in Fig. 7.42 is applicable to the (re)added user $u2$ because the conference host's transition for inviting

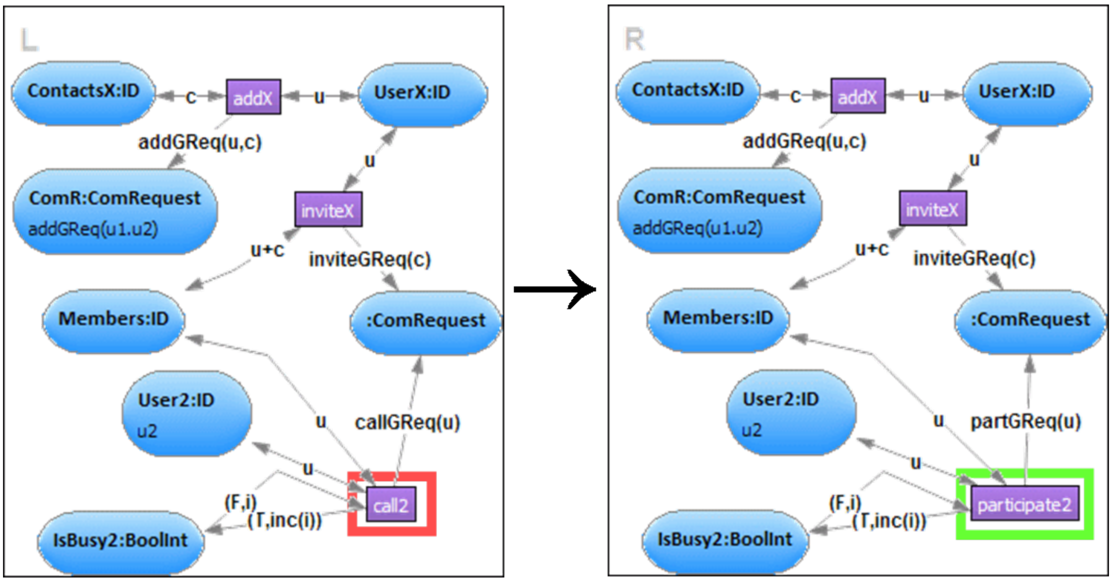


Figure 7.42.: Rule *AdaptToConference*

group members to the conference can be matched. Note that it is not necessary that the adding user $u1$ be the conference host. The transition of $u2$ for calling is replaced by a transition for participating, according to the purpose and the effect of the multi rule of the interaction scheme *InitConference* in Fig. 7.29.

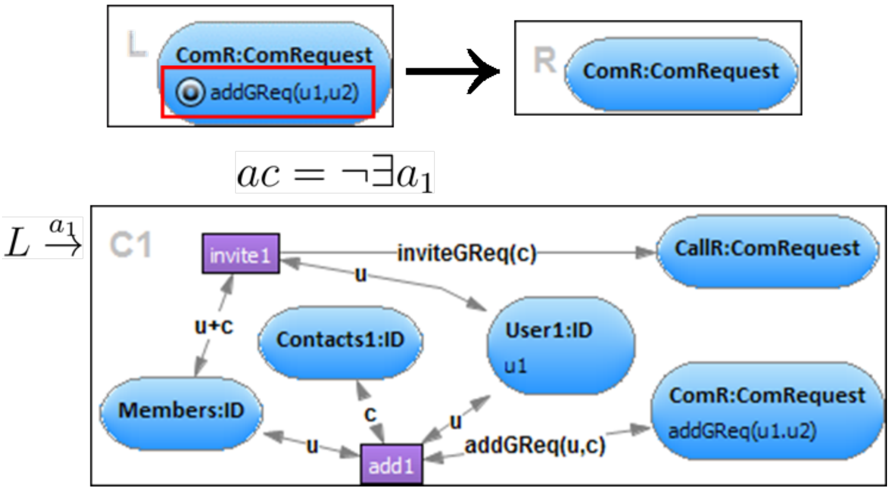


Figure 7.43.: Rule *DiscardAddGroup2* with application condition *ac*

DiscardAddGroup2 The second discarding rule of this handler shown in Fig. 7.43 is applicable if the adding user $u1$ is not the conference's host (identified by the characteristical host transition *invite1* for $u1$). In this case, rule *DiscardAddGroup2* deletes the request token. Note that this rule is also applicable if there is no running conference in this group at all.

ParticipateAdded If the higher prioritized discarding rule *DiscardAddGroup2* is not applicable, we know that the adding user $u1$ is the host of a running conference in this group. According to our requirements for adding a user $u2$ to a group, $u2$ should automatically participate in the conference in this case. This is realized by the rule *ParticipateAdded*, which we do not shown explicitly. This rule is structurally the same as the multi rule of the interaction scheme *ParticipateNewConference* in Fig. 7.31, which lets an invited user $u2$ participate in a conference (cf. also the rule *InviteConference* on page 212). The only difference is that the place `CallR` in the LHS carries the request token $addGReq(u1, u2)$ rather than $callGReq(u1)$.

An active member of a group can choose to leave to group so that they cannot longer participate in conferences or receive messages in this group. If the group owner removes a member from the group, this has the same effect. Both actions produce a request token $leaveGReq(u2)$ where $u2$ is the name of the leaving or removed user. Note that the transitions for leaving and removing both delete the identifier token $u2$ on the group place `Members` (see the RHS of rule *CreateGroup* in Fig. 7.27). The following handler additionally takes care of kicking $u2$ from a running conference or even quitting a conference if $u2$ is its host:

$$\begin{aligned} handler(leaveGReq) :=& (ForceHangup \parallel DiscardLeaveGroup);\\ & RemoveAllJoined2; KickParticipants2;\\ & CloseConference2 \end{aligned}$$

ForceHangup If $u2$ is participating in a conference without being the host in the conference that he is leaving with the current request handling, this rule shown in Fig. 7.44 forces $u2$ to hang up in this conference. The rule's effect is similar to the one of rule *HangupConference* in Fig. 7.35. In addition, we need to match the transition *remove1* of the group owner, to match the unnamed place with the request token for leaving rather than a request token on the place `CallR`. Note that the request token is deleted by this rule. Therefore, the following rules are only applicable if $u2$ is either not participating in a possibly running conference or if he is its host — because rule *ForceHangup* is not applicable then.

DiscardLeaveGroup If $u2$ is not the host of a running conference (identified by the characteristical host transition *invite2* for $u2$, cf. rule *DiscardAddGroup2* in Fig. 7.43), then $u2$ has successfully left the group at this point in the handler and there is no further need for reconfiguration.

RemoveAllJoined2, KickParticipants2, CloseConference2 If the discarding rule *DiscardLeaveGroup* is not applicable, then the leaving user $u2$ is the host of a running conference in this group. In this case, we cannot simply let $u2$ hang up in this

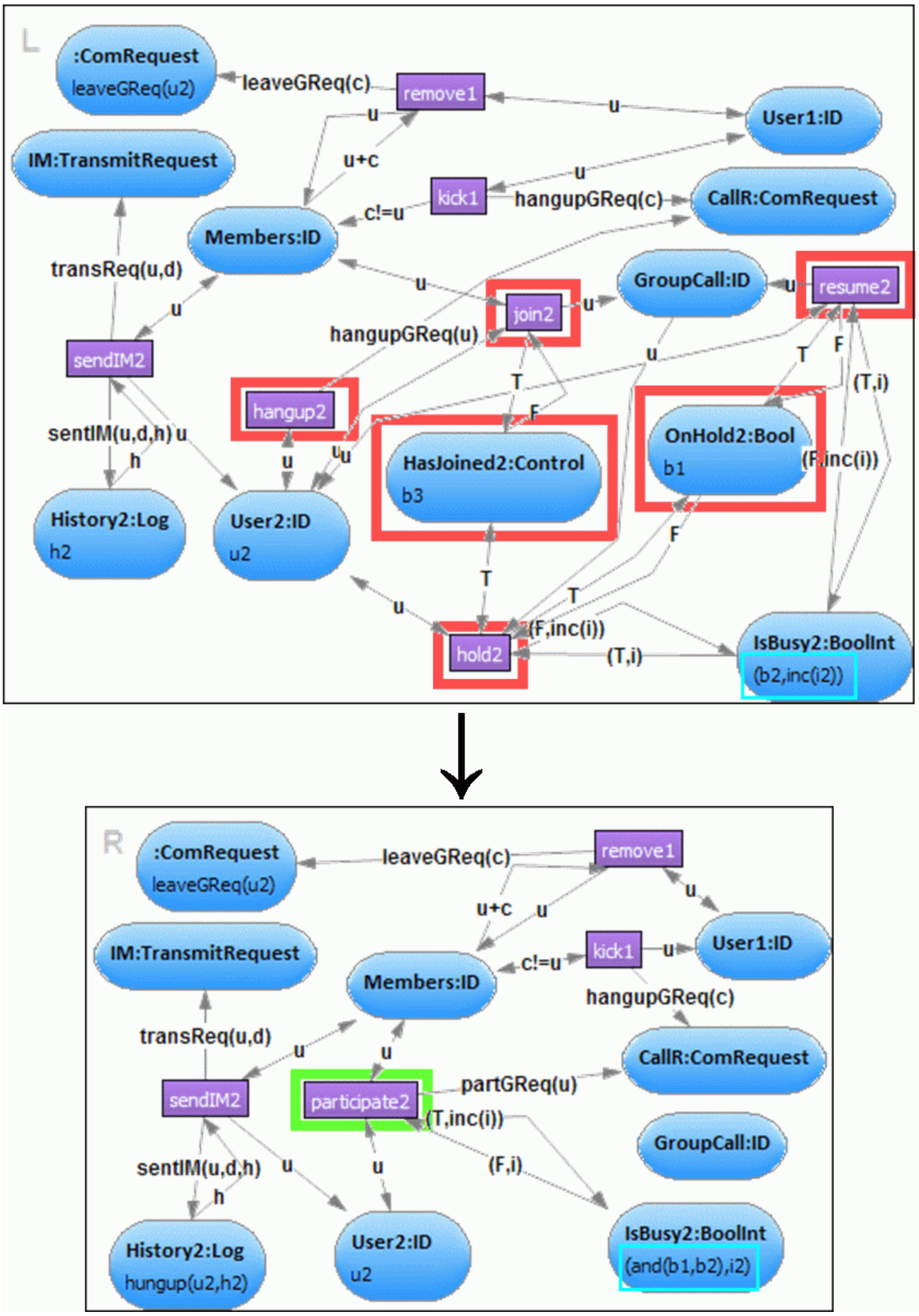


Figure 7.44.: Rule *ForceHangup*

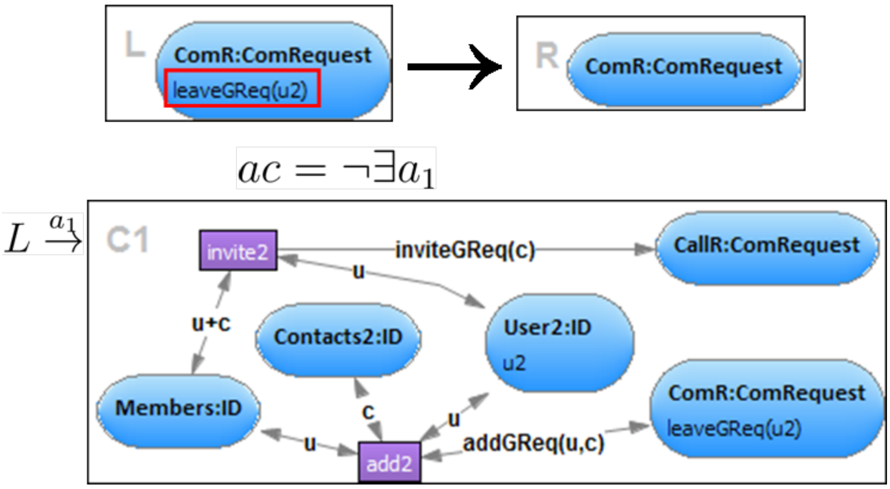


Figure 7.45.: Rule *DiscardLeaveGroup* with application condition *ac*

conference with the rule *ForceHangup* in Fig. 7.44. This rule is not even applicable, because a host client has other transitions in a conference than the other participants. Therefore, we need to achieve the same effect as the interaction schemes for quitting a conference, including kicking all other participants and closing the conference properly. For this, we can adapt the interaction schemes of the handler expression for the *quitGReq* request from Sect. 7.2.5.5 as we have adapted the rule *HangupConference* in Fig. 7.35 to get *ForceHangup*. We do not show these interaction schemes explicitly. After the application of this sequence of interaction schemes, the conference host $u2$ has finally left the group.

With the rules and handlers for managing groups and conferences, we can now give the handler expression for extending a direct call to a group (as promised in Sect. 7.2.4.1). In a direct channel between the user $u1$ and $u2$, the user $u1$ can fire his transition *add*, which produces a request token $exGReq(u1, u2, u3)$, where $u3$ is a name of $u1$'s contact list. $u1$ becomes the owner of the new group. If there is a ongoing direct call between $u1$ and $u2$, this call is ended and a conference in the new group is started with $u1$ being its host. This request is handled by the following expression:

$$\begin{aligned} \mathit{handler}(\mathit{exGReq}) := {} & \mathit{CreateGroupForExt}; (\mathit{HangupForExt} \parallel \mathit{DiscardExtend}); \\ & \mathit{InitConferenceForExt}; \mathit{ParticipateConference}; \\ & \mathit{DiscardInvite}; \mathit{InviteConference} \end{aligned}$$

CreateGroupForExt We do not show this rule explicitly because it is the same as rule *CreateGroup* in Fig. 7.27 up to the marking. The differences are that the request token on the place `ComR1` is $exGReq(u1, u2, u3)$ rather than $groupReq(u1, u2, u3)$, that *CreateGroupForExt* does not delete this token, and that additionally a token $callGReq(u1)$ on the unnamed *ComRequest* place of the new group is created (as if

$u1$ had fired his transition *call*1 in this group). The remaining rules in this handler expression work on these request tokens. Note that adding a third user to a direct channel always results in the creation of a new group. This is the reason why rule *CreateGroupForExt* is the first unconditioned rule of this handler expression.

HangupForExt This rule is the same as rule *HangupDirectCall* in Fig. 7.22 up to slight differences in the marking: Of course, the request token in the LHS has to be *exGReq*$(u1, u2, u3)$ instead of *hangupReq*$(u1, u2)$.[7] Moreover, this request token is not matched on the place of sort *ComRequest* that is part of the direct channel structure but rather on the place `CR1` of $u1$'s client. If this rule *HangupForExt* is applicable, there has been a call in the direct channel between $u1$ and $u2$ so that the following rules should establish a conference between $u1$, $u2$, and possibly $u3$ in the group that has just been created by rule *CreateGroupForExt*. The rule *HangupForExt* preserves the request token so that its handling with the following rules can realize these reconfigurations.

DiscardExtend This discarding rule is intended to terminate the handling in the case that the higher prioritized rule *HangupForExt* is not applicable. For this, it matches two *ComRequest* places and deletes a request token on each of them, *exGReq*$(u1, u2, u3)$ on the one and *callGReq*(*u1*) on the other (which has been produced by rule *CreateGroupForExt* before). With this, all request tokens are deleted, and the handling is terminated because the remaining rules in the handler expression cannot be applied. The rule *DiscardExtend* does not have application conditions, so we omit its simple visual representation, here.

InitConferenceForExt This interaction scheme is almost the same as the interaction scheme *InitConference* in Fig. 7.29. It matches the token *callGReq*(*u1*) that has been created by rule *CreateGroupForExt* in the new group structure and initiates a conference. In addition to the effect of the interaction scheme *InitConference*, the scheme *InitConferenceForExt* creates the request tokens *partGReq*$(u2)$ and *inviteGReq*$(u3)$ on the conference place `CallR`, as if $u1$ and $u2$ had fired their transitions *invite1* (to invite the user $u3$) and *participate2*, respectively.

ParticipateConference This is the same rule as described for the handler expression for the request constructor *partGReq* on page 211, which triggers a reconfiguration that lets participate a user ($u2$ in this case) in a running conference. The request token *partGReq*$(u2)$ has been created by the preceding interaction scheme *InitConferenceForExt*. In the new conference, $u2$ is participating and does not need to join, because he has been in the direct call with $u1$ before.

DiscardInvite, InviteConference These are the same rules as described for the handler expression for the request constructor *inviteGReq* on page 212 and in Fig. 7.33, which

[7] The request token *callGReq*(*u1*) produced by the preceding rule *CreateGroupForExt* is not regarded by rule *HangupForExt*, because it serves for the next rules that create a conference between $u1$, $u2$, and $u3$.

triggers a reconfiguration that lets participate a user ($u3$ in this case) in a running conference. If the rule *DiscardInvite* — which deletes the request token $\mathit{inviteGReq}(u3)$ produced by the preceding interaction scheme *InitConferenceForExt* — is not applicable, then $u3$ can be invited by the conference host $u1$ (cf. Fig. 7.33). In this case, the rule *InviteConference* can be applied on this token, after which $u3$ is participating in the new conference and can decide to join the conference, i. e., the $u3$ is supposed to "ring" w. r. t. the new conference.

Remark (Reuse of request handler rules). For complex request handlings involving effects that have been modeled with rules or interaction schemes in other handler expressions, these rules and schemes can be reused as in the handler for the *exGReq* request. The rule *InitConferenceForExt* creates the tokens $\mathit{partGReq}(u2)$ and $\mathit{inviteGReq}(u3)$ if a direct call between $u1$ and $u2$ is extended to a conference in the new created group. With this, the appended rules *ParticipateConference*, *DiscardInvite*, and *InviteConference* from the handlers for these particular requests reconfigure the conference accordingly. However, for the handler rules for *leaveGReq* (leave a group) request tokens treating the case that the leaving user $u2$ is the host (*RemoveAllJoined2*, *KickParticipants2*, *CloseConference2*, see page 220), we cannot simply reuse the interaction schemes for a quitting host a conference, because the *leaveGReq* request token is independent of a running conference and does not lie on a place in the conference. Therefore, have adapted the interaction schemes as described to match the request token on the other place.

Alternatively, one may want to try to optimize the request handlers w. r. t. reusing rules and interaction schemes. For the mentioned *leaveGReq* request handler, we then had to formulate an additional rule creating the appropriate *quitGReq* request token that is matched by the schemes for the *quitGReq* request handler.

7.2.6. Exclusive Data Transmission

In Sect. 7.2.5.2, we defined a handler expression with a rule that multicasts chat a message in a group to all active members. With this and the sending of messages in direct channels (without reconfiguration), we have realized the nonexclusive communications in Skype. For exclusive communications — i. e., the spoken data in calls and conferences (cf. Sect. 5.1.1.2) —, we let the platform model "observe" the request places of the type *TransmitRequest* in the client cores. We suppose that every time a user $u1$ says something, this corresponds to a firing of the transition that produces a request token $\mathit{transmitReq}(u1, d1)$, where the data value $d1$ represents the said information (see transition *say* in the client core in Fig. 7.3). For a handling of this request, we expect that it distributes the data $d1$ to the appropriate receiving clients, either a user $u2$ who is in a direct call with $u1$ or a set of users participating in a conference together with $u1$. If there is no adequate receiver, we expect the volatile spoken data to vanish without being heard by any other client. The following request handler expression realizes this:

$$\mathit{handler}(\mathit{transReq}) := \mathit{TransmitCall} \parallel \mathit{TransmitConference}$$

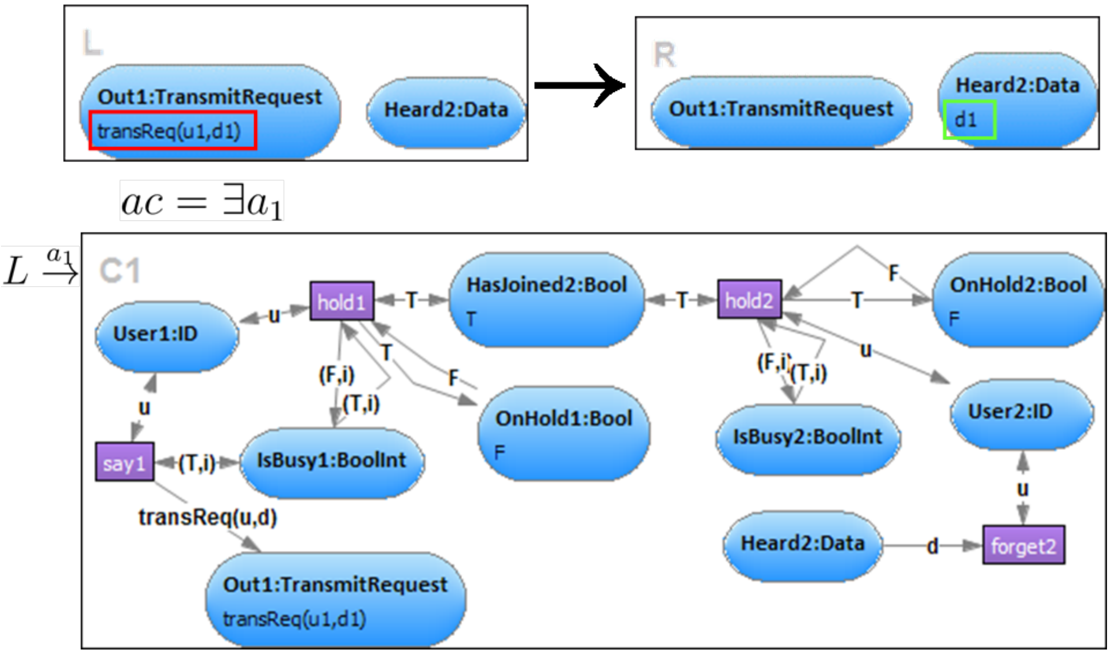


Figure 7.46.: Rule *TransmitCall* with application condition *ac*

TransmitCall This rule shown in Fig. 7.46 transmits the *Data* value of the transmit request token produced by $u1$ to a user $u2$ who is currently in a direct call with $u1$. The rule's LHS and RHS match the source place with the request and the target place that represents all the data that $u2$ heard to far. The application condition $\exists a_1$ ensures that there exists a direct call structure (cf. rule *CallDirect* in Fig. 7.21) between the clients of $u1$ and $u2$ such that $u2$ has joined the call (because of the T token on the place `HasJoined2`) and neither of them has put the call on hold (because of the F tokens on the places `OnHold1` and `OnHold2`).

TransmitConference This interaction scheme shown in Fig. 7.47 multicasts the *Data* value of the transmit request token produced by $u1$ to all joined participants of a running conference in which $u1$ is also currently participating and joined. The kernel rule of this scheme matches the $u1$'s request place and token. The multi rule is intended to match a joined participant $u2$. This is ensured by the application condition $\exists a_1$ for the multi rule. To be satisfied, it demands that $u1$ and $u2$ be connected to a conference (cf. schemes *InitConference* in Fig. 7.29 and *ParticipateNewConference* in Fig. 7.31)where he is participating and joined (determined by the tokens $u1$ and $u2$ on the conference place `GroupCall`). These tokens also imply that neither of them has put the conference call on hold. Maximal matching of *TransmitConference* ensures that all joined participating members of the conference in which $u1$ participates get multicasted the message $d1$, analogously to the interaction scheme *SendGroupMessage* for multicasting messages in groups in Fig. 7.28.

Note that the kernel rule is always applicable because it matches just the request

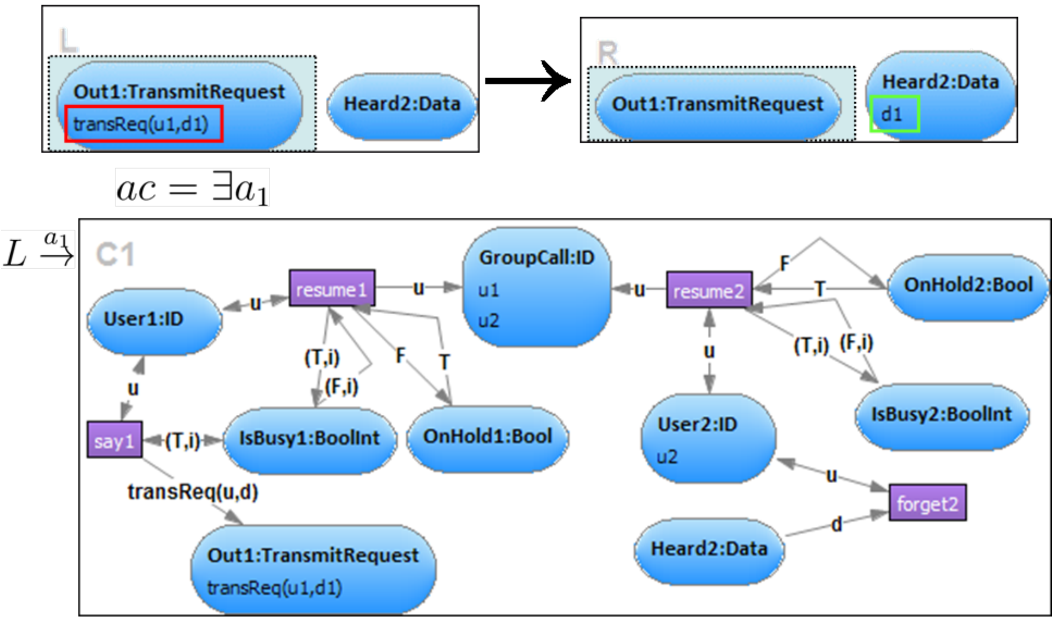


Figure 7.47.: Interaction scheme *TransmitConference* with application condition *ac* for its multi rule

token and its carrying place. If $u1$ is not participating in a conference, only the kernel rule (without a multi rule instantiation) is applied, which has the effect of discarding the message without distributing it to any other client. Because of this, we do not need to formulate a further discarding rule to achieve a functional handling that properly consumes the request token.

7.2.7. Overview: Requests and Handler Function for Skype

To conclude the example modeling of Skype and the definition of the Communication Platform model *SCPM*, we summarize its function *handler*: $Constr(S_{Req}) \rightarrow PSEQ(R \cup IS)$ in Table 7.3 by giving the handler expression for every request constructor in S_{Req} and its corresponding user action.

Table 7.3.: Complete definition of Skype *handler* function

user action	**request constructor**	**handler expression for tokens of request constructor**
	in direct channels (Fig. 7.1)	
requestDirect	*directReq*	*DiscardDirectReqExisting* $\parallel$ *CreateDirectChannel*
requestContact	*contactReq*	*DiscardContactExchange* $\parallel$ *CreateContactExchange*

deny	*delCXReq*	*DeleteContactExchange*
accept	*permReq*	*DeleteOppositeExchange* ; (*ExtendToPermittedChannel* $\parallel$ *CreatePermittedChannel*)
delete	*delCReq*	*DeleteContact*
undelete	*undelCReq*	*UndeleteContact*
call	*callReq*	*DiscardCallDirect* $\parallel$ *CallDirect*
hangup	*hangupReq*	*HangupDirectCall*
forward	*forReq*	*PrepareDirectChannel* ; *ForwardHangup* ; (*DiscardForwardCall* $\parallel$ *ForwardCall*)
add	*exGReq*	*CreateGroupForExt* ; (*HangupForExt* $\parallel$ *DiscardExtend*) ; *InitConferenceForExt* ; *ParticipateConference* ; *DiscardInvite* ; *InviteConference*
in groups (Fig. 7.2)		
requestGroup	*groupReq*	*CreateGroup*
sendIM	*transGReq*	*SendGroupMessage*
call	*callGReq*	*InitConference* ; *ParticipateNewConference*
participate	*partGReq*	*ParticipateConference*
invite	*inviteGReq*	*DiscardInvite* $\parallel$ *InviteConference*
hangup/kick	*hangupGReq*	*RemoveJoined* ; *HangupConference*
quit	*quitGReq*	*RemoveAllJoined* ; *KickParticipants*; *CloseConference*
add	*addGReq*	(*DiscardAddGroup* $\parallel$ *ReaddToGroup* $\parallel$ *AddToGroup*) ; *AdaptToConference* ; (*DiscardAddGroup2* $\parallel$ *ParticipateAdded*)
leave/remove	*leaveGReq*	(*ForceHangup* $\parallel$ *DiscardleaveGroup*) ; *RemoveAllJoined2* ; *KickParticipants2* ; *CloseConference2*
transmitting from Skype client core to exclusive communication		
say	*transReq*	*TransmitCall* $\parallel$ *TransmitConference*

Table 7.3.: Complete definition of Skype *handler* function

7.3. Example Activities in the Skype Platform Model

In this section, we simulate an excerpt of the use case scenarios for Skype described in Sect. 3.1.6. Fort this, we demonstrate in detail a run of the controlling AHOI net $ANI_{SCPM} = (AN_{SCPM}, I_0, m_0)$ as it is constructed for the Skype platform model *SCPM* according to Sect. 5.4. Every user action in the Skype platform AHLI net (consisting of a firing step with an optional reconfiguration that handles a token request) and every external action is simulated by a firing step in the controlling AHOI net. We start the scenarios with the Communication Platform model *SCPM* that has the empty platform AHLI net ANI_0 (over the algebra A^{Skype}) representing a Skype platform where no user has registered so far. The controlling AHOI net constructed for *SCPM* has the initial (individual) marking

$$(I_0, m_0) = (\{i_1, i_2\}, \{i_1 \mapsto (ANI_0, p_1), i_2 \mapsto (\mathit{CreateSkypeClient}, p_2)\})$$

because the rule *CreateSkypeClient* in Fig. 7.5 is the only external rule in *SCPM*, i. e., it does not occur in an token handler expression. The marking on the place p_2 for external rules never changes when the AHOI net fires, so we consider only the evolving of the initial platform AHLI net ANI_0 on place p_1 along the user actions in the following.

7.3.1. Creation and Activation of Skype Clients

The rule *CreateSkypeClient* in Fig. 7.5 models the external action of registering a new user and creating a client core and lies on the place p_2 of the controlling AHOI net.

Creation of a Skype client component for Alice For creating a new client component for Alice, the AHOI net fires its transition *transformation* with the variable assignment $asg_0 = \{n \mapsto ANI_0, r \mapsto \mathit{CreateSkypeClient}, o \mapsto o_0\}$ and a token selection $S_0 = (M_0, m_0, N_0, n_0)$ where $m_0(M_0) = \{(ANI_0, p_1), (\mathit{CreateSkypeClient}, p_2)\}$ are the consumed and $n_0(N_0) = \{(ANI_1, p_1), (\mathit{CreateSkypeClient}, p_2)\}$ are the created tokens. We omit the insignificant details on the concrete individuals for the token selections. In this assignment, the AHLI morphism $o_0 \in A^{AHOI}_{SCPM,Mor}$ is the match

$$o_0 = (o_{0,P}, o_{0,T}, o_{0,A}, o_{0,I})\colon L_{\mathit{CreateSkypeClient}} \to ANI_0$$

with $o_{0,A} = \overline{asg_0^t}$ being induced by the free construction over the token variable assignment $asg_0^t\colon Y_{\mathit{CreateSkypeClient}} \to A_{ANI_0}$ with $asg_0^t(u_1) = \mathit{Alice} \in A^{Skype}_{ID}$. Note that the match's algebra component $o_{0,A}\colon T_\Sigma(Y_{\mathit{CreateSkypeClient}}) \to A_{ANI_0}$ is well-defined because u_1 is the only token variable occurring in this rule and therefore the family of token variable sets $Y_{\mathit{CreateSkypeClient}} = (Y_{\mathit{CreateSkypeClient},s})_{s \in S_{ANI_0}}$ is empty up to the set $Y_{\mathit{CreateSkypeClient},ID} = \{u_1\}$. Further, $L_{\mathit{CreateSkypeClient}}$ is the empty net so that the structural match components $o_{0,P}\colon \emptyset \to P_{ANI_0}$, $o_{0,T}\colon \emptyset \to T_{ANI_0}$, and $o_{0,I}\colon \emptyset \to I_{ANI_0}$ are the unique empty functions.

The rule *CreateSkypeClient* is applicable at the match o_0 because there does not yet exist a place in $\mathit{ANI_0}$ carrying an *ID* token with the value *Alice* and therefore its application condition is satisfied. The pair $(\mathit{transformation}, \mathit{asg}_0)$ is a consistent transition assignment for the controlling AHOI net (AN_{SCPM}, I_0, m_0) because the firing conditions of the transition *transformation* are satisfied under the assignment asg_0:

$$\overline{\mathit{asg}_0}(\mathit{cod}(o)) = \mathit{cod}_{A^{AHOI}_{SCPM}}(o_0) = \mathit{ANI_0} = \overline{\mathit{asg}_0}(n)$$

$$\overline{\mathit{asg}_0}(\mathit{isApplicable}(r, o)) = \mathit{isApplicable}_{A^{AHOI}_{SCPM}}(\mathit{CreateSkypeClient}, o_0) = T = \overline{\mathit{asg}_0}(T)$$

From this, we get that $\overline{\mathit{asg}_0}(\mathit{apply}(r, o)) = \mathit{ANI_1}$, where $\mathit{ANI_1}$ is the result of the direct AHLI net transformation $\mathit{ANI_0} \xRightarrow{\mathit{CreateSkypeClient}, o_0} \mathit{ANI_1}$ along inclusions shown in Fig. 7.48. Note that the places and transitions of the client have the same names as in the rule's RHS, extended with the suffix *-A*.

We conclude that the firing of the controlling AHOI net results in

$$(AN_{SCPM}, I_0, m_0) \xrightarrow{\mathit{transformation}, \mathit{asg}_0, S_0} (AN_{SCPM}, I_1, m_1)$$

where the marking (I_1, m_1) now contains the platform net $\mathit{ANI_1}$ rather than $\mathit{ANI_0}$ on the place p_1.

Activation of Alice's client As the next user action, Alice activates her new client. For this, the controlling AHOI net calculates a firing of Alice's transition *activateA* in $\mathit{ANI_1}$ by firing itself the higher-order transition *tokengame* (see Fig. 5.8) in the step

$$(AN_{SCPM}, I_1, m_1) \xrightarrow{\mathit{tokengame}, \mathit{asg}_1, S_1} (AN_{SCPM}, I_2, m_2)$$

with the variable assignment $\mathit{asg}_1 = \{n \mapsto \mathit{ANI_1}, t \mapsto \mathit{activateA}, \mathit{asg} \mapsto \widehat{\mathit{asg}}_1, s \mapsto \hat{S}_1\}$ and the token selection $S_1 = (M_1, m_1, N_1, n_1)$ where $m_1(M_1) = \{(\mathit{ANI}_1, p_1)\}$ is the consumed and $n_1(N_1) = \{(\mathit{ANI}_2, p_1)\}$ is the created token. The variable assignment $\widehat{\mathit{asg}}_1 = \{u \mapsto \mathit{Alice}\}\colon Var(\mathit{activateA}) \to A^{\mathit{Skype}}$ assigns the name identifier *Alice* from the environment place `UserA` of transition *tokengame* in $\mathit{ANI_1}$ to the arc inscription variable u.[8] We choose as token selection $\hat{S}_1 = (\hat{M}_1, \hat{m}_1, \hat{N}_1, \hat{n}_1)$ for firing *activateA* in $\mathit{ANI_1}$ the obvious one consuming the tokens $\hat{m}_1(\hat{M}_1) = \{(\mathit{Alice}, \texttt{UserA}), (\mathit{Offline}, \texttt{StateA})\}$ and producing the tokens $\hat{n}_1(\hat{N}_1) = \{(\mathit{Alice}, \texttt{UserA}), (\mathit{Online}, \texttt{StateA}), ((F, 0), \texttt{IsBusyA})\}$.

We first verify that $(\mathit{tokengame}, \mathit{asg}_1)$ is a consistent transition assignment for (AN_{SCPM}, I_1, m_1) by checking the firing conditions of the transition *tokengame*.[9] We have that $\mathit{producesReq}(t, n)$ evaluates to false under asg_1 because the firing of *activateA* does not produce a request token (of a request sort in $S_{\mathit{Req}} \subseteq S_{\Sigma_{\mathit{Skype}}}$). Then, it is easy to check that the consistent transition assignment $(\widehat{\mathit{asg}}_1, \mathit{activateA})$ in $\mathit{ANI_1}$ is enabled under the token selection $\hat{S}_1$ so that $\mathit{isEnabled}(n, t, \mathit{asg}, s)$ evaluates to *true* under

[8] Note that because of the assignment of *Alice* to the reserved user variable u, this indicates a user action performed by Alice (cf. Sect. 5.3). The same holds for all assignments of *ID* values to this variable in all following firings.

[9] The token selection condition for AHLI net firing of *tokengame* is satisfied because we have selected the net token $\mathit{ANI_1}$ to be consumed and $\overline{\mathit{asg}_1}(\mathit{fire}(n, t, \mathit{asg}, s))) = \mathit{ANI_2}$ to be produced with the token selection S_1 on place $p1$.

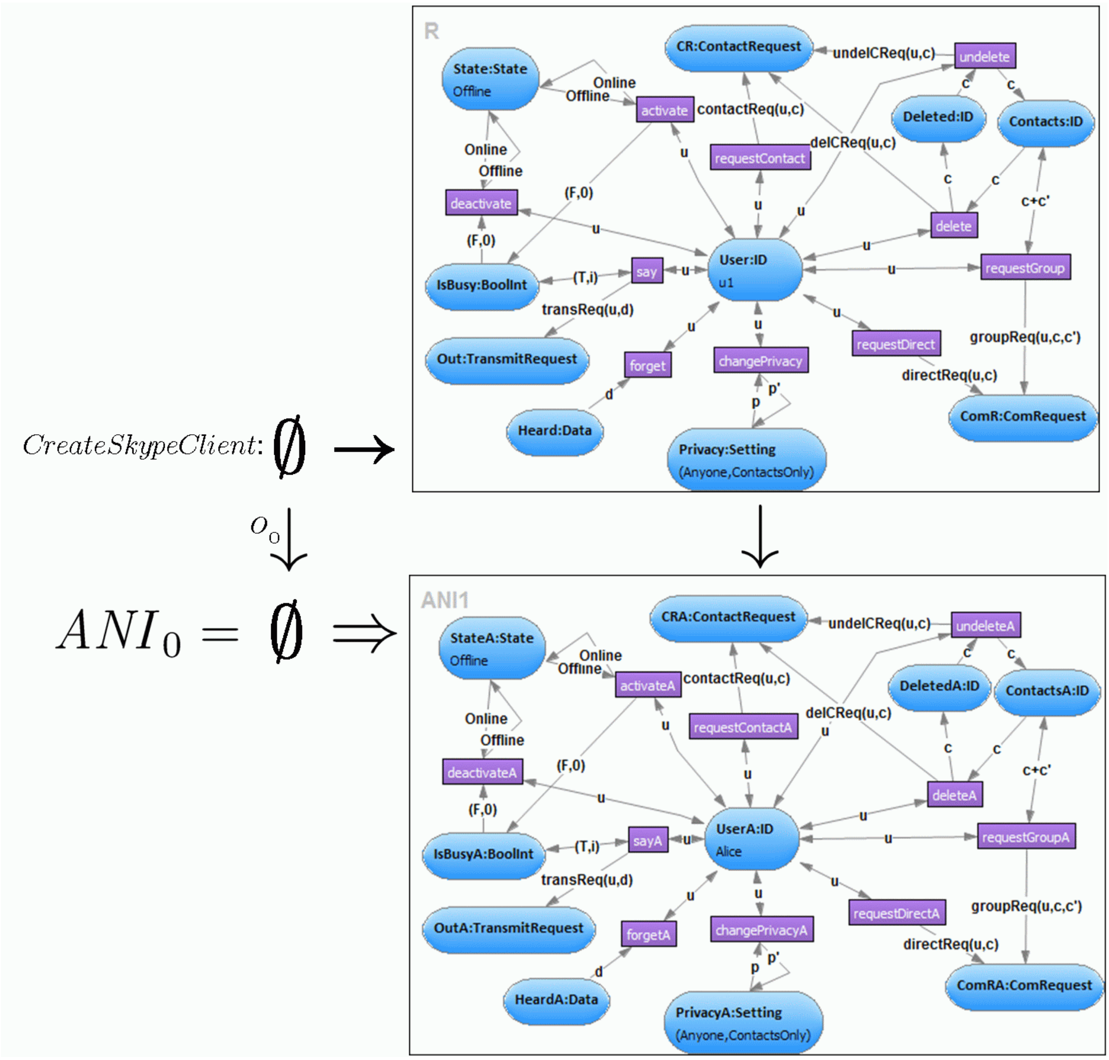


Figure 7.48.: Direct AHLI net transformation $ANI_0 \xRightarrow{CreateSkypeClient, o_0} ANI_1$

$\widehat{asg}_1$. Therefore, we can fire *tokengame* in AN_{SCPM} and get the marking (I_2, m_2) with $\overline{asg_1}(fire(n, t, asg, s)) = ANI_2$ on place p_1. The AHLI platform net ANI_2 shown in Fig. 7.49 results from the firing step

$$ANI_1 \xrightarrow{activateA, \widehat{asg}_1, \hat{S}_1} ANI_2.$$

Remark (Presentation of rule matches and token selections). For the rest of the use case scenarios, we only give the token variable assignment $asg_i^t \colon Y_\varrho \to ANI_i$ for a rule match $o_i \colon L_\varrho \to ANI_i$, as it induces the match's algebra component $o_{i,A}$ by a free construction. We illustrate the structural match components $o_{0,P}$ and $o_{0,T}$ graphically or by explaining the correspondence of places and transitions with appropriately similar names in the LHS and the transformed platform net. Further, we omit the individual component $o_{i,I}$

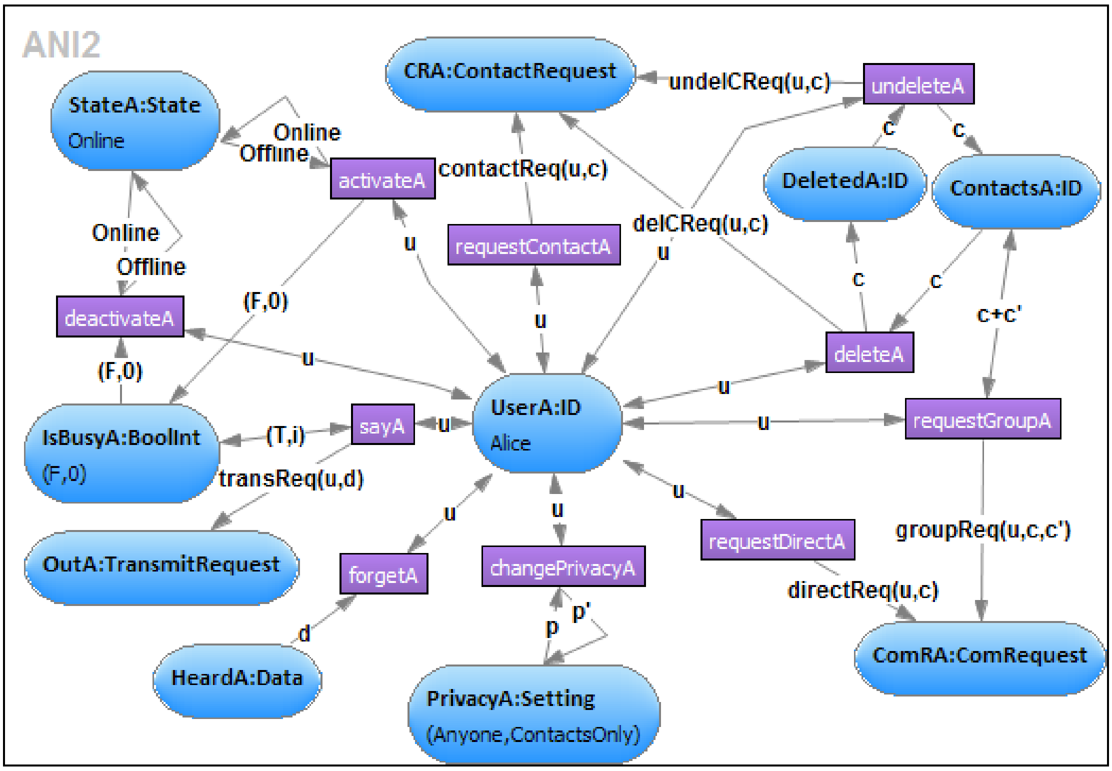


Figure 7.49.: AHLI platform net ANI_2 with activated client for Alice

because it usually can be inferred from the graphical illustration of $o_{i,P}$ and $o_{i,T}$ and the algebra match component $o_{i,A}$.

Also, the concrete individual token selections for firings of the controlling AHOI net AN_{SCPM} and of the platform nets ANI_i usually can be inferred from the variable assignment and the information which token values should be consumed and produced, because in these scenarios, we do not encounter multiple individual tokens with the same value on the same place. For the sake of brevity, we do not give the token selections formally, nor do we verify the enabling of transitions in detail. For a more detailed example of an AHLI net firing step, we refer to the one on page 82.

Creation and activation of clients for Bob and Carol This is carried out analogously to the previous two actions in AN_{SCPM}, leading to the marking (I_3, m_3) with the platform net ANI_3 shown in Fig. 7.50 on place p_1. The two new client core components for Bob and Carol are the same as Alice's one in Fig. 7.49 up to the suffices -B and -C instead of -A for the transition and place names and the ID token on the corresponding `User` place (see zoomed areas).

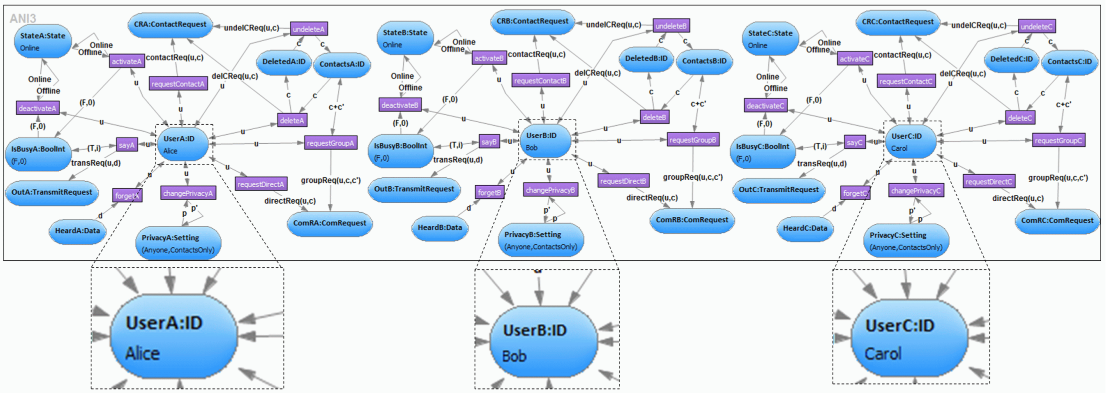


Figure 7.50.: Skype platform net ANI_3 with three activated clients for Alice, Bob, and Carol

7.3.2. Direct Channels and Contact Exchanges

After having registered, Alice gets in contact with Bob via Skype and sends him a short greeting message. Yet, she is not allowed to call Bob, because he has set his privacy option for incoming calls to "contacts only". To become allowed to call him, Alice asks for Bob's permission to add him to her contact list, which he grants.

Alice creates a direct channel to Bob She looks up Bob's contact in the Skype white pages directory and opens a window for an (yet) unauthorized contact as shown in Fig. 3.1, which offers several actions to Alice to communicate with Bob. In the platform net ANI_3, this user action is realized by a firing of Alice's transition *requestDirectA*, which produces a request token Bob's *ID* value, that is then handled by the token request handler expression

$$handler(directReq) = DiscardDirectReqExisting \parallel CreateDirectChannel$$

Formally, the firing and the request handling is calculated by a firing

$$(AN_{SCPM}, I_3, m_3) \xrightarrow{requesthandle, asg_3, S_3} (AN_{SCPM}, I_4, m_4)$$

in the controlling AHOI net, where

$$\begin{aligned} asg_3 = \{&n \mapsto ANI_3, asg \mapsto \widehat{asg}_3, t \mapsto requestDirectA,\\ &s \mapsto \hat{S}_3, n' \mapsto ANI_3', req \mapsto directReq, os \mapsto (o_3)\} \end{aligned}$$

and $S_3 = (M_3, m_3, N_3, n_3)$ with $m_3(M_3) = \{(ANI_3, p_1)\}$ as the consumed and $n_3(N_3) = \{(ANI_4, p_1)\}$ as the created token.

First, we can see that *requestDirectA* can fire in ANI_3 (Fig. 7.50) with $\widehat{asg}_3 = \{u \mapsto Alice, c \mapsto Bob\}$ and a suitable token selection $\hat{S}_3$, resulting in the firing step

$$ANI_3 \xrightarrow{requestDirectA, \widehat{asg}_3, \hat{S}_3} ANI_3'$$

where ANI'_3 is the same as ANI_3 but with an additional request token $\mathit{directReq}(\mathit{Alice}, \mathit{Bob})$ on Alice's place `ComRA`. From this, we get for the firing conditions of *requesthandle* that $\overline{asg_3}(\mathit{isEnabled}(n, t, asg, s))$ and $\overline{asg_3}(\mathit{producesReq}(t, req))$ both evaluate to *true*.

For the handling of the request, we consider the possible matches of the priority list $\mathit{DiscardDirectReqExisting} \parallel \mathit{CreateDirectChannel}$ to ANI'_3, according to Def. 5.2.18: The rule *DiscardDirectReqExisting* in Fig. 7.8 cannot match in ANI'_3 because it would have to match the token value $\mathit{directReq}(u1, u2)$ on $\mathit{directReq}(\mathit{Alice}, \mathit{Bob})$, which demands a token variable assignment asg^t_4 (and the induced the rule match's algebra component) to match the token variables $u1$ on *Alice* and $u2$ on *Bob*. On the other hand, there is no transition between the *ID* client core places carrying the names *Alice* and *Bob* in ANI'_3 that could be matched by the transition *add1* (indicating an already existing direct channel). For the rule *CreateDirectChannel* in Fig. 7.7, there exists a unique match $o_3 \colon L_{\mathit{CreateDirectChannel}} \to \mathit{ANI}'_3$ that maps the places with the suffixes -1 and -2 on the places with the suffixes -*A* and -*B*, respectively, with exactly this mapping of the token variables $u1$ and $u2$. We conclude that (o_3) is a valid match sequence for $\mathit{handler}(\mathit{directReq})$ on ANI'_3 and therefore $\overline{asg_3}(\mathit{isHandleable}(n', req, os))$ evaluates to *true*.

By the rule application sequence (according to Def. 5.2.19)

$$\left(\mathit{ANI}'_3 \xRightarrow{\mathit{handler}(\mathit{directReq}),(o_3)} \mathit{ANI}_4\right) = \left(\mathit{ANI}'_3 \xRightarrow{\mathit{CreateDirectChannel},o_3} \mathit{ANI}_4\right)$$

we get that the marking (I_4, m_4) of the AHOI net AN_{SCPM} after this user action contains the net $\overline{asg_3}(\mathit{handle}(n', req, os)) = \mathit{ANI}_4$ shown in Fig. 7.51 on place p_1 rather than ANI_3. In Fig. 7.51, the client cores from ANI'_3 are marked with a dashed border and the new direct channel (cf. the RHS in Fig. 7.7) is marked with a solid box. Note that for the uniqueness of element names, the places and transitions of the newly created channel structure have the suffixes -*BA* if they belong to Bob, and -*AB* if they belong to Alice or if they share a purpose for both like the common place `HistoryAB` for the sent messages in this channel.

Alice sends a message to Bob For sending a message to Bob (via the direct channel), Alice can fire her transition *sendIMAB* in ANI_4. Analogously to activating Alice's client on page 229, a firing in the platform AHLI net is realized by a firing of the higher-order transition *tokengame* (see Fig. 5.8)

$$(AN_{SCPM}, I_4, m_4) \xrightarrow{\mathit{tokengame},asg_4,S_4} (AN_{SCPM}, I_5, m_5)$$

with the variable assignment $asg_4 = \{n \mapsto \mathit{ANI}_4, t \mapsto \mathit{sendIMAB}, asg \mapsto \widehat{asg}_4, s \mapsto \hat{S}_4\}$ and the token selection S_4 that is analogous to S_1 for this firing step. For the variable assignment

$$\widehat{asg}_4 = \{u \mapsto \mathit{Alice}, c \mapsto \mathit{Bob}, d \mapsto \mathit{Hello}, h \mapsto \mathit{Empty}, o \mapsto \mathit{ContactsOnly}\},$$

there is a suitable token selection $\hat{S}_4$ under which *sendIMAB* is enabled. The assignment $\widehat{asg}_4$ indicates that Alice (variable u) sends to Bob (variable c) the message "Hello"

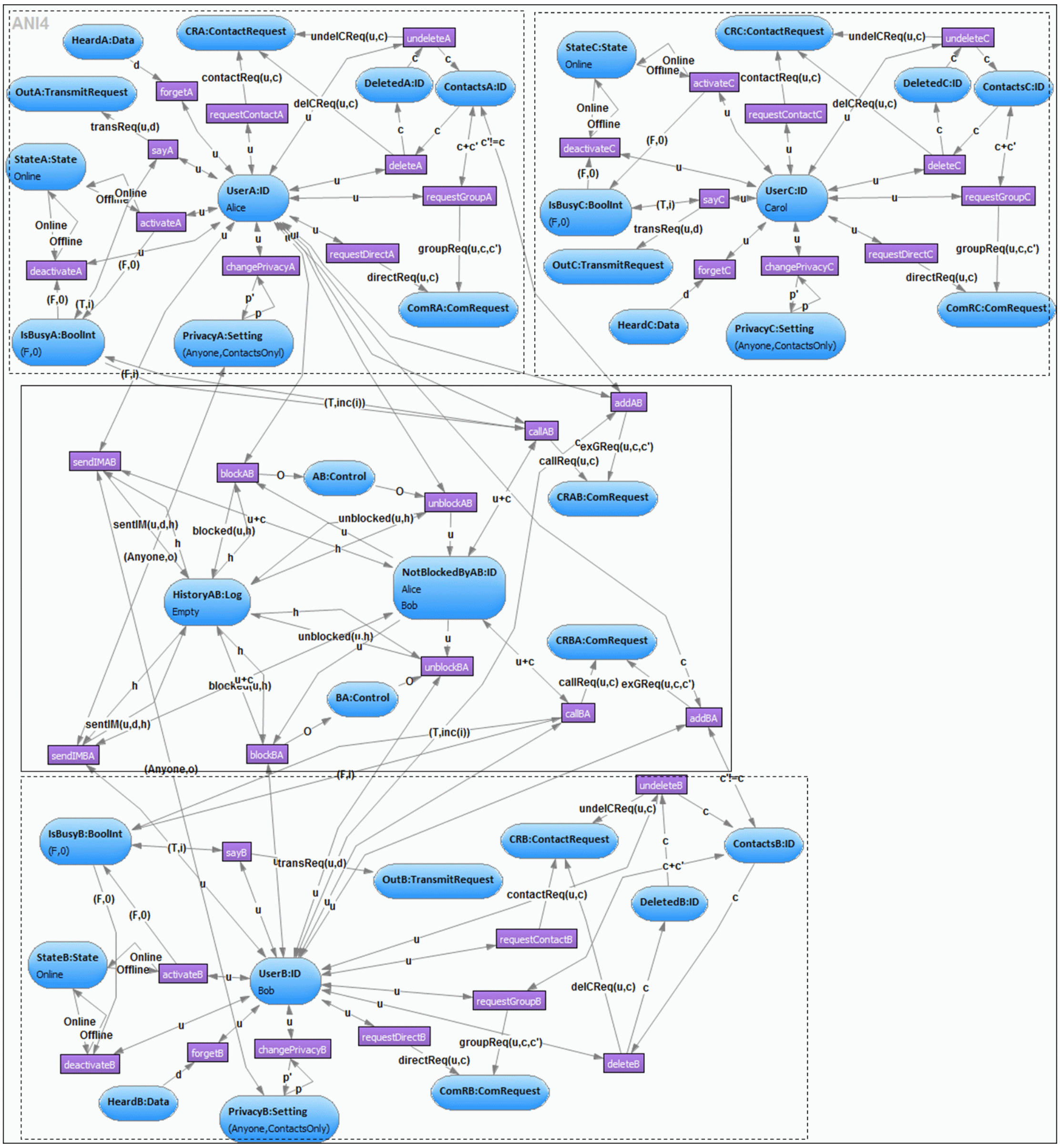


Figure 7.51.: Skype platform net ANI_4 with direct channel between Alice and Bob (in solid box)

(variable d). The variable o is just a free variable to match any privacy setting option that Bob has set for incoming calls, whose exact value is not important for this step as long as Bob's place `PrivacyB` carries a *Setting* token that has as first element (for incoming messages) the *Option* value *Anyone*, which is the case in ANI_4. Analogous to the step for activating Alice's client on page 229, $(tokengame, asg_1)$ is a consistent transition assignment for (AN_{SCPM}, I_4, m_4) because the transition *sendIMAB* in ANI_4 is enabled with $\widehat{asg}_4$ under the token selection $\hat{S}_4$. The AHLI platform net ANI_5 resulting from the firing step

$$ANI_4 \xrightarrow{sendIMAB,\widehat{asg}_4,\hat{S}_4} ANI_5$$

has the same marking as ANI_4 in Fig. 7.51 but with the token value $sentIM(Alice, Hello, Empty)$ on the place `HistoryAB` instead of the $Empty$ history. With this, we consider the data transmitted and available to Bob. Of course, the marking (I_5, m_5) of AN_{SCPM} contains the platform net ANI_5 instead of ANI_4 on place p_1, analogous to the previous steps.

Alice requests contact exchange from Bob The action button "Add to Contacts" in Alice's interface window for the unauthorized contact to Bob in Fig. 3.1 is represented in the AHLI net by a firing step

$$ANI_5 \xrightarrow{requestContactA,\widehat{asg}_5,\hat{S}_5} ANI_5'$$

where $\widehat{asg}_5 = \{u \mapsto Alice, c \mapsto Bob\}$. This produces a request token $contactReq(Alice, Bob)$ on Alice's place `CRA` in the follower net ANI_5'. The firing of $requestContactA$ and the token request handling in ANI_5' again is performed by a firing in the controlling AHOI net

$$(AN_{SCPM}, I_5, m_5) \xrightarrow{requesthandle,asg_5,S_5} (AN_{SCPM}, I_6, m_6)$$

of which we do not regard further details of asg_5 and S_5, as they are analogous to the previous firing step of $requesthandle$. Instead, we concentrate on the handler expression

$$handler(contactReq) = DiscardContactExchange \parallel CreateContactExchange$$

and a possible valid match sequence os_5 on ANI_5' for this (assigned by $asg_5(os) = os_5$ to let the firing in AN_{SCPM} calculate the handling result, analogous to ANI_4 before).

As we expect, the rule $DiscardContactExchange$ in Fig. 7.11 is not applicable to ANI_5', because no part of the disjunctive application condition ac can be satisfied. A valid match would have to match the places `UserA` and `CRA` because of the necessary token variable assignment $asg_5^t(u1) = Alice$ and the implicit application condition expressed by the dashed box in the LHS, expressing that these places correspond to the places of a client core. Then, the subconditions $\exists a_1$ and $\exists a_2$ cannot be satisfied, because Alice does not have Bob's contact on her place `ContactsA` nor on `DeletedA`. Also, there does not yet exist a contact exchange structure between Alice's and Bob's clients so that $\exists a_3$ cannot match a transition for $deny2$. Therefore, we skip this rule and look for matches for the following lower-prioritized one.

The next rule $CreateContactExchange$ in Fig. 7.10 can be matched by a (unique) morphism $o_5 \colon L_{CreateContactExchange} \to ANI_5'$ so that we get a valid match sequence $os_5 = (o_5)$ and the transformation sequence

$$\left(ANI_5' \xRightarrow{handler(contactReq),(o_5)} ANI_6\right) = \left(ANI_5' \xRightarrow{CreateContactExchange,o_5} ANI_6\right)$$

with ANI_6 as shown in Fig. 7.52 where the structure for the contact exchange (cf. the RHS in Fig. 7.10) is marked with a solid box. Again, ANI_6 is the new platform AHLI net that replaces ANI_5 on place p_1 in the marking (I_6, m_6) of AN_{SCPM}.

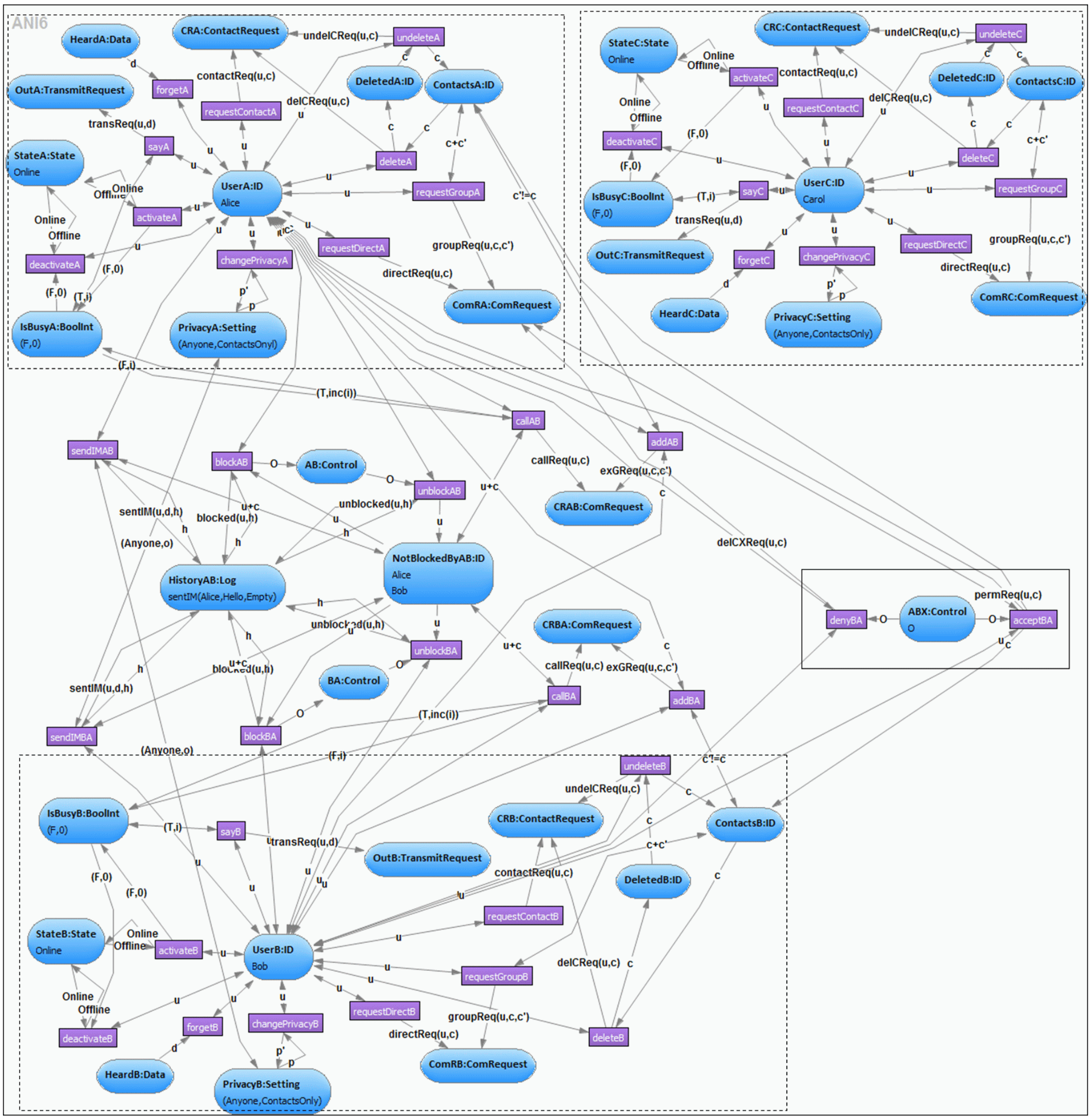


Figure 7.52.: Skype platform net ANI_6 with contact exchange structure between Alice and Bob (in solid box)

Bob accepts Alice's request To complete the contact exchange, we consider the firing step in the AHOI net

$$(AN_{SCPM}, I_6, m_6) \rightarrowtail^{requesthandle, asg_6, S_6} (AN_{SCPM}, I_7, m_7)$$

that first calculates the firing step

$$ANI_6 \rightarrowtail^{acceptBA, \widehat{asg}_6, \hat{S}_6} ANI_6'$$

representing Bob's action that he accepts Alice's request, and then handles the produced request token $permReq(Bob, Alice)$ on Alice's place `CRA` (cf. the contact exchange structure in the RHS in Fig. 7.10) in the result ANI'_6 with the following handler expression. (Note that Alice and Bob already have the other's contact on their contact places *ContactA* and *ContactB* in ANI'_6.)

$$\begin{aligned} \mathit{handler}(\mathit{permReq}) =& \mathit{DeleteOppositeExchange}; \\ & (\mathit{ExtendToPermittedChannel} \parallel \mathit{CreatePermittedChannel}) \end{aligned}$$

The rule *DeleteOppositeExchange* in Fig. 7.13 cannot match a corresponding structure in ANI'_6. As the rule name suggests, it requires that there exist a contact exchange structure in the opposite direction, i. e., in this case, this would rather be two transitions *denyA* and *acceptA* for Alice than *denyB* and *acceptB* for Bob, which indeed exist in ANI'_6. We skip this rule and continue with matching the remaining priority list of two rules.

Because there already exists a direct channel between Alice and Bob in ANI'_6, the rule *ExtendToPermittedChannel* in Fig. 7.14 is applicable at the (unique) match $o_6 \colon L_{\mathit{ExtendToPermittedChannel}} \to ANI'_6$ that matches the places and transitions with suffix -1 to the corresponding elements of Alice with suffix -A and the ones with suffix -2 to the ones of Bob with suffix -B. With this match, we do not need to further regard the lower-prioritized rule *CreatePermittedChannel* and have a valid match sequence (o_6) for this handler. The result of the handling transformation sequence

$$\left(ANI'_6 \xRightarrow{\mathit{handler}(\mathit{permReq}),(o_6)} ANI_7\right) = \left(ANI'_6 \xRightarrow{\mathit{ExtendToPermittedChannel},o_6} ANI_7\right)$$

is the platform net ANI_7 shown in Fig. 7.53 (for the marking (I_7, m_7) on place p_1 of the controlling AHOI net AN_{SCPM}), where the exchange structure between Alice's and Bob's client has been deleted and the transitions *sendIMAB* and *sendIMBA* do not longer depend on the opposite user's privacy settings.

Bob and Carol exchange contacts Analogously to the previous two user actions as firings of *requesthandle* in the control net AN_{SCPM}, Bob and Carol perform a contact exchange, which leads to the marking (I_8, m_8). We do not show this configuration explicitly, where, compared to (I_7, m_7), Bob and Carol now have a token of the other's *ID* name value on their corresponding contact list place, too.

7.3.3. Direct Calls

Now that Alice is allowed to call Bob, she starts a call, in which Bob joins. Alice says something to Bob and hangs up.

Alice starts a direct call to Bob To call Bob, Alice fires her transition *callAB* in the direct channel in ANI_8 between her and Bob. The firing

$$ANI_8 \xrightarrow{\mathit{callAB},\widehat{asg}_8,\hat{S}_8} ANI'_8$$

with the assignment $\widehat{asg}_8 = \{u \mapsto \mathit{Alice}, c \mapsto \mathit{Bob}, i \mapsto 0\}$ consumes the token $(F, 0)$ on Alice's place `IsBusyA` and puts the token $(T, 1)$ instead, indicating that Alice has one

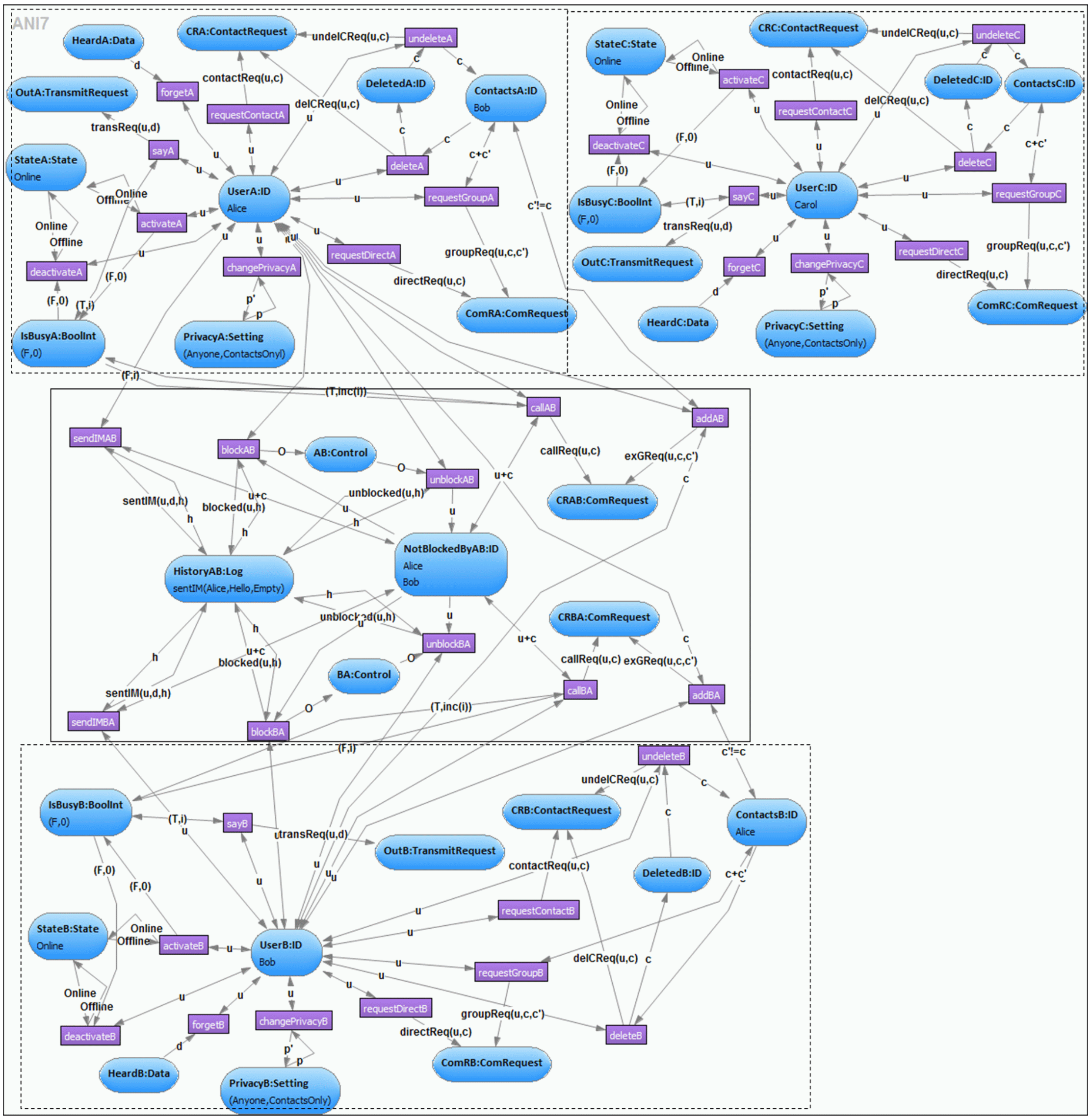


Figure 7.53.: Skype platform net ANI_7 with exchanged contacts and permitted channel between Alice and Bob

ongoing call and that it is true that that she is actively participating in this call. Note that the transition *callAB* requires to read two tokens from the place `NotBlockedByAB`, which indicates that neither Alice has blocked Bob nor vice versa, which is the case in ANI_7. Moreover, the firing produces a request token $callReq(Alice, Bob)$ on the place $CRAB$ that needs to be handled. This means, that this user action is realized by a firing of the transition *requesthandle* in (AN_{SCPM}, I_8, m_8), whose details are analogous to the previous steps and whose result contains the result of the application of the following request handler to ANI_8', again.

$$handler(callReq) = DiscardCallDirect \parallel CallDirect$$

A valid match $o\colon L_{\mathit{DiscardCallDirect}} \to \mathit{ANI}'_8$ for the rule *DiscardCallDirect* in Fig. 7.20 would have to map the token variables $u1$ and $u2$ to *Alice* and *Bob*, respectively, because of the request token $\mathit{callReq}(\mathit{Alice}, \mathit{Bob})$. The first part $\exists a_1$ of this rule's application condition is not satisfies by o in ANI'_8, i. e., there is no valid match for $C1$, because Bob's client is not offline. For the other part $\exists(a_2, \neg\exists a_{21})$, there does exist a (unique) morphism $q_2\colon C2 \to \mathit{ANI}'_8$ such that $q_2 \circ a_2 = o$, because Bob has set his privacy settings for incoming calls to "ContactsOnly". But q_2 does not satisfy the continuation $\neg\exists a_{21}$ of this condition, because this condition requires that there do not exist a morphism $q_{21}\colon C21 \to \mathit{ANI}'_8$ such that $q_{21} \circ a_{21} = q_2$, which does exist because Bob has Alice's contact on his contact list. Therefore, *DiscardCallDirect* is not applicable to ANI'_8.

The lower-prioritized rule *CallDirect* in Fig. 7.21 can match the direct channel structure between Alice's and Bob's clients with a (unique) match $o_8\colon L_{\mathit{CallDirect}} \to \mathit{ANI}'_8$ with the following transformation sequence

$$\left(\mathit{ANI}'_8 \xRightarrow{\mathit{handler}(\mathit{callReq}),(o_8)} \mathit{ANI}_9\right) = \left(\mathit{ANI}'_8 \xRightarrow{\mathit{CallDirect},o_8} \mathit{ANI}_9\right)$$

and ANI_9 in Fig. 7.54 as the result of this user action (also in the marking of $\mathit{AN}_{\mathit{SCPM}}$ after firing *requesthandle*). In ANI_9, the direct channel between Alice and Bob has been enriched by the transitions in the solid box representing the possible actions in a direct call (cf. the RHS in Fig. 7.21). Note that the event of Alice calling has been recorded in the channel's log token on the place `HistoryAB`. We omit the direct channel from Bob to Carol's client and her client itself in this and the following diagrams illustrating configurations of the direct call between Alice and Bob.

Bob joins the call from Alice Although Alice has started a call to Bob in the platform net ANI_9, he still has to join the call to be able to hear what she says. Up to then, we consider his client ringing as an ordinary telephone would do before picking up. This user action consists of firing Bob's transition *joinAB* in the call structure in the step

$$\mathit{ANI}_9 \xrightarrow{\mathit{joinBA},\widehat{\mathit{asg}}_9,\hat{S}_9} \mathit{ANI}_{10}$$

with $\widehat{\mathit{asg}}_9 = \{u \mapsto \mathit{Bob}, i \mapsto 1\}$, which on the one hand replaces the *false* token value on the place *HasJoinedBA* by *true* to indicate that Bob has joined and which on the other hand replaces the token $(F, 1)$ on place *IsBusyB* by $(T, 1)$ indicating that Bob now is busy in the one call he is involved in ANI_{10} (not shown explicitly). In the controlling AHOI net $\mathit{AN}_{\mathit{SCPM}}$, this is calculated by a suitable firing of the transition *tokengame*.

Alice talks to Bob Alice suddenly remembers that she has something else to do and wants to tell Bob that she will call again. In ANI_{10}, this is realized by a firing of Alice's transition

$$\mathit{ANI}_{10} \xrightarrow{\mathit{sayA},\widehat{\mathit{asg}}_{10},\hat{S}_{10}} \mathit{ANI}'_{10}$$

with $\widehat{\mathit{asg}}_{10} = \{u \mapsto \mathit{Alice}, i \mapsto 1, d \mapsto \text{“}\mathit{Sorry, call\ you\ later!}\text{''}\}$ producing the request token $\mathit{transReq}(\mathit{Alice}, \text{“}\mathit{Sorry, call\ you\ later!}\text{''})$ on Alice's place `OutA` in ANI'_{10}. The following handling of this request token takes care of transmitting the spoken data to Bob's

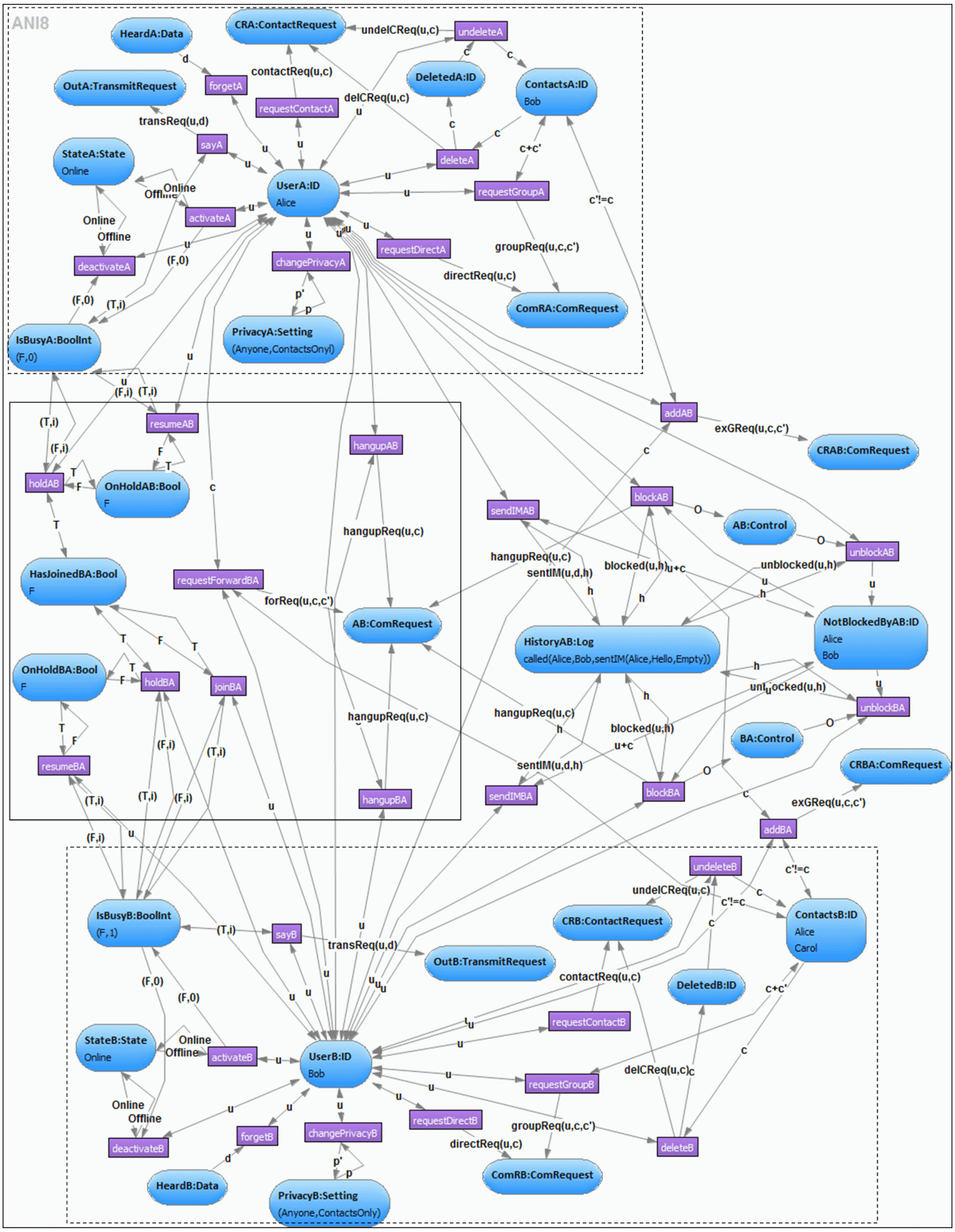


Figure 7.54.: Skype platform net ANI_9 with Alice calling Bob (without Carol)

client.

$$handler(transReq) = TransmitCall \parallel TransmitConference$$

The first rule *TransmitCall* in Fig. 7.46 can match ANI'_{10} at a (unique) match o_{10}, because we can find the structure in the net $C1$ of the application condition $\exists a_1$ in ANI'_{10}, i. e., an ongoing call between the user who produced the request token and some other user, where Bob is the latter in this case. Neither of them has put this call on hold, because of the *true* tokens on the places `onHoldBA` and `OnHoldBA`), and Bob has joined the call in the previous user action, which is required by the *true* token on the place `HasJoined2` in the LHS. Moreover, the place `Heard2` in the rule is determined by the application condition to belong to the client of the user who is connected to $u1$ by the call structure. Therefore, the result of this handling and of Alice's user action is

$$\left(\mathit{ANI}'_{10} \xRightarrow{handler(transReq),(o_{10})} \mathit{ANI}_{11}\right) = \left(\mathit{ANI}'_{10} \xRightarrow{TransmitCall,o_{10}} \mathit{ANI}_{11}\right)$$

with the platform net ANI_{11}, where the request token has been removed and Bob has got the data token $d1$ with the matched value "*Sorry, call you later*!" on his place `HeardB` (not shown explicitly, see marking in Fig. 7.55).

Alice hangs up After this short "conversation", Alice ends the call with the firing step

$$\mathit{ANI}_{11} \xrightarrowtail{hangupAB,\widehat{asg}_{11},\hat{S}_{11}} \mathit{ANI}'_{11}$$

where $\widehat{asg}_{11} = \{u \mapsto \mathit{Alice}, c \mapsto \mathit{Bob}\}$, which produces the request token $\mathit{hangupReq}(\mathit{Alice},\mathit{Bob})$ on the place AB in the call structure in ANI_{11}. For the corresponding handler expression

$$handler(hangupReq) = HangupDirectCall$$

and its single rule *HangupDirectCall* in Fig. 7.22, we find a valid (unique) match $o_{11}\colon L_{HangupDirectCall} \to \mathit{ANI}'_{11}$ that matches the call structure between Alice's and Bob's client. The result of the handling

$$\left(\mathit{ANI}'_{11} \xRightarrow{handler(hangupReq),(o_{11})} \mathit{ANI}_{12}\right) = \left(\mathit{ANI}'_{11} \xRightarrow{HangupDirectCall,o_{11}} \mathit{ANI}_{12}\right)$$

is the platform net ANI_{12} shown in Fig. 7.55, where the call structure has been removed and the direct channel has been restored to the structure before the call. Note that the new busy statuses of Alice and Bobs, the tokens $(F, 0)$ on their places `IsBusyA` and `IsBusyB`, follow from the assignment of false to the token variables $b1$ and $b3$ (because neither Bob nor Alice have put the call on hold) and 0 to $i1$ and $i2$, so that $(and(b1, b2), i1)$ evaluates to $(F, 0)$ for Alice and Bob.

7.3.4. Groups

We conclude the use case scenarios with the application of an amalgamated rule that multicasts a message from Bob to Alice and Carol. For this, he first has to create a Skype group as the channel that transmits the message.

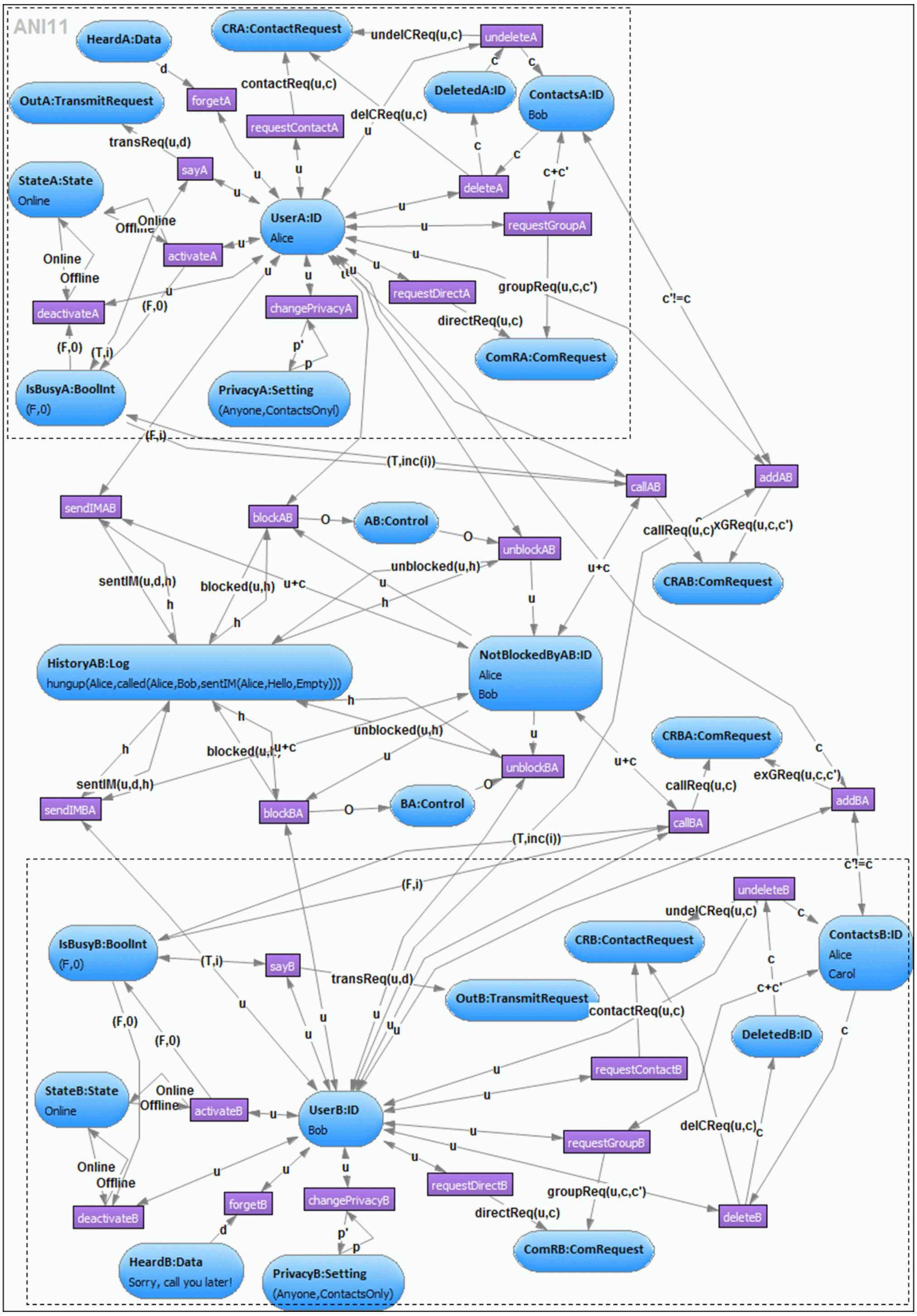


Figure 7.55.: Skype platform net ANI_{12} after Alice calling Bob (without Carol)

Bob founds a group with Alice and Carol The creation of a group structure is triggered by the firing of Bob's transition

$$ANI_{12} \xrightarrow{requestGroupB,\widehat{asg}_{12},\hat{S}_{12}} ANI'_{12}$$

where $\widehat{asg}_{12} = \{u \mapsto Bob, c \mapsto Alice, c' \mapsto Carol\}$. The produced request token $groupReq(Bob, Alice, Carol)$ on the place $ComRB$ in Bob's client core in ANI_{12} is handled by a the handler expression

$$handler(groupReq) = CreateGroup$$

and the rule $CreateGroup$ in Fig. 7.27. There is a (unique) match $o_{12}\colon L_{CreateGroup} \to ANI'_{12}$ that matches Bob's client with the request token (for the elements with name suffix -1 because $u1$ is mapped to Bob), Alice's client (suffix -2 with $u2$ mapped to $Alice$), and Carol's client (suffix -3 with $u3$ mapped to $Carol$). The result of the handling

$$\left(ANI'_{12} \xRightarrow{handler(groupReq),(o_{12})} ANI_{13}\right) = \left(ANI'_{12} \xRightarrow{CreateGroup,o_{12}} ANI_{13}\right)$$

is depicted in Fig. 7.56, where the new group structure is marked with a solid box. Note that the direct channels between Bob' client and the other two clients are not shown in this diagram; they are insignificant for the next step, where Alice sends a message to the group.

Alice sends a group message to Bob and Carol Alice fires her transition $sendIMA$ in Bob's new group in the step

$$ANI_{13} \xrightarrow{sendIMA,\widehat{asg}_{13},\hat{S}_{13}} ANI'_{13}$$

with $\widehat{asg}_{13} = \{u \mapsto Alice, d \mapsto \text{“}Hello\ everybody!\text{''}, h \mapsto Empty\}$. This replaces the log token $Empty$ with $sentIM(Alice, \text{“}Hello\ everybody!\text{''}, Empty)$ on Alice's history place `HistoryA` in this group so that she can access her sent message. The transmission of the message to Bob and Carol as a handling reconfiguration is triggered by the request token $transGReq(Alice, \text{“}Hello\ everybody!\text{''})$ that is produced by this firing step on the group place `IMABC`. The respective request handler expression

$$handler(transGReq) = SendGroupMessage$$

consists of the interaction scheme $SendGroupMessage = (s_1\colon \varrho_K \to \varrho_M)$ in Fig. 7.28, for which we construct a maximal disjoint matching into ANI'_{13}, according to Def. 5.2.14.

First, we find for the scheme's kernel rule ϱ_K, shown in the blue box in Fig. 7.28, the (unique) kernel match $\hat{o}_0\colon L_{\varrho_K} \to ANI'_{13}$. This match maps the token variable $u1$ to $Alice$ and the transition $sendIM1$ to Alice's $sendIMA$ (which just has fired) in Bob's group. Note that the token $u2$ is part of the multi rule ϱ_M, which we instantiate twice, because we find for the multi rule two occurrences in ANI'_{13} that are compatible with the kernel match $\hat{o}_0$. The first possible multi match $\hat{o}_1\colon L_{\varrho_M} \to ANI'_{13}$ maps the transition $sendIM2$ to Bob's transition $sendIMB$ in his group, which implies that the identifier variable $u2$ is mapped to Bob by $\hat{o}_1$. The second possible multi rule match is $\hat{o}_2\colon L_{\varrho_M} \to ANI'_{13}$, which maps the transition $sendIM2$ to Carols's transition $sendIMC$ in Bob's group and, accordingly, $u2$ to $Alice$. There are no other possible matches for the multi rule ϱ_M and the matches $\hat{o}_1$ and $\hat{o}_2$ overlap only in the elements which are

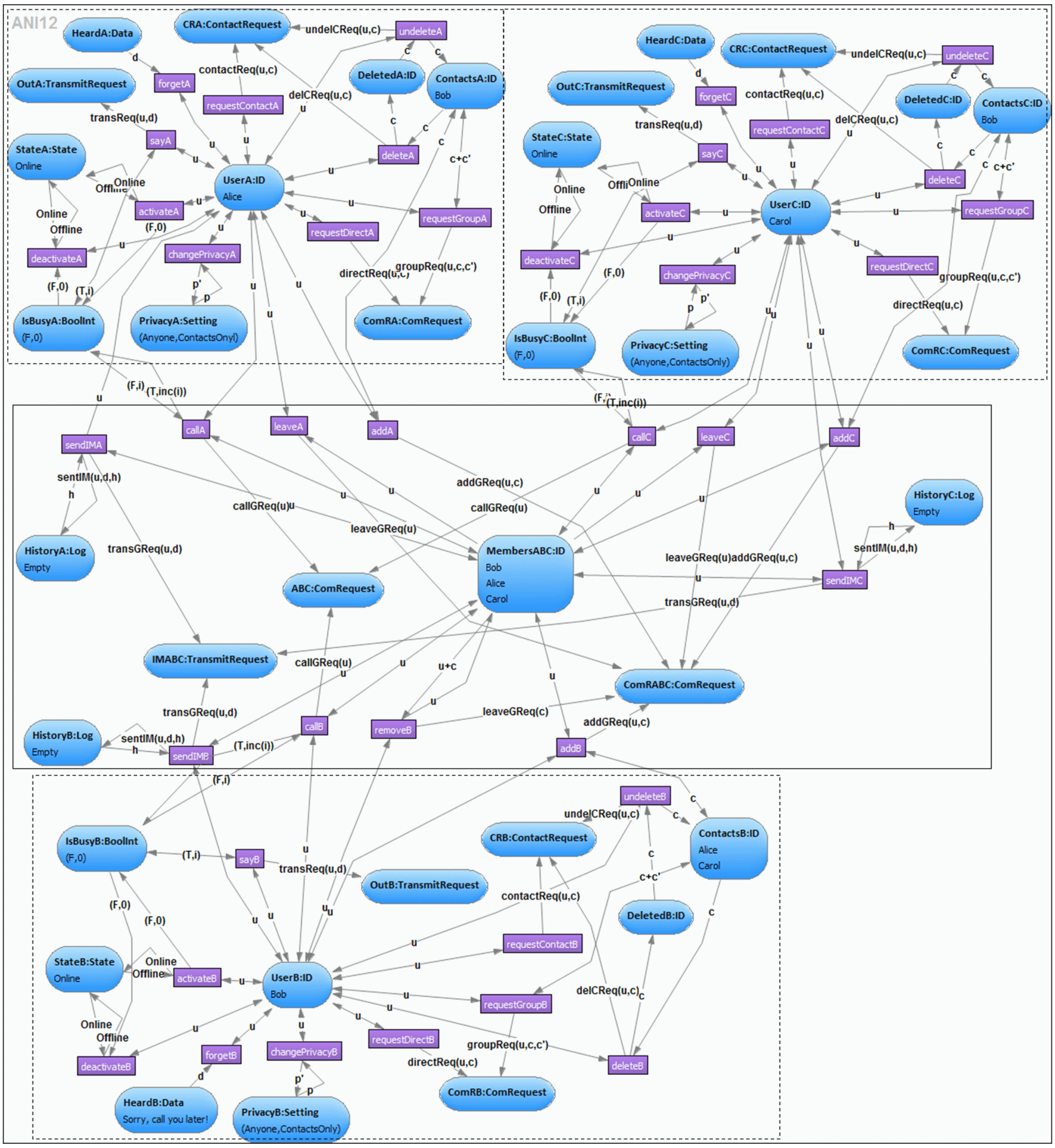


Figure 7.56.: Skype platform net ANI_{13} with Bob's group (without direct channels)

matched by the kernel match $\hat{o}_0$, too. Therefore, $o_{13} = (\hat{o}_0, \hat{o}_1, \hat{o}_2)$ is a valid maximal disjoint matching of *SendGroupMessage* in ANI'_{13}.

According to the construction in Def. 5.2.15, we apply the interaction scheme *SendGroupMessage* at the maximal match o_{13} as follows: We construct the loosely amalgamated rule $\tilde{\varrho}_{(s_1,s_1)}$ shown in Fig. 7.57 over the bundle of instantiated multi rules (s_1, s_1) that results from the maximal matching o_{13} via the colimit construction described in Def. 5.2.13.

The multi rule matches $\hat{o}_1$ and $\hat{o}_2$ are consistent with the kernel match $\hat{o}_0$, i. e., $\hat{o}_1 \circ s_1 =$

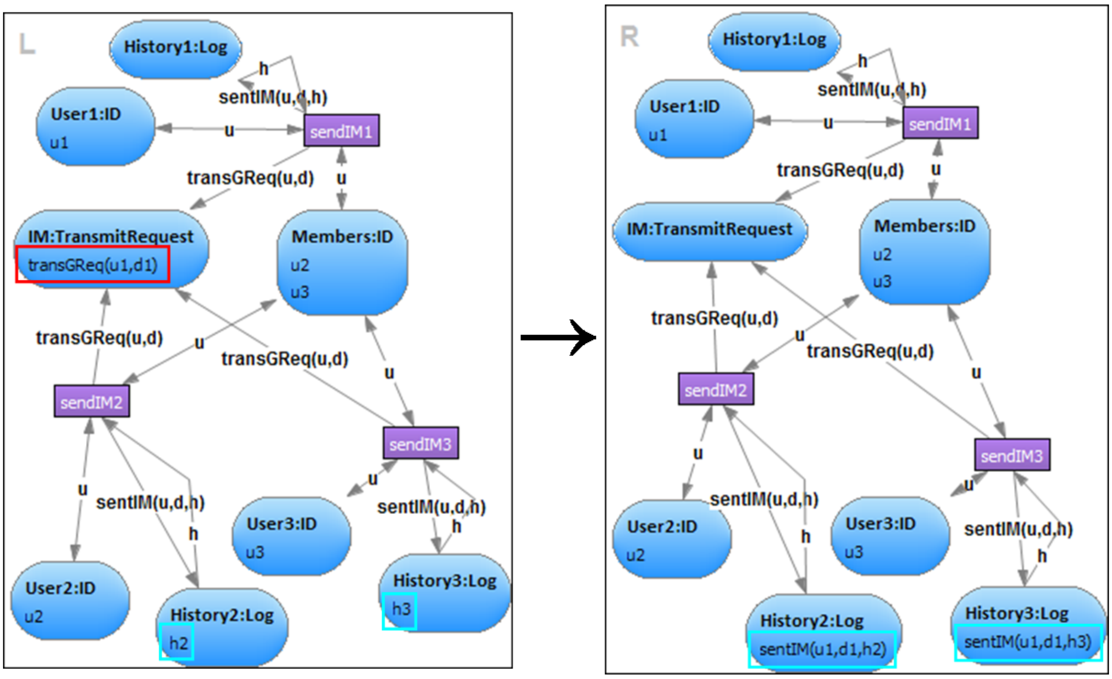


Figure 7.57.: Amalgamated rule $\tilde{\varrho}_{(s_1,s_1)}$ for maximal disjoint match o_{13} of *SendGroupMessage* in ANI'_{13}

$$\begin{tikzcd}
L_K \arrow[r, shift left, "s_{1,L}"] \arrow[r, shift right, "s_{1,L}"'] \arrow[rrr, bend right=40, "\hat{o}_0"'] & L_M \arrow[r, shift left, "t_{0,L}"] \arrow[r, shift right, "t_{1,L}"'] \arrow[rr, bend left, "\hat{o}_1"] \arrow[rr, bend right, "\hat{o}_2"'] \arrow[rr, phantom, bend left=15, "="] \arrow[rr, phantom, bend right=15, "="'] & \tilde{L}_{(s_1,s_1)} \arrow[r, "\tilde{o}_{13}"] & \mathit{ANI}'_{13}
\end{tikzcd}$$

Figure 7.58.: Match $\tilde{o}_{13} \colon \tilde{L}_{(s_1,s_1)} \to \mathit{ANI}'_{13}$ for amalgamated rule $\tilde{\varrho}_{(s_1,s_1)}$ induced by multi matches $\hat{o}_1$ and $\hat{o}_2$

$\hat{o}_0 = \hat{o}_2 \circ s_1$. From this and the LHS $\tilde{L}_{(s_1,s_1)}$ of $\tilde{\varrho}_{(s_1,s_1)}$ as a colimit object in Fig. 7.58 together with the morphisms $t_{0,L}$ and $t_{1,L}$), we get the induced morphism $\tilde{o}_{13}$ as match for the amalgamated rule. According to $\hat{o}_1$ and $\hat{o}_2$, $\tilde{o}_{13}$ matches the transition *sendIM1* to Alice's ($u1$) transition *sendIMA*, transition *sendIM2* to Bob's ($u2$) transition *sendIMB*, and transition *sendIM3* to Carol's ($u3$) transition *sendIMC*. Therefore, the following rule application distributes the message sent by Alice to the history places of Bob and

Carol:

$$\left(ANI'_{13} \xRightarrow{handler(transGReq),(o_{13})} ANI_{14}\right) = \left(ANI'_{13} \xRightarrow{SendGroupMessage,o_{13}} ANI_{14}\right)$$
$$= \left(ANI'_{13} \xRightarrow{\tilde{\varrho}(s_1,s_1),\tilde{o}_{13}} ANI_{14}\right)$$

7.4. Formulation and Validation of Properties in the Skype Platform Model

We apply the validation concepts and results from Chap. 6 to the example activities of the previous section to validate the Skype model properties from Sect. 3.1.7 — and with this also exemplarily for their generalizations to Communication Platforms in Sect. 3.2.1. First, we check the model integrity of the platform nets ANI_1 to ANI_{14} from the example activities w. r. t. to structural constraints representing required properties of a Skype system. Then, we verify that the performed user actions leading from ANI_1 to ANI_{14} have been successful and functional to show that in these use cases the model is sound, i. e., the requests have been completely processed and the platform evolution depends only on the triggering firing steps (without nondeterminism in the request handlings and their outcome). Finally, we exemplarily discuss the degrees of (in)dependence that reflect our modeling intentions w. r. t. Skype for several performed user actions, and we analyze whether they comply to the actual (in)dependencies we find between the user actions in the formal platform model.

7.4.1. Constraints for Model Integrity

For the integrity-related properties of Skype models described in Sect. 3.1.7, we define platform constraints according to Sect. 6.1 (in the graphical representation used in this case study) and check that the platform nets ANI_1 to ANI_{14} satisfy them.

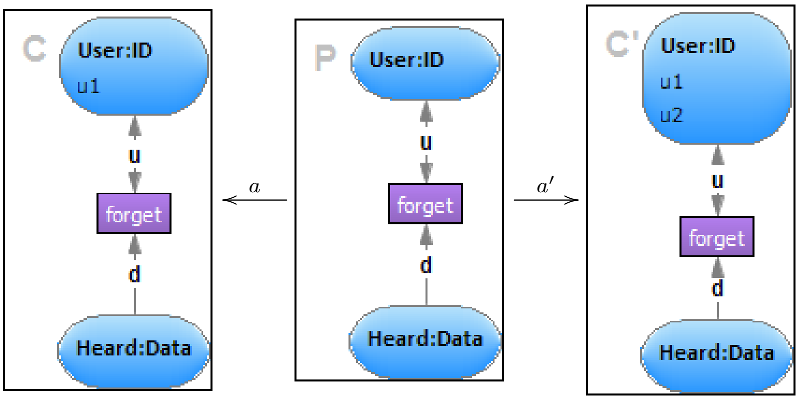
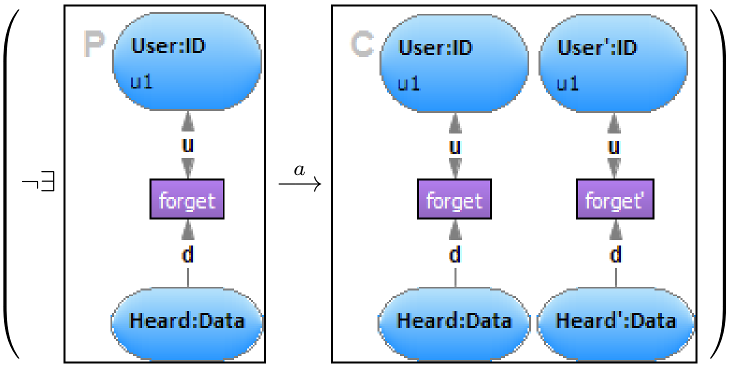


Figure 7.59.: Morphisms of constraint ac_1 for unambiguous user identities

Unique and unambiguous user names The two example constraints for a minimalistic platform in Sect. 6.1.2 describe the model-independent properties that every identifier name is assigned to a unique user and that every user has a unique identifier. For our Skype model, in which client core structures look as in Fig. 7.3, we adapt these constraints because it is sufficient to match the characteristical transition *forget* to locate the identifier place of such a client core. With this, we get the constraints $ac_1 = \forall\,(\exists a \wedge \neg\exists a')$ with the morphisms shown in Fig. 7.59 and

$ac_2 = \forall$


The initial (empty) platform net ANI_0 satisfies both constraints because there does not exist any morphism $P \to \mathit{ANI}_0$ for each. Therefore, we have that "for all" morphisms $P \to \mathit{ANI}_0$ there does not exist a morphism $C \to \mathit{ANI}_0$ compatible to a (and analogously a morphism $C' \to \mathit{ANI}_0$ for ac_1) so that $\mathit{ANI}_0 \models_S ac_1$ and $\mathit{ANI}_0 \models_S ac_2$.

For the platform nets ANI_1 and ANI_2 containing just a client core for Alice, there does only exist one morphism each that matches C to the central *ID* place of this net, similar to the morphism p_1 described on page 144 in Sect. 6.1.2). On the other hand there do not exist morphisms from C' into ANI_1 or ANI_2, which proves that ANI_1 and ANI_2 satisfy ac_1. For the satisfaction of ac_2, we see that there cannot exist any morphism $C \to \mathit{ANI}_0$ that would violate ac_2, because there is no other second place with an *ID* token *Alice* on it.

The same holds for the two other clients of Bob and Carol created in ANI_3 for similar morphisms and of course for Alice's client. The marking on the central *ID* place does not change and no other transition that can be matched by *forget* is created in any of the later platform nets so that they all satisfy ac_1 and ac_2.

Exchanged contacts We require that an entry in the contact list always resulted from a proper contact exchange. The constraint $ac_3 = \forall\,(\exists a \vee \exists a')$ with the morphisms shown in Fig. 7.60 expresses this structurally by the condition that if one user has another contact on his contact list place, then the other user has the contact of the first user on his contact list as well (by $\exists a$) or at least on his place of deleted contacts (by $\exists a'$).

This is easy to verify for the platform nets ANI_0 to ANI_6 (see Fig. 7.52) where no client has any other contact, neither on its contact list nor on its deleted contacts place. Hence, there does not exist a morphism from P in to these nets for which the subcondition $\exists a \vee \exists a'$ could be violated. For the later nets up from ANI_7 (see Fig. 7.55 as a representative), we find a morphism from P into these nets for each contact token on Alice's, Bob's, and

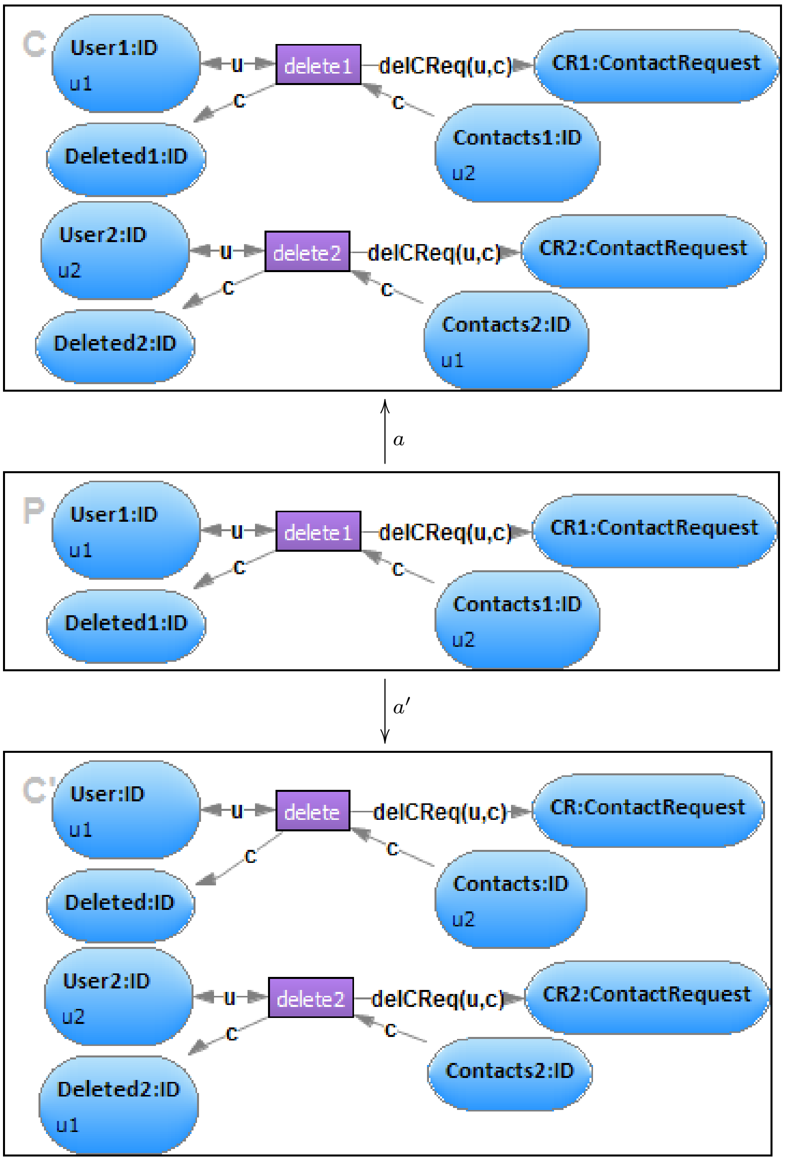


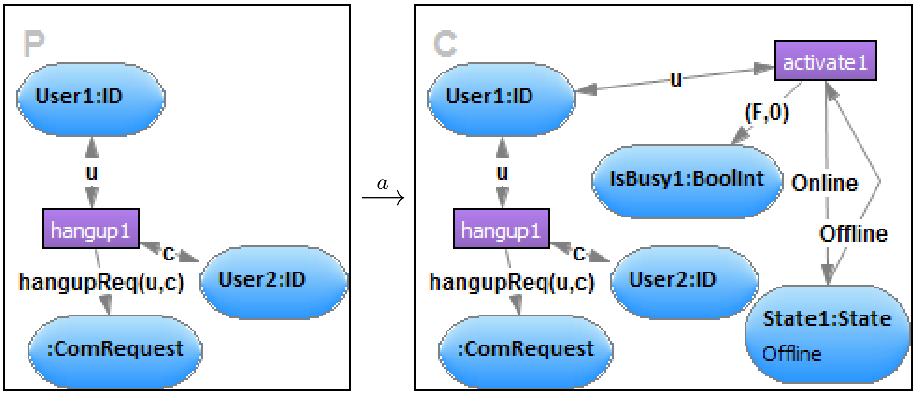
Figure 7.60.: Morphisms of constraint ac_3 for exchanged contacts

Carol's contact list (because they never delete a contact in the example). For each of these morphisms, we find a corresponding morphism that matches C with the appropriate contact list token in the opposite client such that $\exists a$ is satisfied.

State-related exclusion of user actions A necessary precondition for the possibility of performing a user action is its availableness in the user interfaces. Because user actions are triggered by transition firings, we characterize available user actions by the existence of corresponding transitions. The following two structural constraints express the conditions that certain user actions must (not) be available in certain model or client states.

The condition that a user can only be in a call if his client is not offline is formalized by the constraint

$ac_4 = \forall \Big(\neg\exists$

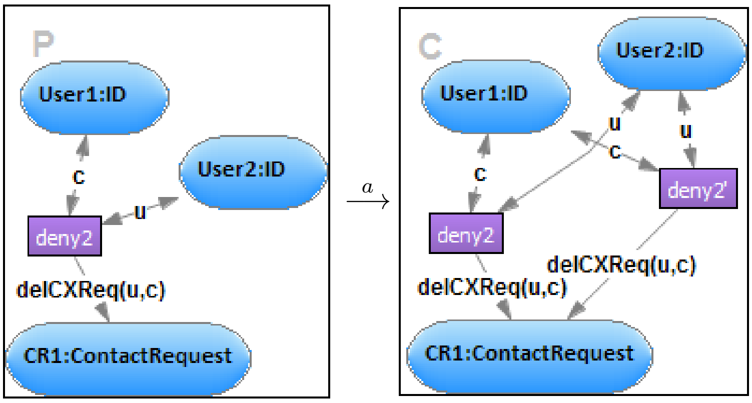


$\Big)$

where the transition *hangup1* identifies the action of hanging up a direct call (cf. rule *HangupDirectCall* in Fig. 7.22). If this action is available to a matched user client, then we forbid that there exist a token *Offline* on the *State* place of this client. Except for the platform nets *ANI₁* and the omitted intermediate steps between *ANI₂* and *ANI₃*, there is no client being offline at any time in the example scenarios so that for every possible morphism $P \to ANI_i$ indeed there does not exist a (compatible) morphism $C \to ANI_i$. In *ANI₁*, there is obviously no transition for *hangup1* to match (see Fig. 7.48) and hence no ongoing call. Therefore all nets in the example scenarios satisfy ac_4.

Similarly, in the constraint

$ac_5 = \forall \Big(\neg\exists$



$\Big)$

a request for contact exchange is identified by a transition such as *deny2* or *deny2′* as created by the rule *CreateContactExchange* in Fig. 7.10. It is easy to verify for *ANI₆* in Fig. 7.52, which is the only platform net form the examples for which a match of the transition *deny2* in P for ac_5 exists, that there does not exists a compatible match for C (which would require a second transition *deny2′*).

Role constraints The roles of a Skype user in a certain context such as “group owner” or “conference host” is determined by his possible characteristical actions in this context,

e. g., the action to remove a member from this particular group or to quit the conference completely (instead just hanging up as a regular participant). With this identification of roles by actions and hence by representing transitions, constraints for limiting user roles in certain contexts can be formalized similarly to ac_5, which forbids that a transition *deny2'* exist if there already exists a transition as *deny2*.

For the condition that a group must have exactly one owner who is characterized by the action of removing group members, it is easy to conceive a constraint that requires for a group structure around a central group place (as `MembersABC` in ANI_{13}, see Fig. 7.56) that there exist exactly one transition like Bob's *removeB*. One may argue that such a constraint is not very useful for our Skype platform model, because the transition for removing a group member (such as *removeB*) is solely created by the rule *CreateGroup* in Fig. 7.27 and it does not occur in any other rule. But, once we have formalized this condition, we could go on to further extend the platform model perhaps by a user action for passing over the ownership of a group to another user. For such development of existing platform models, constraints like this are useful to be considered for validating even properties that obviously seem to be preserved in former platform model versions.

7.4.2. Success and Functionality of User Actions

For the example scenarios, we are interested in whether the reconfiguration requests have been executed successfully and in the functionality of these reconfigurations, i. e., the determinism imposed by the current platform state and the request tokens that are produced by the triggering firing steps. If we found for any of the user actions leading from the platform nets ANI_0 to ANI_{14} that there exist more than one possible match sequence or that the request token be not consumed, we should reconsider the handler expressions and rules w. r. t. to the points discussed in the end of Sect. 6.2.

All user actions in the example scenarios are functional according to Def. 6.2.1 on the corresponding platform net ANI_i in which they occur. For all rule applications in the handling reconfigurations there exists a unique match, each, as stated in the simulation descriptions of the user actions in Sect. 7.3.

For validating the success of these user actions, we can directly use the general constraint formulated in Sect. 6.1.3. It turns out that all platform nets ANI_0 to ANI_{14} satisfy this constraint because their markings do not contain a token of any request sort.

7.4.3. Independence of User Actions

With the detailed descriptions of the possible actions in Skype in Sect. 3.1, we get an intuitive idea of the dependencies of user actions and their effects. For concrete scenarios such as the action sequence from ANI_0 to ANI_{14}, we can validate these expectations by applying the concepts of user action independence from Sect. 6.3.3 to the concrete user actions and by reasoning about a least degree of independence or dependence that these two actions should formally comply to (cf. Table 6.1). If the considered actions do not have the expected degree of (in)dependence, this is a strong indicator for at least one user action not respecting some vital model requirement.

In the following, we examine the actions of the example scenarios under the aspect of minimal independence and dependence degrees to be expected according to modeling requirements. The different formal degrees of independence and dependence correspond to the following intuitive ideas of relations between actions: If we expect that two concrete (functional) actions principally can occur in any order in the platform, we require that they be at least trigger independent. If we furthermore expect that the result be invariant w. r. t. the order, they should be weakly independent. Regarding modeling aspects for handler expressions, we may even require independence, which means that even the handling reconfigurations are not affected by the order of the actions. In contrast, we require that two actions be weakly dependent if we expect that a functional application of the handler expression depends on the order of the actions, or that they even be dependent if the result should not be the same. By requiring trigger dependence, we want to preclude any possibility to perform the actions in a different order. The following examination of the concrete user actions from Sect. 7.3 gives and discuss example justifications for requiring different degrees of independence or dependence — or sometimes both mixed.

7.4.3.1. Sequential Independence of Example Activites

We consider every pair of directly consecutive actions in the example user action sequence, for which we evidently can check the actual degree of sequential independence (see Def. 6.3.10), only. In the following, we denote with ua_i the user action $ANI_i \overset{t_i,\widehat{asg}_i,\hat{S}_i}{\rightarrowtail} ANI'_i \xRightarrow{handler(req_i),(o_i)} ANI_{i+1}$ (or $ANI_i \overset{t_i,\widehat{asg}_i,\hat{S}_i}{\rightarrowtail} ANI_{i+1}$, respectively) from Sect. 7.3, where t_i is the fired transition and req_i is the request constructor of the user action. We start with

$$ua_3 = \left(ANI_3 \overset{requestDirectA,\widehat{asg}_3,\hat{S}_3}{\rightarrowtail} ANI'_3 \xRightarrow{CreateDirectChannel,o_3} ANI_4 \right)$$

because the steps before ua_3 are paired with external actions for creating the clients.

$$\begin{tikzcd}
& & & & ANI_3 \arrow[dl, tail, "{requestDirectA,\widehat{asg}_3,\hat{S}_3}"'] \\
& ua_3 : & & ANI'_3 \arrow[dl, Rightarrow, "{CreateDirectChannel,o_3}"'] \arrow[dr, dashed] & \\
& & ANI_4 \arrow[dr, tail, "{sendIMAB,\widehat{asg}_4,\hat{S}_4}"'] & (\mathbf{2}) & ANI''_3 \arrow[dl, Rightarrow, dashed] \\
ua_4 : & & & ANI_5 &
\end{tikzcd}$$

Figure 7.61.: Sequentially trigger-dependent user actions ua_3 and ua_4

ua_3 and ua_4 (see page 235) are shown sequentially in the left of Fig. 7.61. Before we check the actual degree of dependence between these actions, we first think about to what

$$\begin{tikzcd}
L_3 \arrow[d, "o_3"'] & K_3 \arrow[l, "l_3"'] \arrow[r, "r_3"] \arrow[d] & R_3 \arrow[dr, "o'_3"] \arrow[drrr, dashed, "i_3"] & & L \arrow[dl, "o_t"'] \arrow[dlll, dashed, "i"'] & \\
ANI'_3 & D_3 \arrow[l, "f_3"] \arrow[rr, "g_3"'] & & ANI_4 \arrow[rr, tail, "{sendIMAB,\widehat{asg}_4,\hat{S}_4}"] & & ANI_5
\end{tikzcd}$$

Figure 7.62.: Condition for sequential independence of transformation $(\mathit{CreateDirectChannel}, o_3)$ and firing of $(\mathit{sendIMAB}, \widehat{asg}_4, \hat{S}_4)$

extent we expect these concrete actions should be dependent according to our modeling intentions. The user action ua_3 requests and performs the creation of a direct channel from Alice to Bob. The following user action ua_4 sends a message from Alice to Bob. In the discussion about modeling requirements for the Skype case study in Sect. 7.1.2, we bind the sending of messages to a direct channel context. Therefore, we require that the action ua_4 cannot occur before ua_3, i. e., they be sequentially trigger-dependent.

To validate this expectation, we check the conditions for sequential independence of user actions from Def. 6.3.10. The first condition (S1) (leading to a commuting square (2), cf. Fig. 6.9) holds if the reconfiguration (ϱ_3, o_3) with $\varrho_3 = \mathit{CreateDirectChannel}$ is sequentially independent from the firing step, which corresponds to the transformation (ϱ, o_t) where $\varrho = \varrho(\mathit{sendIMAB}, \hat{S}_4, \widehat{asg}_4)$ is the transition firing rule for the firing step $(\mathit{sendIMAB}, \widehat{asg}_4, \hat{S}_4)$ and o_t is the obvious inclusion match. For the sequential independence (see Def. 6.3.7), there have to exist morphisms i_3 and i as in Fig. 7.62, such that $g_3 \circ i = o_t$ and that i_3 and o'_3 are equal in their components for places, transitions, and algebras. But, there does not even exist any morphism $L \to D_3$, because in D_3 (which is up to the marking the same as ANI_3 in Fig. 7.50) there is no transition that could be matched by *sendIMAB* in L. Therefore, the condition (S1) does not hold and ua_3 and ua_4 are sequentially trigger-dependent, as expected.[10]

$$\begin{tikzcd}
ua_4 : & & ANI_4 \arrow[dl, tail, "{sendIMAB,\widehat{asg}_4,\hat{S}_4}"'] \arrow[dr, tail, "{requestContactA,\widehat{asg}_5,\hat{S}'_5}"] & & : ua'_5 \\
& ANI_5 \arrow[dr, tail, "{requestContactA,\widehat{asg}_5,\hat{S}_5}"'] & (\mathbf{1}) & ANI'_4 \arrow[dl, tail, "{sendIMAB,\widehat{asg}_4,\overline{\hat{S}}'_4}"'] \arrow[dr, Rightarrow, "{CreateContactExchange,o'_5}"] & \\
& ua_5 : & ANI'_5 \arrow[dr, Rightarrow, "{CreateContactExchange,o_5}"'] & (\mathbf{3}) & ANI''_4 \arrow[dl, tail, "{sendIMAB,\widehat{asg}_4,\hat{S}'_4} \ : ua'_4"] \\
& & & ANI_6 &
\end{tikzcd}$$

Figure 7.63.: Sequentially independent user actions ua_4 and ua_5

[10] For the degrees of independence and dependence of user actions without reconfigurations such as ua_4, see the remark on user actions without reconfigurations on page 160 in Sect. 6.3.3.

ua_4 **and** ua_5 (see page 235) as shown in the left of Fig. 7.63 should be sequentially independent, because the sending of a message (ua_4) in Skype is not affected by posing a request for contact exchange (ua_5). First, we verify that the sequential independence condition (S2) holds, i. e., the two firing steps are sequentially permutable. The first permutability condition (Sa) from Def. 6.3.2 requires that the values of the tokens consumed by the firing of $(\mathit{requestContactA}, \widehat{asg}_5)$ can be found in the marking of ANI_4 (see Fig. 7.51), which is easy to see because $pre_A(\mathit{requestContactA}, \widehat{asg}_5) = (\mathit{Alice}, \texttt{UserA})$ is just Alice's identifier token, which is present in all platform nets. For the second condition (Sb), the monoid sum of the consumed token values is

$$\begin{aligned} pre_A(\mathit{sendIMAB}, \widehat{asg}_4) =& (\mathit{Alice}, \texttt{UserA}) \oplus ((\mathit{Anyone}, \mathit{OnlyContacts}), \texttt{PrivacyB}) \\ & \oplus (\mathit{Alice}, \texttt{NotBlockedByAB}) \oplus (\mathit{Bob}, \texttt{NotBlockedByAB}) \\ & \oplus (\mathit{Empty}, \texttt{HistoryAB}) \end{aligned}$$

Clearly, $pre_A(\mathit{sendIMAB}, \widehat{asg}_4)$ is a subsum of the marking sum of ANI_4, because the firing step of $(\mathit{sendIMAB}, \widehat{asg}_4)$ is enabled in ANI_4. The right side of the inequation of (Sb) is the marking sum of ANI'_4, which is the marking sum of ANI_4 extended by an individual token with the value $\mathit{contactReq}(\mathit{Alice}, \mathit{Bob})$ on place `CRA`. Hence, the two firing steps are sequentially permutable so that we get the commutative square (1) in Fig. 7.63 by Theorem 6.3.3. Moreover, (S3) holds because the firing step $ANI'_4 \xrightarrow{\mathit{sendIMAB}, \widehat{asg}_4, \overline{\hat{S}}'_4} ANI'_5$ is parallel independent of the functional handling $(\mathit{CreateContactExchange}, o'_5)$. It turns out that the match o_5 for rule $\mathit{CreateContactExchange}$ in ANI'_5 results from constructing the complement transformation for $(\mathit{CreateContactExchange}, o'_5)$ along the parallel independency square (3). Therefore, ua_4 and ua_5 are sequentially independent as expected. The user actions ua'_5 and ua'_4 in the right of Fig. 7.63 represent these actions in the opposite order with the same result ANI_6.

$$\begin{tikzcd}
 & & ANI_5 \arrow[dl, tail, "\mathit{requestContactA}, \widehat{asg}_5, \hat{S}_5"'] \\
ua_5 : & ANI'_5 \arrow[dl, Rightarrow, "\mathit{CreateContactExchange}, o_5"'] \arrow[dr, tail, dashed] & \\
ANI_6 \arrow[dr, tail, "\mathit{acceptBA}, \widehat{asg}_6, \hat{S}_6"'] & \mathbf{(2)} & ANI''_5 \arrow[dl, Rightarrow, dashed] \\
ua_6 : & ANI'_6 \arrow[dr, Rightarrow, "\mathit{ExtendToPermittedChannel}, o_6"'] & \\
 & & ANI_7
\end{tikzcd}$$

Figure 7.64.: Sequentially trigger-dependent user actions ua_5 and ua_6

ua_5 and ua_6 (see page 236) shown in Fig. 7.64 are expected to be trigger-dependent because Bob should not be able to accept a contact exchange request from Alice (ua_6) before she even has posed it (ua_5). Indeed, similarly to ua_3 and ua_4, the sequential independence condition (S1) is not satisfied, because the transition *acceptBA* to be fired in ua_6 is created by the reconfiguration of ua_5. Hence, these user actions are sequentially trigger-dependent.

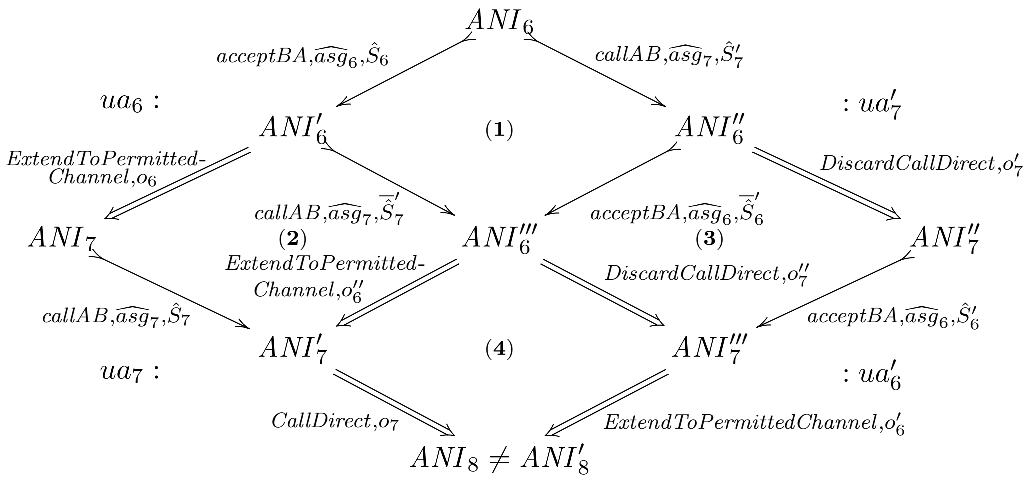

ANI_6
$acceptBA,\widehat{asg}_6,\hat{S}_6$
$callAB,\widehat{asg}_7,\hat{S}'_7$
ua_6 :
ANI'_6
(1)
ANI''_6
: ua'_7
ExtendToPermitted-Channel,o_6
$DiscardCallDirect,o'_7$
$callAB,\widehat{asg}_7,\overline{\hat{S}}'_7$
$acceptBA,\widehat{asg}_6,\overline{\hat{S}}'_6$
ANI_7
(2)
ANI'''_6
(3)
ANI''_7
ExtendToPermitted-Channel,o''_6
$DiscardCallDirect,o''_7$
$callAB,\widehat{asg}_7,\hat{S}_7$
$acceptBA,\widehat{asg}_6,\hat{S}'_6$
ANI'_7
(4)
ANI'''_7
ua_7 :
: ua'_6
$CallDirect,o_7$
$ExtendToPermittedChannel,o'_6$
$ANI_8 \neq ANI'_8$

Figure 7.65.: Sequentially dependent and trigger-independent user actions ua_6 and ua_7

Because up to user action ua_8 we have not given the details of the user actions between Carol and Bob that follow ua_6 (see page 237) in the example activities, we consider the user action ua_8, where Alice calls Bob, to be performed directly after ua_6 as a user action ua_7 in an alternative example scenario. The matches and assignments of ua_7 are identical to those of ua_8.

ua_6 and alternative ua_7 (see ua_8 on page 237) are the user actions where Alice calls Bob after Bob accepts Alice's request for contact exchange (see the left of Fig. 7.65). In Skype, we expect that a call (or at least a try to call) can be performed at any time independently of whether a contact exchange request has been accepted or denied before by the called user. Because of this, we expect that ua_6 and ua_7 are at least trigger-independent to allow the call of ua_7 to be possible before Alice accepts Bob's request in ua_6. Nevertheless, accepting the request in ua_6 affects the effect of Bob's privacy settings for Alice. In our Skype platform model, clients are always created with the privacy setting that incoming calls are allowed for known contacts, only (cf. Fig. 7.49), and Bob does not change these settings in the example scenarios. Therefore before the user action ua_7, we expect that a try such as in ua_6 by Alice to call Bob fails due to the general privacy setting that is not superseded by the exchanged contact. In other words, we expect a different outcome of the user actions if performed in the opposite order, i. e., that they be sequentially dependent (or equivalently, that they not be sequentially weakly independent, cf. Table 6.1). This is an example of two user actions for which we expect a degree of independence together with a degree of dependence.

The sequential independence conditions (S1) and (S2) are easy to verify and lead to the commutative squares (2) and (1) in Fig. 7.65. The transformation $(\mathit{DiscardCallDirect}, o_7')$ is the functional handling of $\mathit{handler}(\mathit{callReq})$ on ANI_6'', which just deletes the $\mathit{callReq}$ request token that is produced by the firing of callAB. The match o_7' satisfies the application subcondition $\exists(a_2, \neg\exists a_{21})$ of the higher-prioritized rule $\mathit{DiscardCallDirect}$ (see Fig. 7.20), because Bob does not have Alice's contact on his contact list. This transformation is parallel independent of the firing of $\mathit{acceptBA}$, leading to the commutative square (3) and satisfying the independence condition (S3). With this, we have that ua_6 and ua_7 are sequentially trigger-independent.

The functional handlings $(\mathit{ExtendToPermittedChannel}, o_6')$ on ANI_7''' (where o_6' essentially matches as o_6'' does on ANI_6''') and $(\mathit{CallDirect}, o_7)$ on ANI_7' lead to different results ANI_8 — where the call from Alice to Bob has started — and ANI_8' — where the call from Alice to Bob has been rejected — due to the different handlings with the application of rule $\mathit{DiscardCallDirect}$ rather than $\mathit{CallDirect}$ if the call is performed before the contact exchange. Hence, the weak independence condition (wS4) is violated and ua_6 and ua_7 are sequentially dependent.[11]

We considered ua_7 as an user action in an alternative scenario after ua_6. Therefore we continue with ua_8 and ua_9 from the actual scenarios, because ua_8 is not performed in the result of ua_7 so that we cannot analyze them for sequential independency.

ua_8 and ua_9 (see page 239) are the user actions where Alice calls Bob and Bob then joins ("picks up") the call in the following sequence of firing steps and transformations:

$$\mathit{ANI}_8 \rightarrowtail^{\mathit{callAB}, \widehat{\mathit{asg}}_8, \hat{S}_8} \mathit{ANI}_8' \xRightarrow{\mathit{CallDirect}, o_8} \mathit{ANI}_9 \rightarrowtail^{\mathit{joinBA}, \widehat{\mathit{asg}}_9, \hat{S}_9} \mathit{ANI}_{10}$$

We expect ua_8 and ua_9 to be sequentially trigger-dependent, because the possibility for Bob to join the call is given by and depends on the prior calling of Alice. Indeed, the sequential independence condition (S1) is violated, analogously to the example for ua_3 and ua_4 in Fig. 7.61.

ua_9 and ua_{10} (see page 239) are shown in the left of Fig. 7.66, where Bob first joins the call from Alice and Alice then says something, which is transmitted to Bob. Of course, Alice should always be able to say something; but if there is no ongoing call, we expect that her speech is not transmitted and simply vanishes unheard. Therefore, we expect on the one hand that these user actions be sequentially trigger-independent so that Alice can talk before or after Bob joining the call, but on the other hand that they be sequentially dependent, because the results should be different (similarly to ua_6 and ua_7).

[11] Note that we do not regard the fact that the transformations from the central ANI_6''' are parallel independent, because the complement transformation $\mathit{ANI}_7' \xRightarrow{\mathit{DiscardCallDirect}, o_7'''} \mathit{ANI}_8$ constructed for (4) is not a functional handling and therefore irrelevant for the independence considerations.

$$\begin{tikzcd}
ua_9 : & & ANI_9 \arrow[dl, tail, "{joinBA,\widehat{asg}_9,\hat{S}_9}"'] \arrow[dr, tail, "{sayA,\widehat{asg}_{10},\hat{S}'_{10}}"] & & \\
& ANI_{10} \arrow[dr, tail, "{sayA,\widehat{asg}_{10},\hat{S}_{10}}"'] & (\mathbf{1}) & ANI'_9 \arrow[dl, tail, "{joinBA,\widehat{asg}_9,\overline{\hat{S}}'_9}"'] \arrow[dr, Rightarrow, "{TransmitConference_K,o'_{10}}"] & : ua'_{10} \\
& ua_{10} : & ANI'_{10} \arrow[dr, Rightarrow, "{TransmitCall,o_{10}}"'] & (\mathbf{3}) & ANI''_9 \arrow[dl, tail, "{joinBA,\widehat{asg}_9,\hat{S}'_9}"] \\
& & & ANI_{11} \neq ANI'_{11} & : ua'_9
\end{tikzcd}$$

Figure 7.66.: Sequentially dependent and trigger-independent user actions ua_9 and ua_{10}

The triggering firing steps of these user actions are sequentially permutable, which leads to the commutative square (1) in Fig. 7.66 and satisfies the sequential independence condition (S2). Now, the functional handling of the transmit request produced by the firing of *sayA* in ANI'_9 is not an application of the rule *TransmitCall* in Fig. 7.46, because it requires a token T on the place `HasJoinedBA` in Fig. 7.54 (which is the same net as ANI'_9 up to the missing request token from the firing of *sayA*). Instead, the lower-prioritized interaction scheme *TransmitConference* in Fig. 7.47 from the handler *handler*(*transReq*) is applied to ANI'_9. We do not find any match for the multi rule of this interaction scheme, so the amalgamated rule that results from maximal matching is just the kernel rule $TransmitConference_K$ of this interaction scheme.[12] The kernel rule matches Alice's request token with o'_{10} and deletes it. The two paths leads to different results ANI_{11}, where Bob has received Alices speech, and ANI'_{11}, where the spoken message from Alice has vanished. Hence, ua_9 and ua_{10} are sequentially dependent, but nevertheless sequentially trigger-independent, because the firing of *joinBA* in ANI'_{10} and the transformation $(TransmitConference_K, o'_{10})$ are parallel independent.

ua_{10} and ua_{11} (see page 239) are the user actions where Alice first says something to Bob in the current call and then ends this call by hanging up. The dependencies are similar as for ua_9 and ua_{10}, i. e., they should be sequentially trigger-independent but also sequentially dependent, because after hanging up, nothing of what Alice says should be transmitted to Bob any longer. Checking the sequential independence conditions for ua_{10} and ua_{11} yields a diagram similar to Fig. 7.65 showing that the conditions (S1) to (S3) are satisfied, but the functional request handlings lead to different results due to the different handlings of the transmit request: Similar to ua_9 and ua_{10} in Fig. 7.66, the application of rule *TransmitCall* in Fig. 7.46 transmits data to Bob if Alice talks (ua_{10}) before hanging up (ua_{11}), and the application of the interaction scheme *TransmitConference* discards this data if Alice talks after hanging up.

[12] Note that, besides for transmitting data in conferences, we designed the interaction scheme *TransmitConference* explicitly to catch this case of discarding an invalid transmit request, as described on page 225.

$$\begin{tikzcd}
 & & ANI_{11} \arrow[dl, tail, "\mathit{hangupAB}{,}\widehat{asg}_{11}{,}\hat{S}_{11}"'] \arrow[dr, tail, "\mathit{requestGroupB}{,}\widehat{asg}_{12}{,}\hat{S}'_{12}"] & & \\
ua_{11}: & ANI'_{11} \arrow[dl, Rightarrow, "\mathit{HangupDirectCall}{,}o_{11}"'] \arrow[dr, tail, "\mathit{requestGroupB}{,}\widehat{asg}_{12}{,}\overline{\hat{S}}_{12}"] & (\mathbf{1}) & ANI''_{11} \arrow[dl, tail, "\mathit{hangupAB}{,}\widehat{asg}_{11}{,}\overline{\hat{S}}'_{11}"'] \arrow[dr, Rightarrow, "\mathit{CreateGroup}{,}o'_{12}"] & :ua'_{12} \\
ANI_{12} \arrow[dr, tail, "\mathit{requestGroupB}{,}\widehat{asg}_{12}{,}\hat{S}_{12}"'] & (\mathbf{2}) & ANI'''_{11} \arrow[dl, Rightarrow, "\mathit{HangupDirectCall}{,}o''_{11}"'] \arrow[dr, Rightarrow, "\mathit{CreateGroup}{,}o''_{12}"] & (\mathbf{3}) & ANI''_{12} \arrow[dl, tail, "\mathit{hangupAB}{,}\widehat{asg}_{11}{,}\hat{S}'_{11}"] \\
ua_{12}: & ANI'_{12} \arrow[dr, Rightarrow, "\mathit{CreateGroup}{,}o_{12}"'] & (\mathbf{4}) & ANI'''_{12} \arrow[dl, Rightarrow, "\mathit{HangupDirectCall}{,}o'_{11}"] & :ua'_{11} \\
 & & ANI_{13} & &
\end{tikzcd}$$

Figure 7.67.: Sequentially independent user actions ua_{11} and ua_{12}

ua_{11} **and** ua_{12} (see page 241) are the user actions where first Alice hangs up the call to Bob and then Bob creates a group with Alice and Carol, shown in the left of Fig. 7.67. Intuitively, the creation of a group in Skype should not be affected by whether there is an ongoing call or vice versa. The call and the group are considered as channels that are not related.

Checking the conditions for sequential independence, we find that (S1) to (S3) are satisfied. Moreover, the complement transformations in square (4), $ANI'_{12} \xRightarrow{\mathit{CreateGroup},o_{12}} ANI_{13}$ (which actually is the same as the reconfiguration of ua_{12}) and $ANI'''_{12} \xRightarrow{\mathit{HangupDirectCall},o'_{11}} ANI_{13}$, are functional so that (S4) is satisfied and therefore ua_{11} and ua_{12} are sequentially independent.

ua_{12} **and** ua_{13} (see page 242) are the user actions where Bob first creates a new group (with Alice and Carol) and Alice then sends an instant message in this group to Bob and Carol, which is realized by the following sequence (where we omit the request handling of ua_{13}):

$$ANI_{12} \xrightarrowtail{\mathit{requestGroupB},\widehat{asg}_{12},\hat{S}_{12}} ANI'_{12} \xRightarrow{\mathit{CreateGroup},o_{12}} ANI_{13} \xrightarrowtail{\mathit{sendIMA},\widehat{asg}_{13},\hat{S}_{13}} ANI'_{13} \ldots$$

We expect that these actions be sequentially trigger-dependent because Alice should not be able to send a message in a group before Bob even has created it. It turns out that the sequential independence condition (S1) is violated, analogously to ua_3 and ua_4 in Fig. 7.61.

With this, we have validated that all intuitive expectations (regarding the modeling requirements for Skype) for the sequences of user actions in the example scenarios are met by the Communication Platform model. Table 7.4 summarizes the actual degrees of independence for these user actions.

ua_3		ua_8	
↓	trigger-dependent	↓	trigger-dependent
ua_4		ua_9	
↓	independent	↓	trigger-independent, dependent
ua_5		ua_{10}	
↓	trigger-dependent	↓	trigger-independent, dependent
ua_6		ua_{11}	
↓	trigger-independent, dependent	↓	independent
ua_7		ua_{12}	
		↓	trigger-dependent
		ua_{13}	

Table 7.4.: Degrees of sequential (in)dependence in example scenarios

7.4.3.2. Parallel Self-Dependence of User Actions

Similar to analyzing consecutive user actions for degrees of sequential independence and dependence, the concept of parallel user action independence is useful to validate the expected effects of single user actions. By checking whether and to which extent a user action is parallel independent to itself, we can identify flaws in the model concerning the creation and modification of channels and other net structures, for which we expect either the same or different effects if a user performs a particular action more than once. In the following, we consider the user actions ua_3 to ua_{13} from the example scenarios in Sect. 7.3 and compare the actual degrees of parallel self-independence to the intuitively expected ones. Note that we expect the token selection of the concurring firing steps to comply to the conditions stated in the remarks on page 151 and page 162, i. e., that the complement firing steps lead to the same result and that the sets of created tokens are disjoint.

ua_3 (see page 232 and top of Fig. 7.68) is the action of Alice creating a direct channel to Bob. In the modeling requirements for direct channels in Sect. 7.1.2, we intend a direct channel to represent the context for the communication between two particular users so that a direct channel should be unique. For this, we expect that Alice's second request for a direct channel is discarded because there already exists one. This means that the handling of the second request should be different (discarding) than the first one (creating) and ua_3 be parallel weakly dependent, i. e., not independent. Because we have modeled the request for a direct channel as a transition in the client core, we expect that Alice can pose the request for a direct channel to Bob more than once, but this expectation comes from our technical insight of the formal platform model rather than from our intuitive understanding of the Skype platform.

To ensure that we get a commuting square (1) from Theorem 6.3.3 with strong parallel permutability, we consider the concurring parallel firing step for a different token selection $\hat{S}_3^*$ instead of the same selection $\hat{S}_3$ (cf. the remark on page 162).

Checking the conditions for parallel independence shows that (P1) to (P3) are sat-

$$\begin{tikzcd}
 & & \mathit{ANI}_3 \arrow[dl, tail, "\mathit{requestDirectA},\widehat{asg}_3,\hat{S}_3"'] \arrow[dr, tail, "\mathit{requestDirectA},\widehat{asg}_3,\hat{S}^*_3"] & & \\
 & \mathit{ANI}'_3 \arrow[dl, Rightarrow, "\mathit{CreateDirectChannel},o_3"'] \arrow[dr, tail, "\mathit{requestDirectA},\widehat{asg}_3,\overline{\hat{S}}^*_3"] & \textbf{(1)} & \mathit{ANI}^{*\prime}_3 \arrow[dl, tail, "\mathit{requestDirectA},\widehat{asg}_3,\overline{\hat{S}}_3"'] \arrow[dr, Rightarrow, "\mathit{CreateDirectChannel},o^*_3"] & \\
\mathit{ANI}_4 \arrow[dr, tail, "\mathit{requestDirectA},\widehat{asg}_3,\hat{S}^{*\prime}_3"'] & \textbf{(2)} & \mathit{ANI}''_3 \arrow[dl, Rightarrow, "\mathit{CreateDirectChannel},o'_3"'] \arrow[dr, Rightarrow, "\mathit{CreateDirectChannel},o^{*\prime}_3"] & \textbf{(3)} & \mathit{ANI}^*_4 \arrow[dl, tail, "\mathit{requestDirectA},\widehat{asg}_3,\hat{S}'_3"] \\
 & \mathit{ANI}'_4 \arrow[dr, Rightarrow, "\mathit{DiscardDirectReqExisting},o^{*\prime\prime}_3"'] & \textbf{(4)} & \mathit{ANI}^{*\prime}_4 \arrow[dl, Rightarrow, "\mathit{DiscardDirectReqExisting},o''_3"] & \\
 & & \mathit{ANI}_5 & &
\end{tikzcd}$$

Figure 7.68.: Parallel weakly independent and weakly dependent user action ua_3

isfied. For the commutative square (4), only the weak condition (wP4) holds, because the higher-prioritized rule *DiscardDirectReqExisting* of the channel requests handler can match on ANI'_4 as the functional handling. As expected, ua_3 turns out to be parallel weakly independent and weakly dependent to itself.

ua_4 (see page 235) is Alice sending an instant message to Bob. Intuitively, sending a message should not depend on whether the same message already has been send. But, we find that the firing step $\mathit{ANI}_4 \xrightarrow{\mathit{sendIMAB},\widehat{asg}_4,\hat{S}_4} \mathit{ANI}_5$ is not parallel permutable because it removes the *Empty* log on the history place `HistoryAB` for this direct channel (see Fig. 7.51) and creates a new log token with the sent message (see Fig. 7.52). In this case, the parallel dependency of ua_4 does not comply to our expectation, but for general messages, which may be different, we would expect this dependency, because the history depends on the order in which the messages are sent. With this is mind, we can conclude that the dependency is what we expect for the general case, even if for this special case of equal messages the user actions formally are dependent, although they do lead to the same result. But the conditions for independence are only sufficient but not necessary for confluence of user actions.

ua_5 (see page 235) is the creation of a contact exchange structure, in which Bob can decide whether he wants to exchange contacts with Alice. Usually, in Communication Platforms, one can only request a contact exchange form another user only once until the other user accepts or denies. This is a similar situation as for direct channels. The independence analysis shows that the conditions (P1) to (P3) and (wP4) are satisfied as for the direct channel in ua_3 so that we get weak independence and weak dependence for ua_5 w. r. t. itself. This means that if a contact exchange structure from one particular user to another has been created, a second request for such a exchange is possible but treated differently, as we expected.

ua_6 (see page 236) as the action that Bob accepts Alice's request for contact exchange should not allow Bob to perform this accepting action again, which would be redundant. Indeed, the firing step $\mathit{ANI}_6 \xrightarrow{\mathit{acceptBA},\widehat{asg}_6,\hat{S}_6} \mathit{ANI}'_6$ is not even parallel permutable with itself, because the marking of ANI'_6 (cf. Fig. 7.52) does not contain the necessary control token for firing *acceptBA* (which is deleted by the request handling of ua_6, anyway). Hence, ua_6 is parallel trigger-dependent to itself as expected.

$$\begin{tikzcd}
 & & \mathit{ANI}_8 \arrow[dl, tail, "\mathit{callAB},\widehat{asg}_8,\hat{S}_8"'] \arrow[dr, tail, "\mathit{callAB},\widehat{asg}_8,\hat{S}^*_8"] & & \\
 & \mathit{ANI}'_8 \arrow[dl, Rightarrow, "\mathit{CallDirect},o_8"'] \arrow[dr, "\mathit{callAB},\widehat{asg}_8,\overline{\hat{S}}^*_8"] & \textbf{(1)} & \mathit{ANI}^{*\prime}_8 \arrow[dl, "\mathit{callAB},\widehat{asg}_8,\overline{\hat{S}}_8"'] \arrow[dr, Rightarrow, "\mathit{CallDirect},o^*_8"] & \\
\mathit{ANI}_9 \arrow[dr, dashed] & \textbf{(2)} & \mathit{ANI}''_8 \arrow[dl, Rightarrow, dashed] & & \mathit{ANI}^*_9 \\
 & \mathit{ANI}'_9 & & &
\end{tikzcd}$$

Figure 7.69.: Parallel trigger-dependent user action ua_8

ua_8 (see page 237 and top of Fig. 7.69) is the action where Alice initiates a call to Bob. In Skype, we expect that Alice is busy in this call, even if for now she just waits that Bob answers the call. Users who are busy in a call without putting this call on hold should not be able to participate in another call. Therefore, ua_8 should be at least sequentially dependent so that the second request for a call to Bob is denied with a different handling, but still leading to the same platform net where Alice in is a call with Bob.

The triggering firing step indeed is (strongly) parallel permutable to itself so that we get (1) for the independence condition (P1). But, the handling via $\mathit{CallDirect}, o_8$ is not parallel independent to the complement firing, because this transformation deletes the transition *CallAB* so that (P2) is violated. ua_8 therefore is trigger-dependent to itself, i. e., the action of calling a user prevent that the same (and other) users can be called directly afterward. This even exceeds our expectation of sequential dependence but is consistent to it, because we (reasonably) do not expect any grade of independence for this user action.

ua_9 (see page 239) is the action where Bob joins the call from Alice, i. e., he "picks up the phone". Obviously, this action should not be possible twice. The firing step $\mathit{ANI}_9 \xrightarrow{\mathit{joinBA},\widehat{asg}_9,\hat{S}_9} \mathit{ANI}_{10}$ is not (strongly) parallel permutable with itself, because the token on place `HasJoinedB` with the value F needed for firing transition *joinBA* is no longer available in the marking of $\mathit{ANI}_{\mathit{10}}$ (cf. Fig. 7.54). With this, ua_9 is parallel trigger-dependent to itself.

$$\begin{tikzcd}
 & & \mathit{ANI}_{10} \arrow[dl, tail, "\mathit{sayA},\widehat{\mathit{asg}}_{10},\hat{S}_{10}"'] \arrow[dr, tail, "\mathit{sayA},\widehat{\mathit{asg}}_{10},\hat{S}^*_{10}"] & & \\
 & \mathit{ANI}'_{10} \arrow[dl, Rightarrow, "\mathit{TransmitCall},o_{10}"'] \arrow[dr, tail, "\mathit{sayA},\widehat{\mathit{asg}}_{10},\overline{\hat{S}}^*_{10}"'] & (\mathbf{1}) & {\mathit{ANI}^*_{10}}' \arrow[dl, tail, "\mathit{sayA},\widehat{\mathit{asg}}_{10},\overline{\hat{S}}_{10}"] \arrow[dr, Rightarrow, "\mathit{TransmitCall},o^*_{10}"] & \\
\mathit{ANI}_{11} \arrow[dr, tail, "\mathit{sayA},\widehat{\mathit{asg}}_{10},\hat{S}^{*\prime}_{10}"'] & (\mathbf{2}) & \mathit{ANI}''_{10} \arrow[dl, Rightarrow, "\mathit{TransmitCall},o'_{10}"] \arrow[dr, Rightarrow, "\mathit{TransmitCall},o^{*\prime}_{10}"'] & (\mathbf{3}) & \mathit{ANI}^*_{11} \arrow[dl, tail, "\mathit{sayA},\widehat{\mathit{asg}}_{10},\hat{S}'_{10}"] \\
 & \mathit{ANI}^*_{11} \arrow[dr, Rightarrow, "\mathit{TransmitCall},o^{*\prime\prime}_{10}"'] & (\mathbf{4}) & \mathit{ANI}^{*\prime}_{11} \arrow[dl, Rightarrow, "\mathit{TransmitCall},o''_{10}"] & \\
 & & \mathit{ANI}_{12} & &
\end{tikzcd}$$

Figure 7.70.: Parallel independent user action ua_{10}

ua_{10} (see page 239 and top of Fig. 7.70) represents Alice saying something that is transmitted to Bob in their ongoing call. We could demand weak parallel independence for this user action to itself, i. e., that both of Alice's spoken messages are transmitted somehow in after performing the user action twice. However, talking to each other is an elementary action in calls and conferences so that we expect that the transmission of (spoken) data is independent to all other actions, except for putting a call on hold or hanging up. Regarding that the transmission should not be affected by other actions at all, it is also reasonable to demand independence without weak dependence.

It is easy to verify that all parallel independence conditions (P1) to (P4) hold for this user action, because the triggering firing steps simply produce a request token without any preconditions and the handling transformation moves this token to the receiving client. For all the nets below square (1) in Fig. 7.70, the functional handlings are the same as for ua_{10} concurring itself because Alice and Bob are participating in the call between them, which is identified by the application conditions of the rule *TransmitCall* (see Fig. 7.46).

ua_{11} (see page 241) is the action where Bob hangs up and terminates the call from Alice. As for regular phone calls, a call in Skype can only be hung up once so that ua_{11} should be trigger-dependent to itself.

The triggering firing step $\mathit{ANI}_{\mathit{11}} \xrightarrow{\mathit{hangupAB},\widehat{\mathit{asg}}_{11},\hat{S}_{11}} \mathit{ANI}'_{\mathit{11}}$ is (strongly) parallel permutable to itself, because the transition *hangupAB* simply creates a request token for terminating the call without removing any other token (see Fig. 7.54). But as for ua_8, the handling transformation $\mathit{ANI}'_{\mathit{11}} \xRightarrow{\mathit{HangupDirectCall},o_{11}} \mathit{ANI}_{\mathit{12}}$ deletes the transition *hangupAB* (see Fig. 7.55) so that the parallel independence condition (P2) is violated and ua_{11} is trigger-dependent to itself.

ua_{12} (see page 242) is Bob's action where he creates a new group with Alice and Carol. In Skype, one can freely create groups with a selection of his contacts without any constraints (see page 48) so that this user action should be independent to itself (and to other user actions).

All conditions for parallel independence are satisfied by ua_{12} because the triggering firing step $ANI_{12} \xrightarrow{requestGroupB,\widehat{asg}_{12},\hat{S}_{12}} ANI'_{12}$ (see Fig. 7.55) just creates a request token without removing any other token and because the handling transformation $ANI'_{12} \xRightarrow{CreateGroup,o_{12}} ANI_{13}$ just deletes this request token and creates the group structure without any further application conditions (see Fig. 7.27).

ua_{13} (see page 243) is the action where Alice sends an instant message in Bob's group so that Bob and Carol receive it. Sending messages in groups is as elementary as the transmission of speech in calls and conference (see $ua_1$0) and should be independent of any other action, except blocking the sender. But, the sent and received messages are stored in the history logs of each user in this group so that we get an unexpected dependency for sending the same group message twice: $ANI_{13} \xrightarrow{sendIMA,\widehat{asg}_{13}} ANI'_{13}$ is not parallel permutable, because it deletes the *Empty* history token on place `HistoryB` (see Fig. 7.56) and creates a history with the sent message appended. As for ua_4, the general case of sending different messages is expected to be dependent and we can see that sending the same message twice indeed leads to the same result, even if formally the action ua_{13} is self-dependent and violates (P1).

ua_3:	weakly independent, weakly dependent	ua_9:	trigger-dependent
ua_4:	dependent	ua_{10}:	independent
ua_5:	weakly independent, weakly dependent	ua_{11}:	trigger-dependent
ua_6:	trigger-dependent	ua_{12}:	independent
ua_8:	trigger-dependent	ua_{13}:	dependent

Table 7.5.: Degrees of parallel self-(in)dependence in example scenarios

With this, we have validated that all intuitive expectations (regarding the modeling requirements for Skype) for the parallel self-dependencies of user actions in the example scenarios are met by the Communication Platform model. Table 7.5 summarizes the actual degrees of independence for these user actions.

8. Tool Support

> Non mi interessa essere capito, mi interessa essere!
> Capito? Essere! Capito?
>
> *(Caparezza)*

This chapter describes the development of a prototypical tool environment called the CPEditor, which supports editing, simulation, and validation of Communication Platform models. The CPEditor is currently under development; we give an outlook on the complete set of required features that it will support, together with a status of their implementations.

8.1. Requirements for Tool Support for Modeling of Communication Platforms

We gather the main requirements for the CPEditor by enumerating the most important features that we want to be supported by the CPEditor.

Editing According to Def. 5.3.3, the CPEditor has to allow the designer to create the following essential parts of a Communication Platform model

$$CPM = (\Sigma, A, ANI, R, IS, S_{Req}, handler)$$

- For the AHLI net ANI that represents the current configuration of the platform, the CPEditor accepts a custom signature for the arc inscriptions and a custom implementation of an Σ-algebra A for the algebraic token values of the marking. Σ should possibly extend the common platform signature $Sigma_{CP}$ for Communication Platform Models (see Table 5.1). The set of sorts S_{Req} can be marked as request sorts.

- A suitable component of the CPEditor allows the user to edit the current platform AHLI net ANI w. r. t. the given signature Σ and algebra A.

- Similar components allow the designer to create and edit the sets R and IS of correct AHLI transformation rules and interaction schemes with nested application conditions. The token values of the markings of the rules' nets are Σ-terms over a set of token variables.

- For the constructor operations of the request sorts S_{Req}, the designer can define handler expressions as priority list sequences of the existing transformation rules and interaction schemes, where all handler expressions together implicitly define the *handler* function.

Simulation User actions and external actions in the Communication Platform can be simulated as they would be simulated by the control higher-order net in Sect. 5.4. The CPEditor helps by highlighting the activated transitions in the platform AHLI net. Because platform nets can become quite complex and intertwined, a special simulation view listing only the currently possible user actions is the best choice of a control for simulation. To be able to provide the names of the users who can perform the user actions, the editor should know about a particular sort for identities and a reserved variable for transition ownership (cf. Sect. 5.1.1.1). Possible user actions can be selected and performed, where optionally defined handlers are applied immediately. For this, the CPEditor searches valid match sequences and applies them directly after the triggering transition firing. External rules are applied manually by selecting the corresponding rule. The performed external and user actions are kept in a history and can be undone.

Validation To support the validation of Communication Platform models, the designer can define platform constraints in the CPEditor, which are validated automatically or manually after every external and user action. Moreover, the CPEditor gives a warning if possible user actions are not functional or not successful (see Defs. 5.3.2 and 6.2.1). The independence of user actions can be checked in two ways. For two possible user actions that are selected in the simulation view, the CPEditor checks and reports the degree of their parallel (in)dependency (cf. Sect. 6.3.3). For checking sequential independence, the designer can select a possible user action of which the degree of sequential independence to the immediately previously performed one is checked and reported.

8.2. Available Software Frameworks and Tools

We present the Eclipse platform and some plugins that provide essential features and foundations or that implement similar functionalities we can possibly reuse for our Communication Platform tool.

The Eclipse Platform

Eclipse started its existence as an integrated development environment for Java (written itself in Java) and was released in 2001 by IBM. In 2004, the Eclipse Foundation has been founded, which since then has been maintaining Eclipse' main and associated projects. Besides the fact that it is open and free software, the main factor for the wide distribution of Eclipse is the great flexibility that users and developers gain from its extensible plugin architecture. In addition to the various high-quality plugins and projects hosted and controlled by the main Eclipse website[1], the Eclipse Marketplace[2] lists over thousand solutions and plugins under free or commercial licenses for Eclipse. For example,

[1] http://eclipse.org/projects/listofprojects.php

[2] http://marketplace.eclipse.org

for almost any major programming language there are (third-party) plugins extending the Eclipse workbench by editors and tools to a development environment for the corresponding language, e. g., Ada, C, C++, COBOL, Perl, PHP, Python, Ruby, XML, LaTeX etc. Other examples are plugins incorporating valuable development tools for managing projects, tasks, and source code (by versioning tools such as CVS or SVN). Eclipse features also a plugin development environment as a standard plugin so that plugins for Eclipse can be written, managed, and packaged directly in Eclipse.

Frameworks for Visual Eclipse Editors: EMF, GEF, MuvitorKit

We use the following three open source frameworks, which are Eclipse plugins, as the basis for editor plugins in Eclipse that support visual models such as graphs and Petri nets.

The Eclipse Modeling Framework (EMF) allows developers to define structured data models over the EMF meta model called ECore [SBPM09].[3] The ECore meta model is almost equivalent to the standard Essential MOF (EMOF) [OMG06], which is a part of the foundations of the UML and its class diagrams. ECore and EMOF provide a minimal set of elements such as classes, attributes, and references, required for modeling object-oriented systems.

The most important aspect of EMF is that it can automatically generate Java code out of the models. This code can be used in any application to create and manipulate model instances, to listen for changes in the model, to save a model instance to a file, and to load it afterward. These persistency operations (with an XMI file format) and more generated features such as a tree-based editor for instances of the model can be used directly as they are generated. All this spares the developers cumbersome work and supports model-driven development, especially if the model is object to changes during the development process, where EMF then can simply regenerate consistent code for it.

The Graphical Editing Framework (GEF) is used to create graphical editors for visual and diagram languages as plugins for the Eclipse workbench.[4] GEF follows the principles of the Model-View-Controller pattern, which is an architectural design pattern dividing an application into separate parts. The goal is to keep the data model apart from its representation in the graphical user interface (view) and to synchronize model changes with its views and vice versa by a controlling logic part (controller). The data model for the editor can be chosen freely, but usually GEF editors are built on given EMF models. For the views, figures, and layout of the editor interface, GEF provides the graphical rendering toolkit Draw2D and suitable graphical Eclipse viewers that integrate into the workbench. Many operations and features common to most graphical editors are supported such as zooming, automatic layout of figures, drag and drop etc.

In contrast to EMF, GEF does not provide any code generation. It is rather used as a programming framework whose abstract classes have to be implemented and by the

[3] EMF project website — http://www.eclipse.org/emf

[4] GEF project website — http://www.eclipse.org/gef

developers. There exists the Graphical Modeling Framework (GMF), which integrates GEF and EMF to a model-driven approach for generating visual editors. These editors have many features, but the generated code can be more complex than plain GEF editors and it can be difficult to extend the generated editors to support visualizations of complex heterogeneous models.

The Multi-View Editor Kit (MuvitorKit) The GEF is a very abstract framework due to its independency from a special model type. Many good practices and stages of the development of GEF editors could be reused if the editors to be developed had similar architectures and if we can expect some features of the underlying model as for EMF-generated models, which we usually have. GEF in combination with EMF models is adequate for building editors with a single panel showing one diagram at a time. However, for heterogeneous nested visual languages whose components do not only consist of single graph-like diagrams, we have to come up with additional mechanisms to manage the different components of such models. For example, in reconfigurable Petri nets, we have the current net on the one hand, and the transformation rules on the other hand, both needing their own editor component, although a rule basically consists of some Petri nets with special relations between their parts.

We used our past experiences to generalize recurring code fragments for many editor features and to document them properly for simplifying the implementation of GEF editors. This development lead to our GEF-based framework MuvitorKit [MBE09a, MBE09b]. The MuvitorKit supports nested models and models needing multiple graphical viewers. Its architecture is designed in a way that encapsulates complex underlying mechanisms in GEF and simplifies the interaction and integration with the Eclipse workbench.

In general, implementing a GEF editor involves two main tasks. On the one hand, you have to use the GEF architecture to implement controllers that mediate between an (EMF) model element and the (Draw2D) view part. On the other hand, the editor has to be integrated properly into the Eclipse workbench. To keep the second task as simple as possible, MuvitorKit contains abstract classes for building editors and graphical classes with GEF's editing capabilities, which need only the GEF-specific information to be implemented. In short, most parts that are not GEF-related and integrate the editor into the workbench are already configured reasonably in MuvitorKit to provide many features to every MuvitorKit implementation with little effort for the developer. Nevertheless, the editor can be controlled via well-documented special methods in the abstract classes, following our requirement for encapsulating good practices in simple-to-use methods. The central part of every editor based on MuvitorKit is a tree-based editor component that provides an outline of the whole model. This outline component is properly integrated into the workbench as a base to access all the graphical views for specific nested model elements.

Note that — in contrast to GMF, where you specify and generate an editor via an abstract model — the aim of MuvitorKit is to help users with easy-to-use and extendable default implementations for building complex GEF editors. Most generalizations of editor features are only possible because we assume to have a generated EMF model and make use of the generated code's special features. The whole MuvitorKit framework and its parts are tailored to be used together with an EMF model.

The animation package supports complex animations of the model that are synchronized in all the views that belong to the editor and showing a part of the model.

The RONEditor

Reconfigurable Object Nets (RONs) are a simplified variant of higher-order nets (cf. Sect. 4.6) with two types of tokens — P/T nets as "object nets" and P/T net transformation rules — to integrate transition firing and rule-based transformation of P/T nets [BEHM07, BM08, EFS11]. The (higher-order) RONs have special types of transitions with fixed semantics such as firing of object net transitions, transporting object net tokens through the RON, applying transformation rules to object nets, and splitting object nets.

The RONEditor[5] is a visual editor and simulator for RONs as an Eclipse plugin based on EMF, GEF, and the MuvitorKit.[6] To simulate the transformations on object nets, the RON editor uses the AGG engine with a converter from object nets and rules to attributed graphs and graph transformation rules.

The Graph Transformation Engine AGG

Although AGG[7] is not an Eclipse plugin, it provides useful functionalities, which we use in other Eclipse-based tools. AGG is a development environment for attributed graph transformation systems [Tae04], which supports the algebraic approach to graph transformation from [EEPT06]. The user interface provides editor components for graphs and transformation rules, which may be optionally typed over a type graph. In the recent version of AGG, nested application conditions, interaction schemes with maximal matching, and graph constraints are supported as well [RET11].

The transformation engine of AGG can be used without AGG's graphical interface via an programming interface as a back-end engine in other tools. For example, we use the AGG engine in the RONEditor for finding matches for the object net transformation rules and to compute the results of transformations on object nets.

Support for Algebraic High-Level Nets in Eclipse Editors

The diploma thesis [Fis10] developed an AHLI net editor environment as Eclipse plugins based on EMF, GEF, and MuvitorKit. The environment contains a specification editor for defining custom algebraic signatures, that can be used for the AHLI nets in the editor. This specification editor has been generated with the help of the Eclipse Xtext framework that is used for developing (textual) domain-specific languages, optionally in close relation to EMF models of the languages. The carrier sets and the operation's

[5] RONEditor — An editor for Reconfigurable Object Nets — http://tfs.cs.tu-berlin.de/roneditor

[6] Actually, MuvitorKit evolved within the RONEditor, which was the first MuvitorKit editor implementation.

[7] AGG — The Attributed Graph Grammar System — http://tfs.cs.tu-berlin.de/agg

semantics for net signatures are given and integrated into the editor by suitable Java classes.

A current diploma thesis is extending the AHLI net editor environment by AHLI net transformations with rules and interaction schemes with maximal matching, both featuring nested application conditions. For the simulation of the transformations in the AHLI net editor, the transformation engine of AGG is used similarly as in the RONEditor by encoding AHLI nets and transformation rules as attributed graphs and graph transformation rules.

8.3. CPEditor — A Visual Eclipse-Based Tool for Designing, Simulating, and Validating Communication Platform Models

This section describes the implementation and functions of the CPEditor in detail. Note that some parts are currently under development and will be finished soon.

8.3.1. Tool Architecture

We are implementing the CPEditor as a plugin for the Eclipse platform based on the MuvitorKit plugin (see Fig. 8.1). Additionally, we use functionalities and plugins described in Sect. 8.2.

A signature and algebra for the platform AHLI nets have to be provided as an external text file in the specification format of [Fis10] and a compatible implementing Java class that has to be registered in the plugin information of the CPEditor. For accessing the signature and algebra and for evaluating terms, we reuse several core helper classes from the AHLI editor in [Fis10].

For persistency operations on edited models, the MuvitorKit loads and saves instances of the classes that have been generated for an EMF meta model of Communication Platform models. The main editor component is — as in every MuvitorKit implementation — a tree-based overview of the edited model, which serves as the entry point to opening graphical views for the AHLI nets, rules, interactions schemes, constraints, and handler expressions.

The graphical views of the CPEditor are implemented as GEF editor components working on the currently edited EMF platform model instance. For the integration of these views into the CPEditor and the Eclipse workbench and for accessing them, we use the methods and abstract implementations of graphical GEF viewers provided by the MuvitorKit. Note that the support of the MuvitorKit classes for graphical views encompassing multiple separate parts considerably helps in building editor components for rules and nested constraints.

For checking the applicability of rules and interaction schemes at the current AHLI platform net and for performing them, the simulation component of the CPEditor translates the nets, rules and schemes to corresponding counterparts over attributed graphs. The transformation engine of AGG is used to find valid matches for the translated rules

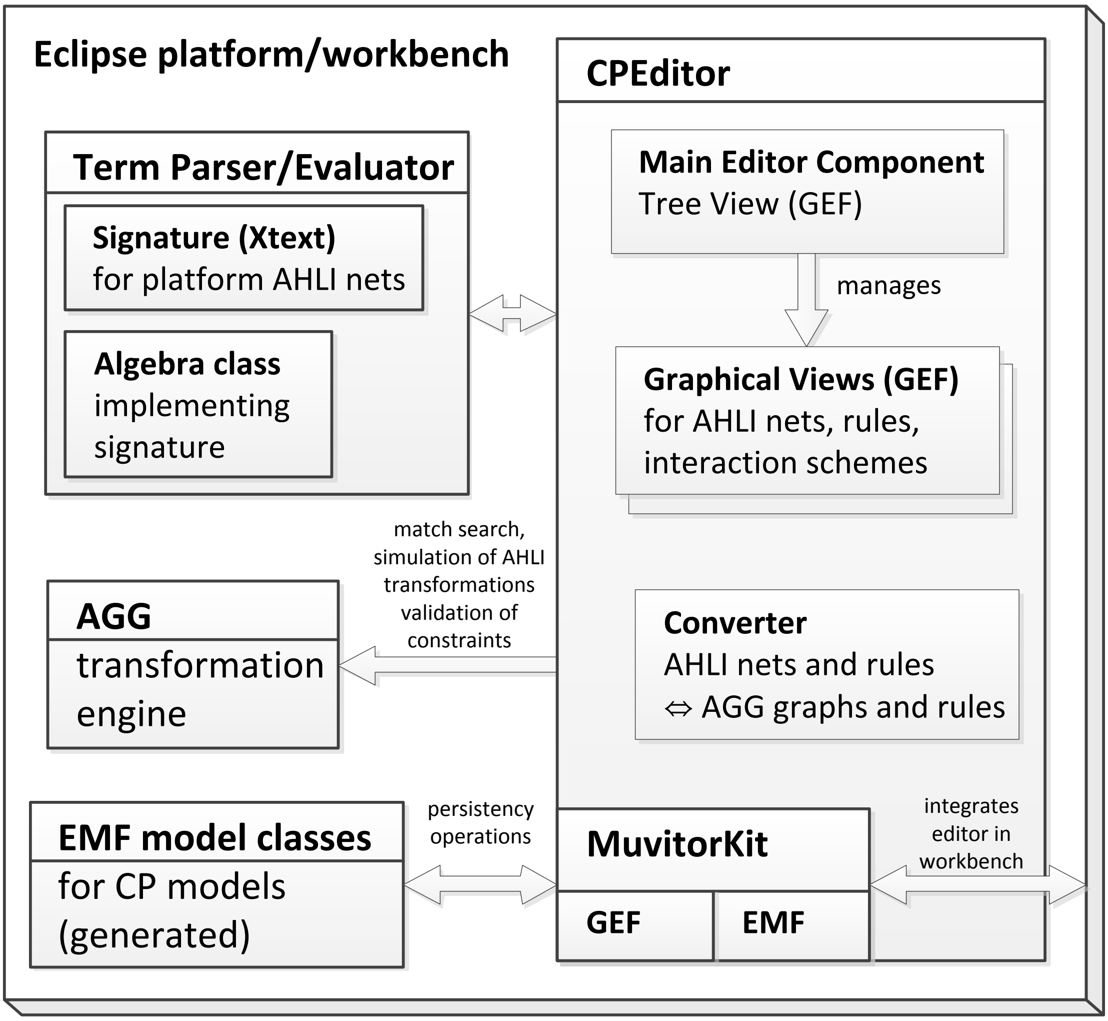


Figure 8.1.: Architecture of the CPEditor

and to calculate the results of the transformations at these matches. By a special translation of the AHLI nets to attributed graphs that exploits the possibility to encode data in node and arc attributes, we ensure that only such graph matches are found by AGG for the translated graph rule that can be interpreted as proper AHLI net matches.

8.3.2. Editing of Communication Platform Models

A screenshot of the CPEditor editor is shown in its current implementation stage in Fig. 8.2. Because the extended AHLI editor is the basis for the CPEditor and some features are still to be added, some parts still refer to AHLI nets rather than CP. In the file managing component marked by [1] that is called “Navigator”, files with the suffix “.ahl” can be created and opened, which invokes the CPEditor’s main view. This view [2] shows a tree-based overview of all model elements. In the example in Fig. 8.2, we see a specification “Nat” that has been set for all rules and nets. The tree elements below

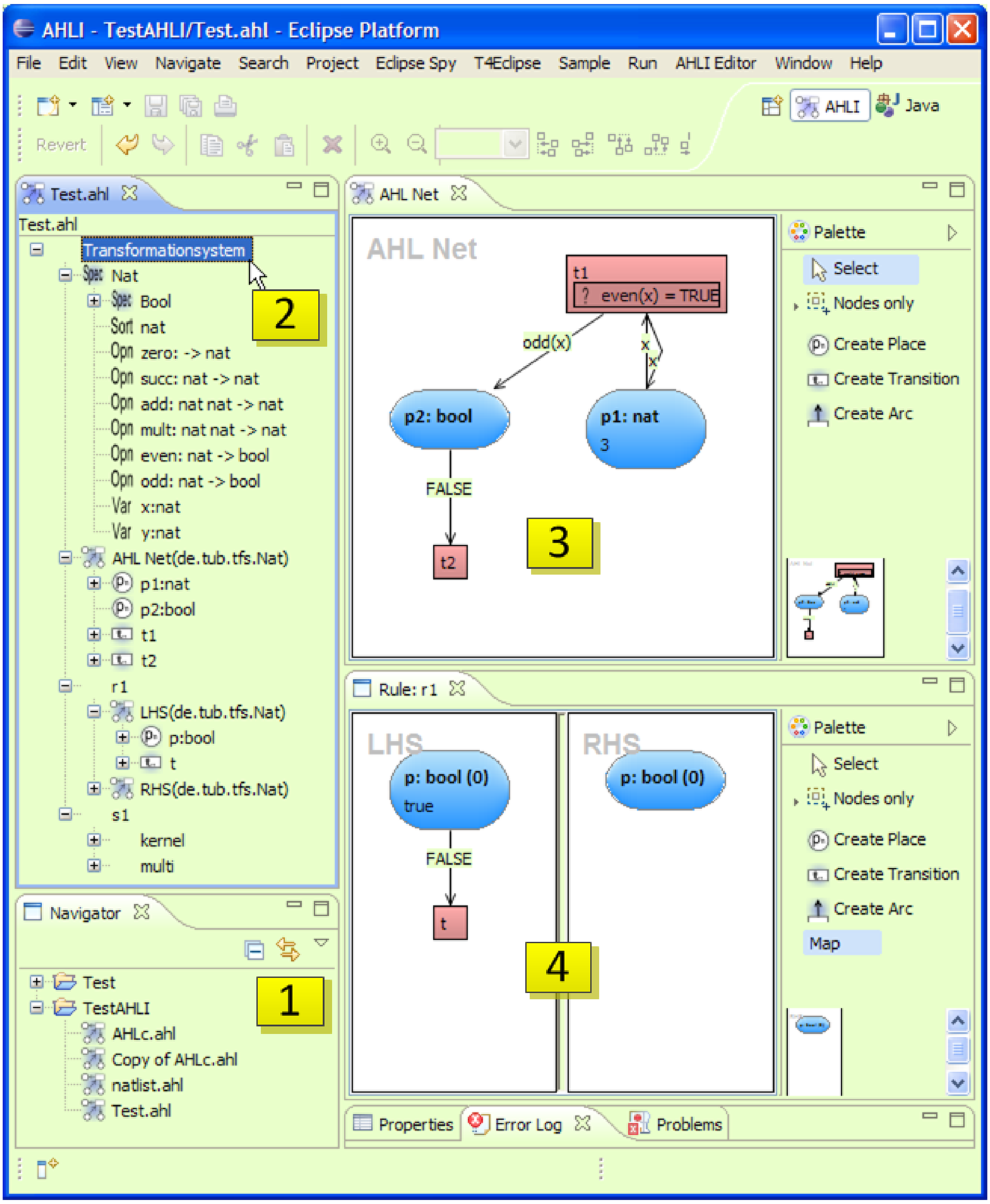


Figure 8.2.: Screenshot of the currently extended AHLI editor

“Nat” stand for a subspecifiation “Bool” followed by the sort “nat” and the operations and variables that are defined in the specification “Nat”. Then, there are an AHLI net, which is also shown graphically in the component [4], a rule “r1” shown graphically in component [3], and an interaction schema “s1” with its kernel rule and a multi rule, which are not shown graphically.

In the tree editor [2], one can create AHLI nets, rules and interaction schemes via the context menu. Note that for new Communication Platform models, also the specification and the implementing algebra Java class has to be set via this context menu.

AHLI nets can be edited graphically by opening the graphical view for a net from the tree editor like [3]. Places, transitions, and arcs are created by the corresponding tools in the palette to the right. When creating a place, the editor asks for a sort of the specification for the new place. For created arcs, the editor lets the user build a term expression according to the sort of the connected place. Tokens and firing conditions are added via the context menu on the corresponding place or transition. In the example in Fig. 8.2, the place $p1$ carries an integer token of the value 3 and the transition $t1$ has a firing condition $even(x) = TRUE$.

Transformation rules are edited as AHLI nets, but the graphical view for a rule like in [4] shows the LHS and the RHS of a rule and allows to map places, transitions, and tokens by a tool in the palette. The mappings are denoted by numbers in parentheses after the names of the net elements. Via the context menu in the tree editor, one can create nested application conditions for rules. Application conditions can either be boolean operators or a morphism with one of the four quantifiers $\exists, \neg\exists, \forall, \neg\forall$. For editing a condition morphism, a graphical view like [4] for rules can be opened from the tree editor that displays the morphism's domain and codomain.

Interaction schemes are edited as rules by opening the kernel and multi rules in view like [4]. The user is not supposed to manipulate the kernel morphisms; we rather implemented the editor such that it automatically synchronizes all multi rules over the kernel rule to keep the kernel morphisms consistent. This means that if an element is created in the kernel rule, then in all multi rules of the interaction scheme, a corresponding element is created that is mapped by the created kernel element. Similarly, deleting elements in the kernel rule deletes all corresponding elements in the multi rules as well. Elements in multi rules that are in the image of a kernel morphism cannot be deleted.

8.3.3. Simulation

In the current version of the CPEditor, one can choose an AHLI net in the tree editor and apply a rule or an interaction scheme on it. The editor translates the selected AHLI net to an AGG graph and tries to apply a translation of the selected rule or interaction scheme to it. The result of a transformation in AGG is then reflected to the selected AHLI net. Interaction schemes are matched maximally disjoint and injective, and application conditions are respected. It is possible to use variables of the signature as token values in the rules.

Transitions can be fired, which is realized in the editor by the application of a rule that simulates the effects of the firing step (cf. [Fis10]). Unfortunately, up to now the firing conditions of a transition are not evaluated when firing it.

Remark (Formal aspects of translating AHLI nets to AGG graphs). For PTI nets, [MEE11] defines a functor translating PTI nets and rules to typed attributed graphs as used in AGG and shows that it is an $\mathcal{M}$-functor. The recent concept of $\mathcal{M}$-functor guarantees that a functor not only preserves transformations but also creates them. For PTI nets, this means that if there exists a transformation for the translated graph and graph rule of a PTI net and a PTI net rule a transformation, then the transformation is also possible for the PTI side. We used this translations already for implementing net transformations in the RONEditor to ensure that the graph matches calculated by AGG for the translated rules are also valid net matches. For the AHLI nets in the CPEditor, we use a translation based on the essential principles for translating PTI nets and rules by an $\mathcal{M}$-functor. A formal proof that this translation of AHLI nets to graphs in the CPEditor yields an $\mathcal{M}$-functor is future work.

8.3.4. Current Development Status

In the current stage of implementation, the CPEditor supports graphical editing of AHLI nets and transformation by rules and interactions schemes with token variables and nested application conditions. For completely covering the requirements give in Sect. 8.1, we are working on implementing the remaining features in the CPEditor as follows.

Editing For selected suitable specification operations, it is possible to define handler expressions over the existing rules and interaction schemes that have a token for this operation in their LHS. For the validation, platform constraints can be created and edited as the application conditions for rules.

Simulation Transitions for which there are enough tokens in their predomain are marked as enabled. There might be some free variables to be bound to algebraic values which are used to check the firing condition before firing the transition. For a simulation step of a user action in the Communication Platform model, the editor checks if a firing step produces a token request for which a handler expression has been defined. The editor then automatically applies the handler expression. Furthermore, a special section in the tree editor shows the known users, which are identified by an identifier place that is incident with arcs that are inscribed with the reserved user variable u (see Sect. 5.1.1.1), only. For the known users, the enabled transitions are shown as subnodes. With this, simulations can be performed intuitively via the tree editor without searching for transitions in the possibly complex graphical view of the platform AHLI net.

Validation While simulating a model, the editor gives a warning when an applied handler expression is not functional on the current net (see Sect. 6.2). For the independence of user actions, there are two possibilities. On the one hand, one can select two enabled transitions from the tree editor and let the editor perform a check for parallel independence of the user actions (including reconfigurations by token request handlers). On the other hand, for one selected transition the user can let the editor perform a check for sequential independence with the immediately preceding user action in the history. In both cases, a detailed diagnosis report on the satisfied independence conditions from

Sect. 6.3.3 is given. The independence checks are implemented by using API functions of the AGG tool for independence of graph transformations.

9. Related Work

Gli incontri, gli scontri, lo scambio di opinioni,
persone che son fatte di nomi e di cognomi,
venghino signori, che qui c'è il vino buono,
le pagine del libro e le melodie del suono.
Si vive di ricordi, signori, e di giochi,
di abbracci sinceri, di baci e di fuochi.

(Caparezza)

We recapitulate various related work that is cited in different parts of this thesis.

9.1. Human-Centric Communication Spaces and Platforms

Human-centric engineering with a focus on technology rather than on respecting human-related features in models is the goal of the Project Oxygen [Rud01]. The concept of human-centricity for modeling, where the individual characteristics of the users are to be respected and represented in models, goes back at least three decades [Ric83]. The idea of awareness for human demands lead to the notion of I-centric communications [AMSPZ01, Arb03, Ste03] where the communication systems are expected to adapt themselves to the needs of its users and their communication spaces (cf. Sect. 2.2.3). In the context of I-centric communications, the term of an individual's communication space denotes the entirety of objects this individual might want to interact with. The definition of Communication Spaces on which this thesis builds on (see Sect. 2.3.1) instead follows the perspective of communication spaces for multi-agent system in [PSH06], where a communication space is defined over a set of agents; although the authors do not refer to human-centric systems.

The roots of modeling distributed systems lie in the early seventies out of which advanced concepts such as compositional actor systems [Agh86] have been developed. This development culminated in a standard reference model of open distributed processing [IOS96, IOS98], whose definitions for fundamental terms such as "actions", "communication", or "behavior" have become basic knowledge for system modelers (see Sect. 2.2.1), not least because of the wide adoption of the UML for modeling purposes.

In the context of communication modeling, the term "platform" has been used in [MSM97, BGM+03, Sie04] in the sense of hardware architectures and in [LNW03] in the sense of abstraction layers, where the last does not make any assumptions on the nature of the designed applications (cf. Sect. 2.5).

High-Level Modeling of Communication Platforms and Skype

The intention of high-level modeling of Communication Platforms that this thesis follows is similar to the idea of uncoupling communication from other system features by “interface-based design” in [RSV97]. The principles of Peer-to-Peer networks and their peers [MKL$^+$02] are close to the concept of how we represent users from a high-level modeling viewpoint in Communication Platforms. The possible behavior of user in a a high-level model fits well with the idea of describing a system by Life Sequence Charts in [HM03], i. e., to give a set of possible user actions in an interface and their resulting perceivable effects.

The articles [HK99, Nar02] focus on descriptive rather than on normative modeling of user behavior.

The Skype application is not shipped with an official documentation. Besides a collection of frequently asked questions and how-to guides on the Skype website, there are books such as [AAD07] as the main resources describing the details of how to use Skype and its many features. Most research articles on Skype have their focus on analyzing technical and quantitative aspects of Skype’s network behavior [DK06, ZZZY10, BS04]. A related approach on modeling Skype with transformation systems of attributed typed graphs is given in [KHTR10], still for analyzing network properties rather than user behavior.

These related work articles are discussed in more detail in Sect. 3.2.3.

9.2. Formal Modeling Approaches

The Unified Modeling Language (UML) [OMG10] as a heterogeneous technique comprises the most common and influential modeling approaches for various aspects of software systems. Prominent process languages and algebras for modeling interaction in concurrent communication systems [BK84, Hoa85, Mil80, Pra91, Mil99]. Both are shortly considered in Sect. 2.4.2.

Graph Transformations

An overview of graph transformation approaches is given in [Roz97]. The approach that we refer to in this thesis is the algebraic graph transformation approach with double pushouts from [EEPT06] and its generalized categorical framework of $\mathcal{M}$-adhesive systems (the former “weak adhesive transformation systems”).

Advanced concepts for transformation systems Application conditions are discussed in [EH85] and [HHT96] for graph transformations. The nested application conditions for $\mathcal{M}$-adhesive systems stem from [HP05, HP09].

Amalgamated graph transformations have been introduced in [BFH87] together with an amalgamation theorem. Amalgamated transformations and the amalgamation theorem have been generalized to $\mathcal{M}$-adhesive systems in [GEH10, Gol11]. In [LETE04], the authors describe how to simulate a kind of Petri nets by amalgamated graph transformations.

An overview of control structures for graph transformations is given in [HEET99]. Transformation units [Kus98, KK99] are a powerful formalism to specify application sequences of graph rules by textual expressions. Other approaches for controlling transformation sequences use graphical notations such as UML activity diagrams [EGLT11] or high-level Petri nets [HEM05].

Petri Nets

The thesis that initially introduced Petri nets [Pet62] has had a rich offspring [Rei85a, RT86, MOM89, MM90, Win87, Bau90]. Petri nets have been used as a standard technique for modeling concurrent systems. There have been approaches of modeling communication systems with Petri nets, as well [ERRW03]. However, these system were static and mostly aimed at other levels of abstraction than our high-level modeling of Communication Platforms. The same holds for the approach of modeling multicasting operations with Petri nets in [PT07, LL95, JGJ$^+$00] (see Sect. 3.2.5.1 for details).

High-level Petri nets The general concept of high-level Petri nets has been founded in [Rei91, JR91], which lead to Coloured Petri nets [Jen92, Jen94, Jen97]. A similar approach are the algebraic high-level (AHL) nets [EPR94, EHP$^+$02], which rely on signatures and algebras [EM85] rather than on ML for the data tokens as in Coloured Petri nets (cf. Sect. 2.4.2).

Petri Nets with individual tokens The notion of individual tokens in Petri nets has been used and discussed in several domains [GR83, Rei85b, BMMS99, Gla05, GP95, GP09] (see Chap. 4 for details).

Relations of graph transformations and the firing behavior of Petri nets with individual tokens — albeit they are not explicitly denoted by this term — have been discussed in [Kre81, CM95] (cf. the remark for Sect. 4.1.3).

Dynamic Petri nets The idea of Petri nets whose structure can change during runtime came up first in [Val78]. Since then, there have been several concepts of dynamic Petri nets [SB94, HP00, KR07, HB08, AB09] (for details see the paragraph on Petri nets in Sect. 2.4.2).

Reconfigurable Petri nets Rule-based transformation of Petri nets based on double pushouts and $\mathcal{M}$-adhesive systems has been developed for P/T nets [EHP$^+$07, PEHP08] and for AHL nets [PER95, Pra08]. The basic idea of Petri nets as tokens in Petri nets for the technique of higher-order nets in [HEM05] stems from [Val98].

10. Conclusion

Non ho voglia di andare d'accordo, ho voglia di andare.
D'accordo? Di andare. D'accordo?

(Caparezza)

In this chapter, we conclude the thesis by summarizing the results of this thesis and by pointing out possible extensions as future work.

10.1. Main Results

With the approach of Communication Platform models, we have developed a modeling approach that integrates and extends techniques and formal foundations of algebraic specification, high-level Petri nets, and graph transformations. The behavioral semantics of Communication Platform models is considered as the firing behavior of a specially constructed higher-order Petri net for the corresponding platform model.

The approach of Communication Platform models respects general modeling principles that resulted in considerations of Skype as a Communication Platform with typical features. These principles focus on high-level abstraction and representation of the possible actions of a platform's users as transitions in a platform Petri net. The users of a platform can be identified easily by special places with identifier tokens on it representing a user's name that are connected with the user's action transitions. Performing user actions involves the firing of such an action transition followed by an optional reconfiguration of the platform net, which may change data tokens and the possible actions of users. This model view on platforms totally abstracts from particular user interfaces and communication technologies and addresses the general modeling of features that are typical for all Communication Platforms.

We have developed techniques for validating properties that are interesting for general Communication Platforms regarding the structural integrity of platforms and the independence of user actions. These techniques are based on formal concepts and results of constraints and independence of transformations in $\mathcal{M}$-adhesive systems. The notions of independence for transformations in $\mathcal{M}$-adhesive systems have been extended considerably to be applicable for user actions in Communication Platforms as combinations of firing steps and transformation sequences.

The case study on Skype reconsiders the modeling requirements regarding the approach of Communication Platform models. A detailed modeling of the most important features of Skype demonstrates that Communication Platform models are adequate for modeling human-centric communication-based systems such as Skype following the general modeling principles for Communication Platforms. Moreover, the new validation techniques have been applied to example scenarios in the Skype Communication Platform model to validate interesting properties of Skype models that have been identified in the initial discussion of Skype as a Communication Platform.

Finally, we have outlined the development of an Eclipse-based tool for editing, simulating, and validating Communication Platforms and have reported on the current state of this development.

10.2. Summary of Conceptual and Technical Results

We summarize the main conceptual and technical results that we have achieved in this thesis, regarding the main goal of this thesis of developing a high-level formal modeling approach for Communication Platforms.

Communication Spaces and Platforms

In Chap. 2, we have introduced and discussed the notion of Human-Centric Communication Spaces (HCCS) as an abstract concept and view of general communication-based and human-related systems. We set the focus of this thesis to modeling Communication Platforms (CP), which we understand as a special kind of concrete communication systems that adapt well to the principles of HCCS. A survey over popular formal modeling techniques w. r. t. the main aspects of HCCS (content and contextuality, topology, and interaction) evaluated possible options for an integrated modeling technique of CP.

Conceptual Results In Sect. 2.5, Communication Platforms have been characterized as a notion that encompasses heterogeneous communication-based systems such as Skype, Second Life, and Facebook. From the survey over modeling techniques and the following choice of reconfigurable high-level Petri nets as a promising candidate for an integrated modeling approach covering the main aspects of HCCS, we have formulated general principles for modeling CP adequately with this approach in Sect. 3.2.6.

Transformation of Petri Nets with Individual Tokens

As a basis for models of Communication Platforms, we have presented a new Petri net formalism based on P/T nets and AHL nets in Chap. 4 called “Petri nets with individual tokens” (in contrast to “collective tokens”), together with a rule-based transformation approach that is based on the graph transformation approach with double pushouts.

Conceptual Results The transformation of Petri nets with individual tokens based on double pushouts overcomes several restrictions that are inconvenient in terms of usability and intuitiveness of the transformation approach. More precisely, in contrast to the collective approach, we can formulate rules for nets with individual markings that can change these markings independently from the structural context of the marked place. Such marking-changing rules are essential on the one hand for modeling communication systems and platforms (especially for realizing multicasting of data as we have discussed in Sect. 3.2.5.1) and on the other hand for the following technical result of correspondence of firing steps and firing rule applications.

Technical Results We have characterized the applicability of PTI and AHLI net transformation rules by gluing conditions (see Theorems 4.1.9 and 4.2.10), which are sufficient and necessary. For the rule-based transformations of P/T nets and AHL nets with individual tokens (called PTI and AHLI nets, respectively), we have shown the important result that firing steps correspond to the application of special firing rules (see Theorems 4.1.11 and 4.2.12). Moreover, token-injective PTI and AHLI net morphisms preserve firing behavior (see Theorems 4.1.12 and 4.2.13). The individual net classes are syntactically and semantically compatible to each other (see Theorem 4.5.4) and to their collective counterparts (see Cors. 4.4.6 and 4.4.7 and Theorems 4.4.8 and 4.4.10), which we have shown by suitable functors.

Another main technical results is the instantiation of the framework of $\mathcal{M}$-adhesive transformation systems for reconfigurable PTI and AHLI nets (see Theorem 4.3.3 and Theorems 4.3.4 and 4.3.5).

Communication Platform Models

For coping with some challenges for an adequate modeling of CP, we have defined Communication Platform models (see Def. 5.3.3) as an extension of reconfigurable AHLI nets, i. e., a net representing the actual system configuration and rules that manipulate the net's structure and marking.

We tailor this approach to Communication Platform models by extending it with the advanced transformation concepts of amalgamated transformations, variable application conditions, and token request handlers in order to deal with the principal challenges of reconfigurable Petri nets that we gathered in Sect. 3.2.7. Moreover, we present higher-order nets with a specially suited algebra as a control structure to combine firing steps and reconfigurations to user actions of a Communication Platform model.

Conceptual Results In Sect. 5.1, we have concretized the general principles for modeling CP with reconfigurable Petri nets to more specific concepts of representing Communication Platforms as reconfigurable AHLI nets, regarding the representation of actions and their owners, modeling the perception of data by users, and relating reconfigurations to firing steps. The token request handlers of Defs. 5.2.17 and 5.2.19, which express the relation of reconfigurations to their triggering firing steps, are the basis for the definition of user actions in Communication Platform models in Def. 5.3.1.

We represent the behavioral semantics of a particular Communication Platform model as the firing behavior of a algebraic higher-order net for which we have given a construction in Sect. 5.4.

Technical Results We have employed the advanced transformation concepts of nested application conditions and amalgamated rules over interaction schemes for Communication Platform models. Because of a technical incompatibility of nested application conditions with rules over a term algebra with variables that we have described in Sect. 5.2.1.1, we have adapted these concepts. First, as a solution for rules of reconfigurable AHLI nets, we have defined variable application conditions (VAC, see Def. 5.2.5) with a structural satisfaction (see Def. 5.2.6). In Theorem 5.2.10, we have stated the conceptual correctness of the VAC and their relation to the original nested application conditions for $\mathcal{M}$-adhesive systems. Secondly, we have loosened the definition of interactions schemes and amalgamation for rules in Defs. 5.2.12 and 5.2.13 so that interactions schemes can be formulated with VACs. In Def. 5.2.15, we have stated that loosely amalgamated rules are well-defined and have the desired result when applied.

Validation Techniques

Regarding model properties about model integrity and action independence that we have discussed in Sect. 3.2.1 and which are interesting from the general perspective of Communication Platforms, we have defined several concepts that help the platform designer to validate such properties.

Conceptual Results Based on the VACs for rules of Communication Platforms models, we have adapted the definition of graph constraints to structural platforms constraints in (see Def. 6.1.2) as a means to express structural conditions that can be checked for an AHLI net that represents the current configuration of a platform model. Such structural properties are useful to check and ensure e. g.the uniqueness of identities in platform models or the success of user actions.

We have defined parallel and sequential independence of user actions in Communication Platform models in Defs. 6.3.9 and 6.3.10, which consist of several conditions based on the independence of the action's underlying firing and transformation steps. Another important property for the independence of user actions is the functionality of the token request handlers that occur in a certain user action (see Def. 6.2.1). The staged conditions of the different grades of user action independence are useful for locating errors and inconsistencies in defective models.

Technical Results For the structural platform constraints, Theorem 6.1.3 states the relation to the general notion of graph constraints. In Theorem 6.3.13, we have shown that the parallel and sequential independence of user actions are compatible to each other. This result builds on the permutability of firing steps (see Theorem 6.3.3)and a local Church-Rosser theorem for rule applications and firing steps that we have proven in Theorem 6.3.8, which in turn relies on the correspondence of AHLI firing steps and rule applications shown in Theorem 4.2.12.

Modeling Case Study: Skype

For the model case study, we have reconsidered the modeling requirements for Skype regarding the approach of Communication Platform models and have defined the extent of the case study and the features of Skype that it shall cover. An extensive and thorough formal modeling of these parts of Skype has been realized with a Communication Platform model. For this model, a typical example scenario of user actions in Skype has been played through, which we also used to demonstrate the validation techniques for Communication Platform models.

Conceptual Results The modeling case study shows how many typical features of Communication Platforms that are offered by Skype can be modeled with the Communication Platform model approach following the general modeling principles for Communication Platforms in Sect. 3.2.6. With the high-level abstraction from interfaces and technical details, these concepts can directly be used to model similar typical features such as sending messages, exchanging and managing contacts, and performing calls and conference, in other concrete Communication Platforms.

The example validation of properties in Sect. 7.4 for the Skype model demonstrate how the technical concepts and results from Chap. 6 can be applied to validate model properties regarding model intergrity and independence of user actions. As discussed in Sect. 3.2.1), these validations can be generalized to properties that are interesting for arbitrary Communication Platforms in general.

10.3. Future Work

> An incurable optimist is one who says: "Everything is for the best; mankind will survive. And even if mankind doesn't survive, it is still for the best." Then there is what I would call a *pessimistic* optimist. A pessimistic optimist is one who sadly shakes his head and says: "I'm very much afraid that everything is for the best!"
>
> *(Raymond Smullyan)*

Regarding the results of this thesis, there are some interesting issues and promising ideas to be further examined and elaborated as future work.

Structural constraints for $\mathcal{M}$-adhesive systems The technical problem of matching variables in rules and formulating proper application conditions described in Sect. 5.2.1.1 is present in all $\mathcal{M}$-adhesive transformation systems of structural objects that are integrated with algebraic data and variables. The presented variable application conditions and structural constraints are a solution for AHLI nets to circumvent this problem for modeling. Unfortunately, for these varied concepts, it is not obvious how to achieve important general results such as the lemma for equivalent application conditions that are shifted over morphisms or rules from [HP05, HP09]. Instead of solving this problem for AHLI nets, only, the new concepts and results shown in this thesis should first be generalized to arbitrary $\mathcal{M}$-adhesive systems. A first step towards a general solution for this problem would be a categorical formalization of the integration of structural models with algebraic data such as AHL(I) nets or attributed graphs. For these approaches, it is easy to project on structure and data by forgetful functors, which is a promising starting point for the categorical characterization of data–structure factorization of morphisms.

Structuring and variations in token request handlers In the Communication Platform models modeling case study on Skype from Sect. 7.2, we could reuse rules in more than one request handler (cf. the remark on page 129). But it also occurs that the rules used in the token request handlers are equal to other rules (or their combinations) up to slight differences in their marking. For example, the effect of the rule *CreatePermittedChannel* in Fig. 7.15 is intended to be the combination of the effects of th rules *ExtendToPermittedChannel* in Fig. 7.14 and *RuleCreateDirectChannel* in Fig. 7.7. We cannot simply reuse these two rules from the other request handlers, because they expect to match different request tokens. For a better modularization of the request handlers and reusability of the rules, we could define operators on transformation rules for varying the markings in the rules' components consistently, especially the consumed request tokens.

For improving the usability of request handlers and w. r. t. modeling more complex reconfigurations in Communication Platforms, further constructs from control structures for transformation systems such as “if-then-else”, “as along as possible”, and (recursive) nesting of named expressions may be considered for extending the definition of valid request handler expressions.

Static analysis of communication platform models The presented validation concerning the degrees of user action independence consider concrete user actions in actual scenarios. A huge improvement for checking model properties is the step from case-based analysis to static analysis, i. e., verifying properties by conditions on the model without regarding concrete configurations. The concept of proving local confluence by checking critical pairs of transformation rules from [EEPT06] recently has been extended to rules with nested application conditions in [EHL^{+}10a]. For transformation systems, the approach of critical pairs is useful to prove the independence of two rules in all possible applications. By taking the specialties of user actions as consecutive firing and transformation steps into consideration, conditions for local confluence of user actions possibly can be found by considering critical pairs of the user action's components.

Proof of correctness for tool translation As mentioned in Sect. 8.3.3, we employ the graph transformation tool AGG for calculating transformation results and for analyzing independence of rule applications. For this, we translate AHLI nets and rules to typed attributed graphs and graph rules, because AGG works in the domain of graphs, only. The results calculated by AGG are then interpreted as AHLI nets again. This translation is based on the same principles that are used in the RONEditor for simulating PTI net transformation with AGG and in [MEE11] for defining a functor translating PTI nets and rules to typed attributed graphs. This functor has been shown to be an $\mathcal{M}$-functor, which is a useful property for proving that independent translated graph transformations imply that the originating PTI transformations are independent. Although the translation from AHLI nets to AGG works well, it is still outstanding to show that it corresponds to an $\mathcal{M}$-functor.

Formal specification of actions and roles The definition of Communication Platform models in Def. 5.3.3 is based on the technique of transformations of general AHLI nets. For certain features of Communication Platforms such as the ownership of actions, we require that they be represented structurally in the platform net in Sect. 3.2.6, for which we propose to use arcs with a reserved arc variable to a place carrying an identifier token in Sect. 5.1.1.1. This approach is an implicit way of specifying actions. A better approach for building platform models based on informal specifications might be to support explicit definition of action sets and roles for certain contexts in the formal model. For this, Communication Platform models could be extended by diagrams similar to Fig. 7.1 playing the role of detailed use case diagrams from the UML. The AHLI platform nets and handler rules then could be checked for consistency with the action–role models to verify that the platform specification is covered by the Communication Platform model.

Interaction between communication platforms Generally, the concepts and results from this thesis may serve as a first approach to support the modeling of more concepts of HCCS (cf. page 27) such as adaption of systems to special human needs, which lies beyond the aims of the work at hand. But there is an abundance of open issues to be treated already for our focus field of Communication Platforms.

The integration of several Communication Platform models is an interesting subject that goes towards a more general support for the notion of HCCS (see Sect. 2.3.1), which states that users can be simultaneously in several Communication Spaces and even transfer content between spaces. The higher-order net controlling the actions and reconfigurations in one Communication Platform could be extended to support multiple platform nets at the same time. With a relation between the representations or identities of a particular user in different platforms, the “known data” or contacts of users could be transferred between the platforms. A candidate for a technical approach to this extension is the rule-based transformation of the higher-order net extended by concepts of distributed graph transformation [EOP06]. Distributed graphs are (network) graphs that are interpreted as diagrams of some category. A concrete example are network graphs whose nodes are graphs again and whose arcs are graph morphisms. Distributed graph transformation manipulates the network graph and the local objects at the same time. By transferring this idea to higher-order nets, rules manipulating the higher-order net could access and distribute data (tokens) between the platform nets, which lie as tokens on the places in the higher-order net. With this, it would be possible to model actions such as multicasting data between the different platforms that are controlled by the higher-order net if the platform nets have suitable structural interfaces recognized by the rules. Unfortunately, first considerations of this idea suggest that distributed graph transformation systems are only $\mathcal{M}$-adhesive if their rules do not change the network graph, which limits the generality of this approach. Nevertheless, for the purpose of synchronizing and distributing data between platform nets in the controlling higher-order net, this is sufficient.

A. Appendices

A.1. Signatures and Algebras

A.1.1. Common Signature and Algebra for Communication Platform Models

The following signature Σ_{CP} in Table A.1 and the Σ_{CP}-algebra A^{CP} in Table A.2 are described informally in Sect. 5.1.3.

$\Sigma_{CP} =$		
sorts:	$ID, Data, Control, Bool, Int, BoolInt$	
opns:	$T, F\colon \to Bool$	
	$\bullet\colon \to Control$	
	$0\colon \to Int$	
	$neq\colon ID\ ID \to Bool$	
	$make\colon Bool\ Int \to BoolInt$	
	$inc\colon Int \to Int$	
	$dec\colon Int \to Int$	
	$and\colon Bool\ Bool \to Bool$	
	$or\colon Bool\ Bool \to Bool$	
	$not\colon Bool \to Bool$	
vars:	$u, c, c', \ldots : ID$	$i, i', \ldots : Int$
	$b, b', \ldots : Bool$	$d, d', \ldots : Data$
tokenvars:	$u1, u2, \ldots : ID$	$i1, i2, \ldots : Int$
	$b1, b2, \ldots : Bool$	$d1, d2, \ldots : Data$

Table A.1.: Signature Σ_{CP} for Communication Platform Models

The algebra A^{CP} — which in the following we denote only as A — consists of the following components:

carrier sets:
$A_{ID} = \{A, B, C, \ldots, a, b, c, \ldots, 0, 1, 2, \ldots, 9\}^*$
$A_{Data} = \{A, B, C, \ldots, a, b, c \ldots, 0, 1, 2, \ldots, 9, ?, !, \ldots\}^*$
$A_{Control} = \{\bullet\}$ $\quad$ $A_{Bool} = \{T, F\}$
$A_{Int} = \mathbb{Z}$ $\quad$ $A_{BoolInt} = A_{Bool} \times A_{Int}$

constants:
$T_A = T \in A_{Bool}$ $\quad$ $F_A = F \in A_{Bool}$
$\bullet_A = \bullet \in A_{Control}$ $\quad$ $0_A = 0 \in A_{Int}$

functions:
$neq_A \colon A_{ID} \times A_{ID} \to A_{Bool}$, with $neq_A(i, i') = \ i \neq i'$
$make_A \colon A_{Bool} \times A_{Int} \to A_{BoolInt}$, with $make_A(b, i) = (b, i)$
$inc_A \colon A_{Int} \to A_{Int}$, with $inc_A(i) = i + 1$
$dec_A \colon A_{Int} \to A_{Int}$, with $dec_A(i) = i - 1$
$and_A \colon A_{Bool} \times A_{Bool} \to A_{Bool}$, with $and_A(b, b') = b \wedge b'$
$or_A \colon A_{Bool} \times A_{Bool} \to A_{Bool}$, with $or_A(b, b') = b \vee b'$
$not_A \colon A_{Bool} \to Bool$, with $not_A(b) = \neg b$

Table A.2.: Σ_{CP}-algebra A^{CP} for Communication Platform Models

A.1.2. Algebra for AHOI Nets Controlling Platform Models

Consider a concrete Communication Platform model

$$CPM = (\Sigma_{ANI}, A_{ANI}, ANI, R, IS, S_{Req}, handler)$$

For the token net values of an Σ_{CP}-algebra (see Table 5.2), we first need to fix countable universes I_0, P_0, T_0 of individual tokens and names for places and transitions for all token nets — suitable for ANI — in order to have a proper carrier set of token nets.

We introduce some shorthand notations for the algebra definition:

- For some algebra $\hat{A}$, we denote by $Nets(I_0, P_0, T_0, \hat{A})$ the set of all AHLI nets N with $P_N \subseteq P_0, T_N \subseteq T_0, I_N \subseteq I_0, A_N = \hat{A}$.

- The morphism class

  $$\mathcal{M} = \{f \in Mor_{\mathbf{AHLINets}(\mathbf{\Sigma}_{ANI})} \mid f_\Sigma = id_\Sigma, f_A \text{ isomorphic, and } f_P, f_T, f_I \text{ injective}\}$$

  is according to the $\mathcal{M}$-adhesive category in Theorem 4.3.4.

- The set RUL contains all rules with variable application conditions $(L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R, ac)$ such that $L, K, R \in Nets(I_0, P_0, T_0, T_\Sigma(Y))$ for some token variable subset $Y \subseteq Y_{\Sigma_{ANI}}$, ac is a variable application condition (see Def. 5.2.5), and $l, r \in \mathcal{M}$.

- The set $MATCH$ contains all morphisms $o\colon L \to N$ such that o is almost injective ($o^S \in \mathcal{M}$), $L \in Nets(I_0, P_0, T_0, T_\Sigma(Y))$ for some token variable subset $Y \subseteq Y_{\Sigma_{ANI}}$, and $N \in Nets(I_0, P_0, T_0, A_{ANI})$.

The Σ_{AHOI}-Algebra A^{AHOI}_{cpm} for a control AHOI net (see Sect. 5.4) is defined as in Table A.3.

Table A.3.: Σ_{AHOI}-algebra A^{AHOI}_{CPM} for an AHOI net controlling a given platform model CPM

The algebra A^{AHOI}_{CPM} — which in the following we denote only as A — consists of the following components:

carrier sets:

$A_{System} = Nets(I_0, P_0, T_0, A_{ANI}) \cup \{undef\}$

$A_{Rules} = RUL \cup \{(s_i\colon \varrho_0 \to \varrho_i)_{1\leq i\leq n} \mid \varrho_0, \varrho_i \in RUL \wedge s_i$ is kernel morphism (see Def. 5.2.12)$\}$

$A_{Trans} = T_0$

$A_{VarAsg} = \{asg\colon X \to A_{ANI} \mid X \subseteq X_{\Sigma_{ANI}}\}$

$A_{Sel} = \{(M, m, N, n) \mid M, N \subseteq I_0 \wedge m\colon M \to P_0 \wedge n\colon M \to P_0\}$

$A_{Mor} = MATCH \cup \{(o_k\colon \hat{L}_k \to N)_{0\leq k\leq j} \mid o_k \in MATCH\}$

$A_{ReqConstr} = Constr(S_{Req})$ (see Def. 5.2.16)

$A_{MorSeq} = \{(o_k)_{0\leq k\leq m} \mid o_k \in A_{Mor}\}$

$A_{Bool} = \{T, F\}$

constants:

$T_A = T \in A_{Bool}$ $\qquad F_A = F \in A_{Bool}$

functions:

$isEnabled_A\colon A_{System} \times A_{Trans} \times A_{VarAsg} \times A_{Sel} \to A_{Bool}$, with

$$isEnabled_A(n, t, asg, s) = \begin{cases} T & \text{, if } n \neq undef, (t, asg) \in CT_n, \\ & \text{and } (t, asg) \text{ enabled under } s \\ F & \text{, else} \end{cases}$$

$fire_A\colon A_{System} \times A_{Trans} \times A_{VarAsg} \times A_{Sel} \to A_{System}$, with

$$fire_A(n, t, asg, s) = \begin{cases} n' & \text{, if } enabled_A(n, t, asg, s) = T \\ undef & \text{, else} \end{cases}$$

where n' results from the firing step $n \overset{t,asg,s}{\rightarrowtail} n'$.

$producesReq_A\colon A_{Trans} \times A_{System} \to A_{Bool}$, with

$$producesReq_A(t, n) = \begin{cases} T & \text{, if } \exists req \in A_{ReqConstr} \text{ such that} \\ & producesReq_A(t, n, req) = T \\ F & \text{, else} \end{cases}$$

$cod_A\colon A_{Mor} \to A_{System}$, with $cod_A(o) = N$

where $o = (\hat{o}\colon L \to N) \in MATCH$ or $o = (o_k\colon \hat{L}_k \to N)_{0\leq k\leq j}$

$isApplicable_A \colon A_{Rules} \times A_{Mor} \to A_{Bool}$, with

$$isApplicable_A(r,o) = \begin{cases} T & \text{, if either } r \in RUL, o \in MATCH, \\ & (r,o) \text{ satisfy the gluing condition} \\ & \text{in } \textbf{AHLInets} \text{ (see Def. 4.2.9),} \\ & \text{and } o \text{ structurally satisfies the VAC of } r \\ & \text{ (see Def. 5.2.6),} \\ & \text{or otherwise } r \notin RUL, o \notin MATCH, \\ & \text{and } o \text{ is a maximal disjoint} \\ & \text{matching for } r \text{ (see Def. 5.2.14)} \\ F & \text{, else} \end{cases}$$

$apply_A \colon A_{Rules} \times A_{Mor} \to A_{System}$, with

$$apply_A(r,o) = \begin{cases} N & \text{, if } isApplicable_A(r,o) = T \\ undef & \text{, else} \end{cases}$$

where N results from $cod_A(m) \overset{r,o}{\Longrightarrow} N$, which is either a direct transformation along inclusions (for $r \in RUL$) or an application of an interaction scheme along inclusions (for $r \notin RUL$, see Def. 5.2.15).

$producesReq_A \colon A_{Trans} \times A_{System} \times A_{ReqConstr} \to A_{Bool}$, with

$$producesReq_A(t,n,req) = \begin{cases} T & \text{, if } t \in T_n \text{ and} \\ & \exists (\tau,p) \in POST_n(t) \colon \tau = req(\dots) \\ F & \text{, else} \end{cases}$$

$isHandleable_A \colon A_{System} \times A_{ReqConstr} \times A_{MorSeq} \to A_{Bool}$, with

$$isHandleable_A(n,req,os) = \begin{cases} T & \text{, if } os \text{ is a valid match sequence of} \\ & handle_{CPM}(req) \text{ on } n \text{ (see Def. 5.2.18)} \\ F & \text{, else} \end{cases}$$

$handle_A \colon A_{System} \times A_{ReqConstr} \times A_{MorSeq} \to A_{System}$, with

$$handle_A(n,req,os) = \begin{cases} n' & \text{, if } isHandleable_A(n,req,os) = T \\ undef & \text{, else} \end{cases}$$

where N results from the transformation sequence $n \xRightarrow{handle_{CPM}(req),os} n'$ (see Def. 5.2.19).

Table A.3.: Σ_{AHOI}-algebra A^{AHOI}_{CPM} for an AHOI net controlling a given platform model CPM

A.1.3. Algebra for the Skype Case Study

The following Σ_{Skype}-algebra A^{Skype} in Table A.4, which we use for the Skype Communication Platform model $SCPM$, and its signature Σ_{Skype} are described informally in Sect. 7.2.1. Note that the values of the sorts for request constructors simply wrap values of the sorts for identifiers, data, and histories.

The algebra A^{Skype} — which in the following we denote only as A — consists of the following components:

carrier sets:

The sorts A_{ID}, A_{Data}, $A_{Control}$, A_{Int}, $A_{BoolInt}$ of the subsignature Σ_{CP} are the same as in the algebra A^{CP} in Table A.2

$A_{State} = \{Online, Offline\}$

$A_{Option} = \{ContactsOnly, Anyone\}$

$A_{Setting} = A_{Option} \times A_{Option}$

$$\begin{aligned} A_{History} = \{&Empty, sentIM(i,d,h), blocked(i,h), unblocked(i,h),\\ &called(i,i',h), part(i,h), invited(i,i',h), hungup(i,h),\\ &forwarded(i,i',h) \mid i,i' \in A_{ID} \wedge d \in A_{Data} \wedge h \in A_{History}\} \end{aligned}$$

$$\begin{aligned} A_{ComRequest} = \{&directReq(i,i'), callReq(i,i'), hangupReq(i,i'),\\ &forReq(i,i',i''), groupReq(i,i',i''), exGReq(i,i',i''),\\ &callGReq(i), quitGReq(i), addGReq(i,i'), partGReq(i),\\ &leaveGReq(i), hangupGReq(i) \mid i,i',i'' \in A_{ID}\} \end{aligned}$$

$$\begin{aligned} A_{ContactRequest} = \{&contactReq(i,i'), permReq(i,i'), delCXReq(i,i'),\\ &delCReq(i,i'), undelCReq(i,i') \mid i,i' \in A_{ID}\} \end{aligned}$$

$A_{TransmitRequest} = \{transGReq(i,d), transReq(i,d) \mid i \in A_{ID} \wedge d \in A_{Data}\}$

constants:

$Online_A = Online \in A_{State}$ $\quad$ $Offline_A = Offline \in A_{State}$

$ContactsOnly_A = ContactsOnly \in A_{Option}$ $\quad$ $Anyone_A = Anyone \in A_{Option}$

$Empty_A = Empty \in A_{History}$

functions:

$makeSetting_A \colon A_{Option} \times A_{Option} \to A_{Setting}$, with $makeSetting_A(o,o') = (o,o')$

The functions for the operations that yield values from the set $A_{History}$ simply wrap the parameters in a compound term with the operation's name, e. g., $sentIM_A \colon A_{ID} \times A_{Data} \times A_{History} \to A_{History}$ is defined as $sentIM_A(i,d,h) = sentIM(i,d,h)$.

The same holds for the request constructors. For example, the function $directReq_A \colon A_{ID} \times A_{ID} \to A_{ComRequest}$ is defined as $directReq_A(i,i') = directReq(i,i')$.

Table A.4.: Σ_{Skype}-algebra A^{Skype}

A.2. Gluing Conditions and Initial Pushouts

A.2.1. Categorical Gluing Condition with Initial Pushouts

The following definitions, which can also be found in [EEPT06], provide notions to formulate an abstract categorical condition, the so-called gluing condition, which is necessary and sufficient for the (unique) existence of pushout complements in $\mathcal{M}$-adhesive categories. The gluing condition is used to show that the corresponding set-theoretical gluing conditions in the categories **PTINets** (see Def. 4.1.8) and **AHLINets** (see Def. 4.2.9) are necessary and sufficient conditions for the application of transformation rules $\varrho = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ in suitable $\mathcal{M}$-adhesive categories $(\mathbf{PTINets}, \mathcal{M})$ (see Fact A.2.16) and $(\mathbf{AHLINets}, \mathcal{M})$ (see Fact A.2.19), respectively.

Definition A.2.1 (Boundary, Initial Pushout)
Given a morphism $f\colon L \to G$ in an $\mathcal{M}$-adhesive category $(\mathbf{C}, \mathcal{M})$, a morphism $b\colon B \to L$ with $b \in \mathcal{M}$ is called the *boundary* over f if there is a pushout complement of f and b such that (1) is a pushout that is *initial* over f. Initiality of (1) over f means, that for every pushout (2) with $b' \in \mathcal{M}$ there exist unique morphisms $b^*\colon B \to D$ and $c^*\colon C \to E$ with $b^*, c^* \in \mathcal{M}$ such that $b' \circ b^* = b, c' \circ c^* = c$ and (3) is a pushout. B is then called the *boundary object* and c the *context* with respect to f.

$$\begin{tikzcd}
B \arrow[r, "b"] \arrow[d] \arrow[dr, phantom, "(1)"] & L \arrow[d, "f"] \\
C \arrow[r, "c"'] & G
\end{tikzcd}
\qquad
\begin{tikzcd}
B \arrow[r, "b^*"] \arrow[rr, bend left, "b"] \arrow[d] \arrow[dr, phantom, "(3)"] & D \arrow[r, "b'"] \arrow[d] \arrow[dr, phantom, "(2)"] & L \arrow[d, "f"] \\
C \arrow[r, "c^*"] \arrow[rr, bend right, "c"'] & E \arrow[r, "c'"] & G
\end{tikzcd}$$

Definition A.2.2 (Categorical Gluing Condition)
Let $l\colon K \to L \in \mathcal{M}$ and $f\colon L \to G$ be morphisms in a given $\mathcal{M}$-adhesive category $\mathbf{C}$ with initial pushouts. We say that l and f satisfy the categorical gluing condition if for the initial pushout (1) over f there exists a morphism $b^*\colon B \to K$ such that $l \circ b^* = b$.

$$\begin{tikzcd}
B \arrow[r, "b"'] \arrow[rr, bend left, "b^*"] \arrow[d, "g"'] \arrow[dr, phantom, "(1)"] & L \arrow[d, "f"] & K \arrow[l, "l"] \arrow[r, "r"'] & R \\
C \arrow[r, "c"'] & G & &
\end{tikzcd}$$

Given a production $\varrho = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$, we say ϱ and f satisfy the categorical gluing condition, iff l and f satisfy the categorical gluing condition. △

Fact A.2.3 (Categorical Gluing Condition)
Given an $\mathcal{M}$-adhesive category $(\mathbf{C}, \mathcal{M})$ with initial pushouts, a match $f\colon L \to G$ satisfies the categorical gluing condition with respect to $l\colon K \to L \in \mathcal{M}$ (or a production $\varrho = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$, respectively) if and only if the context object D exists, i. e., there is a pushout complement (2) of l and m:

$$\begin{tikzcd}
B \arrow[r, "b"'] \arrow[rr, bend left, "b^*"] \arrow[d, "g"'] \arrow[dr, phantom, "(\mathbf{1})"] & L \arrow[d, "f"] \arrow[dr, phantom, "(\mathbf{2})"] & K \arrow[l, "l"'] \arrow[r, "r"'] \arrow[d, "k"] & R \\
C \arrow[r, "c"] \arrow[rr, bend right, "c^*"'] & G & D \arrow[l, "d"'] &
\end{tikzcd}$$

If it exists, the context object D is unique up to isomorphism.

Proof. See Theorem 6.4 in [EEPT06]. □

In order to show initiality of a pushout (1) over a morphism $f\colon L \to G$ with corresponding opposite morphism $B \to C$ in an $\mathcal{M}$-adhesive category $\mathbf{C}$ (cf. Def. A.2.1), one has to show for every pushout (2) over f with corresponding opposite morphism $D \to E$ that there exist unique morphisms $b^*\colon B \to D$ and $c^*\colon C \to E$ forming a pushout (cf. Def. A.2.1). The following Lemma states that the existence of the required morphism c^* induces the remaining requirements.

Lemma A.2.4 (Morphism-Pushout-Lemma)
Given pushouts (1) and (2) in an $\mathcal{M}$-adhesive category $\mathbf{C}$, where $b, d \in \mathcal{M}$. If there is a morphism $c^*\colon C \to E$ with $c = e \circ c^*$ then this c^* is unique and $c^* \in \mathcal{M}$ and there exists a unique $b^*\colon B \to D \in \mathcal{M}$ with $b = d \circ b^*$ such that (3) is pushout in $\mathbf{C}$.

$$\begin{tikzcd}
B \arrow[r, "b"] \arrow[d, "g"'] \arrow[dr, phantom, "(\mathbf{1})"] & L \arrow[d, "f"'] \arrow[dr, phantom, "(\mathbf{2})"] & D \arrow[l, "d"'] \arrow[d, "h"] \\
C \arrow[r, "c"] \arrow[rr, bend right, "c^*"'] & G & E \arrow[l, "e"']
\end{tikzcd}
\qquad
\begin{tikzcd}
B \arrow[r, "b^*"'] \arrow[rr, bend left, "b"] \arrow[d, "g"'] \arrow[dr, phantom, "(\mathbf{3})"] & D \arrow[r, "d"'] \arrow[d, "h"'] \arrow[dr, phantom, "(\mathbf{2})"] & L \arrow[d, "f"] \\
C \arrow[r, "c^*"] \arrow[rr, bend right, "c"'] & E \arrow[r, "e"] & G
\end{tikzcd}$$

Proof. We use in the following that also $c, e \in \mathcal{M}$ due to pushouts (1) and (2) and the fact that $\mathcal{M}$-morphisms are closed under pushouts.

uniqueness of c^*: Let $\bar{c}\colon C \to E$ with $c = e \circ \bar{c}$. Then there is $e \circ \bar{c} = c = e \circ c^*$, which by the fact that e is a monomorphism implies that $\bar{c} = c^*$.

unique morphism b^*: Because $d \in \mathcal{M}$ the pushout (2) is also a pullback and because (1) is a pushout there is $f \circ b = c \circ g = e \circ c^* \circ g$ from the pullback property follows that there is a unique morphism $b^*\colon B \to D$ with $c^* \circ g = h \circ b^*$ and $b = d \circ b^*$ leading to commuting diagram (3).

$$\begin{tikzcd}
B \arrow[r, "b^*"'] \arrow[rr, bend left, "b"] \arrow[d, "g"'] \arrow[dr, phantom, "(\mathbf{3})"] & D \arrow[r, "d"'] \arrow[d, "h"'] \arrow[dr, phantom, "(\mathbf{2})"] & L \arrow[d, "f"] \\
C \arrow[r, "c^*"] \arrow[rr, bend right, "c"'] & E \arrow[r, "e"] & G
\end{tikzcd}$$

$b^*, c^* \in \mathcal{M}$: The morphisms b and c are $\mathcal{M}$-morphisms. So the fact that $b = d \circ b^*$ and $c = e \circ c^*$ together with the fact that $\mathcal{M}$-morphisms are closed under decomposition implies that $b^*, c^* \in \mathcal{M}$.

(3) is pushout: Because (1) is a pushout, (2) is a pullback, and $e, b^* \in \mathcal{M}$, this follows directly from item 2 of Thm. 4.26 in [EEPT06] ($\mathcal{M}$ pushout-pullback decomposition lemma). □

In the following two lemmas, we show that the constructions $(\_^\sharp \otimes \_)^\oplus$ and $\mathcal{P}_{fin}(\_^\sharp)$, which are used in the pre and post conditions, and conditions of AHLI nets, respectively, are compositional. We need these properties for the proofs of Fact A.2.18 and Fact A.2.19.

Lemma A.2.5 (Compositionality of $(\_^\sharp \otimes \_)^\oplus$)
Given three AHLI nets

$$AN_i = (\Sigma_i, P_i, T_i, pre_i, post_i, cond_i, type_i, I_i, A_i) \text{ where } (i \in \{1,2,3\})$$

together with signature morphisms $f_\Sigma \colon \Sigma_1 \to \Sigma_2$, $g_\Sigma \colon \Sigma_2 \to \Sigma_3$ *and functions* $f_P \colon P_1 \to P_2$, $g_P \colon P_2 \to P_3$ *with* $type_2 \circ f_P = f_S \circ type_1$, *then*

$$\left(g_\Sigma^\sharp \otimes g_P\right)^\oplus \circ \left(f_\Sigma^\sharp \otimes f_P\right)^\oplus = \left((g_\Sigma \circ f_\Sigma)^\sharp \otimes (g_P \circ f_P)\right)^\oplus$$

Proof. Let $\sum_{i=1}^{n}(term_i, p_i) \in (T_{\Sigma_1}(X_1) \otimes P_1)^\oplus$.
Due to the freeness of $T_{\Sigma_1}(X_1)$ there is $V_{f_\Sigma}\left(g_\Sigma^\sharp\right) \circ f_\Sigma^\sharp = (f_\Sigma \circ g_\Sigma)^\sharp$,
and we have

$$\begin{aligned}
&\left(g_\Sigma^\sharp \otimes g_P\right)^\oplus \left(\left(f_\Sigma^\sharp \otimes f_P\right)^\oplus \left(\sum_{i=1}^{n}(term_i, p_i)\right)\right)\\
&= \left(g_\Sigma^\sharp \otimes g_P\right)^\oplus \left(\sum_{i=1}^{n}\left(\left(f_\Sigma^\sharp\right)_{type_1(p_i)}(term_i), f_P(p_i)\right)\right)\\
&= \sum_{i=1}^{n}\left(\left(g_\Sigma^\sharp\right)_{type_2(f_P(p_i))}\left(\left(f_\Sigma^\sharp\right)_{type_1(p_i)}(term_i)\right), g_P(f_P(p_i))\right)\\
&= \sum_{i=1}^{n}\left(\left(g_\Sigma^\sharp\right)_{f_S(type_1(p_i))}\left(\left(f_\Sigma^\sharp\right)_{type_1(p_i)}(term_i)\right), g_P(f_P(p_i))\right)\\
&= \sum_{i=1}^{n}\left(V_{f_\Sigma}\left(g_\Sigma^\sharp\right)_{type_1(p_i)}\left(\left(f_\Sigma^\sharp\right)_{type_1(p_i)}(term_i)\right), g_P(f_P(p_i))\right)\\
&= \sum_{i=1}^{n}\left(\left(V_{f_\Sigma}\left(g_\Sigma^\sharp\right) \circ f_\Sigma^\sharp\right)_{type_1(p_i)}(term_i), g_P(f_P(p_i))\right)\\
&= \sum_{i=1}^{n}\left((g_\Sigma \circ f_\Sigma)^\sharp_{type_1(p_i)}(term_i), g_P \circ f_P(p_i)\right)\\
&= \left((g_\Sigma \circ f_\Sigma)^\sharp \otimes (g_P \circ f_P)\right)^\oplus \left(\sum_{i=1}^{n}(term_i, p_i)\right)
\end{aligned}$$

□

Lemma A.2.6 (Compositionality of $\mathcal{P}_{fin}(\_^{\sharp})$)
Given three AHLI nets

$$AN_i = (\Sigma_i, P_i, T_i, pre_i, post_i, cond_i, type_i, I_i, A_i) \textit{ where } (i \in \{1,2,3\})$$

together with signature morphisms $f_\Sigma \colon \Sigma_1 \to \Sigma_2$ and $g_\Sigma \colon \Sigma_2 \to \Sigma_3$.
Then for all $E \in \mathcal{P}_{fin}(Eqns(\Sigma_1))$ there is

$$\mathcal{P}_{fin}(g_\Sigma^\sharp)(\mathcal{P}_{fin}(f_\Sigma^\sharp)(E)) = \mathcal{P}_{fin}((g_\Sigma \circ f_\Sigma)^\sharp)(E)$$

Proof. Let $E \in \mathcal{P}_{fin}(Eqns(\Sigma_1))$. For $e = (t_l, t_r) \in Eqns(\Sigma_1)$ the extension $f_\Sigma^\sharp$ of a signature morphism $f_\Sigma \colon \Sigma_1 \to \Sigma_2$ to equations of Σ_1 is defined by $f_\Sigma^\sharp(e) = ((f_\Sigma^\sharp)_s(t_l), (f_\Sigma^\sharp)_s(t_r))$, where s is the sort of terms t_l and t_r, i. e., $t_l, t_r \in T_{\Sigma_1}(X_1)_s$. The function $(f_\Sigma^\sharp)_s$ on the right hand side of the equation is the extension of f_Σ to terms of type s, i. e.$(f_\Sigma^\sharp)_s \colon T_{\Sigma_1}(X_1)_s \to V_{f_\Sigma}(T_{\Sigma_2}(X_2))_s$. Due to the definition of the forgetful functor V_{f_Σ} there is $V_{f_\Sigma}(T_{\Sigma_2}(X_2))_s = T_{\Sigma_2}(X_2)_{f_S(s)}$ and hence there is $(f_\Sigma^\sharp)_s \colon T_{\Sigma_1}(X_1)_s \to T_{\Sigma_2}(X_2)_{f_S(s)}$. So we have

$$\begin{aligned}
&\mathcal{P}_{fin}\left(g_\Sigma^\sharp\right)\left(\mathcal{P}_{fin}\left(f_\Sigma^\sharp\right)(E)\right)\\
&= \mathcal{P}_{fin}\left(g_\Sigma^\sharp\right)\left(\mathcal{P}_{fin}\left(f_\Sigma^\sharp\right)(\{e \mid e \in E\})\right)\\
&= \mathcal{P}_{fin}\left(g_\Sigma^\sharp\right)\left(\left\{\left(\left(f_\Sigma^\sharp\right)_s(t_l), \left(f_\Sigma^\sharp\right)_s(t_l)\right) \mid (t_l, t_r) \in E\right\}\right)\\
&= \left\{\left(\left(g_\Sigma^\sharp\right)_{f_S(s)}\left(\left(f_\Sigma^\sharp\right)_s(t_l)\right), \left(g_\Sigma^\sharp\right)_{f_S(s)}\left(\left(f_\Sigma^\sharp\right)_s(t_r)\right)\right) \mid (t_l, t_r) \in E\right\}\\
&= \left\{\left(V_{f_\Sigma}\left(g_\Sigma^\sharp\right)_s\left(\left(f_\Sigma^\sharp\right)_s(t_l)\right), V_{f_\Sigma}\left(g_\Sigma^\sharp\right)_s\left(\left(f_\Sigma^\sharp\right)_s(t_r)\right)\right) \mid (t_l, t_r) \in E\right\}\\
&= \left\{\left(\left(V_{f_\Sigma}\left(g_\Sigma^\sharp\right) \circ f_\Sigma^\sharp\right)_s(t_l), \left(V_{f_\Sigma}\left(g_\Sigma^\sharp\right) \circ f_\Sigma^\sharp\right)_s(t_r)\right) \mid (t_l, t_r) \in E\right\}\\
&= \left\{\left((g_\Sigma \circ f_\Sigma)_s^\sharp(t_l), (g_\Sigma \circ f_\Sigma)_s^\sharp(t_r)\right) \mid (t_l, t_r) \in E\right\}\\
&= \mathcal{P}_{fin}\left((g_\Sigma \circ f_\Sigma)^\sharp\right)(E)
\end{aligned}$$

□

In the following two lemmas, we show that the constructions $\_ \otimes \_$ and $\_^\sharp$ preserve monomorphisms, which is used in the proofs of Fact A.2.18 and Fact A.2.19.

Lemma A.2.7 ($\_ \otimes \_$ preserves Monomorphisms)
Given two AHLI nets

$$AN_i = (\Sigma_i, P_i, T_i, pre_i, post_i, cond_i, type_i, I_i, A_i) \textit{ where } (i \in \{1,2\})$$

and a monomorphism $f = (f_P, f_T, f_\Sigma, f_A, f_I) \colon AN_1 \to AN_2$ in **AHLINets**.
Then $f_A \otimes f_P \colon A_1 \otimes P_1 \to A_2 \otimes P_2$ is a monomorphism in **Sets**.

Proof. Let $(a_1, p_1), (a_2, p_2) \in A_1 \otimes P_1$ with $(f_A \otimes f_P)(a_1, p_1) = (a, p) = (f_A \otimes f_P)(a_2, p_2)$. Then there is

$$\begin{aligned}(f_{A,type_1(p_1)}(a_1), f_P(p_1)) &= (f_A \otimes f_P)(a_1, p_1) = (a, p)\\ &= (f_A \otimes f_P)(a_2, p_2) = (f_{A,type_1(p_2)}(a_2), f_P(p_1))\end{aligned}$$

which means that $f_{A,type_1(p_1)}(a_1) = a = f_{A,type_1(p_2)}(a_2)$ and $f_P(p_1) = p = f_P(p_2)$. Because f is a monomorphism in **AHLINets** the function f_P is injective implying that $p_1 = p_2$. This means that

$$f_{A,type_1(p_2)}(a_1) = f_{A,type_1(p_1)}(a_1) = a = f_{A,type_1(p_2)}(a_2)$$

and because f is a monomorphism in **AHLINets** there is $f_{A,type_1(p_2)}$ injective implying that $a_1 = a_2$. Hence there is $(a_1, p_1) = (a_2, p_2)$ implying that $f_A \otimes f_P$ is injective, i. e., it is a monomorphism in **Sets**. □

Lemma A.2.8 ($\_^\sharp$ preserves Monomorphisms)
Given a signature morphism $f\colon \Sigma_1 \to \Sigma_2$. If f is a monomorphism in **Sig** *then $f^\sharp\colon T_{\Sigma_1}(X_1) \to V_f(T_{\Sigma_2}(X_2))$ is a monomorphism in* $\mathbf{Alg(\Sigma_1)}$.

Proof. Because monomorphisms in $\mathbf{Alg(\Sigma_1)}$ are exactly the injective homomorphisms it is sufficient to show that $f^\sharp$ is injective. Let $t_1, t_2 \in T_{\Sigma_1}(X_1)_s$ with $f^\sharp_s(t_1) = t = f^\sharp_s(t_2)$. The recursive definition of terms allows us to prove this property by a structural induction over $t \in V_f(T_{\Sigma_2}(X_2))_s = T_{\Sigma_2}(X_2)_{f_S(s)}$.

Basis. Consider $t = x \in X_{2,f_S(s)}$. This means that there are $x_1, x_2 \in X_{1,s}$ with

$$f^\sharp_s(x_1) = x = f^\sharp_s(x_2) \quad\Leftrightarrow\quad f_X(x_1) = x = f_X(x_2)$$

and because f_X is injective this means $t_1 = x_1 = x_2 = t_2$.
Consider the other case $t = c$ with $c\colon \to f_S(s) \in OP_2$. This means that for $i = 1, 2$ there are $c_i\colon \to s \in OP_1$ with

$$f^\sharp_s(c_1) = c = f^\sharp_s(c_2) \quad\Leftrightarrow\quad f_{OP}(c_1) = c = f_{OP}(c_2)$$

and because f_{OP} is injective this means $t_1 = c_1 = c_2 = t_2$.

Hypothesis. For $t^i \in V_f(T_{\Sigma_2}(X_2))_{s_i} = T_{\Sigma_2}(X_2)_{f_S(s_i)}$ $(i = 1, \ldots, n)$ there is

$$f^\sharp_{s_i}(t_1) = t^i = f^\sharp_{s_i}(t_2) \Rightarrow t_1 = t_2$$

Step. Let $t = op(t^1, \ldots, t^n)$ with $op\colon f_S(s_1) \ldots f_S(s_n) \to f_S(s) \in OP_2$ and $t^i \in T_{\Sigma_2}(X_2)_{f_S(s_i)}$ $(i = 1, \ldots, n)$. Then from $f^\sharp_s(t_1) = t = f^\sharp_s(t_2)$ follows that there are $op_i\colon s_1 \ldots s_n \to s \in OP_1$ $(i = 1, 2)$ such that

$$\begin{aligned}&f^\sharp_s(t_1) = t = f^\sharp_s(t_2)\\ &\Leftrightarrow f^\sharp_s(op_1(t^1_1, \ldots, t^n_1)) = op(t^1, \ldots, t^n) = f^\sharp_s(op_2(t^1_2, \ldots, t^n_2))\\ &\Leftrightarrow f_{OP}(op_1)(f^\sharp_{s_1}(t^1_1), \ldots, f^\sharp_{s_n}(t^n_1)) = op(t^1, \ldots, t^n) = f_{OP}(op_2)(f^\sharp_{s_1}(t^1_2), \ldots, f^\sharp_{s_n}(t^n_2))\\ &\Leftrightarrow f_{OP}(op_1) = op = f_{OP}(op_2) \wedge \forall i \in \{1, \ldots, n\}\colon f^\sharp_{s_i}(t^i_1) = t^i = f^\sharp_{s_i}(t^i_2)\end{aligned}$$ □

Due to the injectivity of f_{OP} there is $op_1 = op_2$ and by the induction hypothesis there is $t^i_1 = t^i_2$ for all $i \in \{1, \ldots, n\}$. Hence we have $t_1 = t_2$.

A.2.2. Initial Pushouts in Sets

In this section we define a gluing condition in the category **Sets** and show that the satisfaction of the condition for a suitable $\mathcal{M}$-adhesive category $(\mathbf{Sets}, \mathcal{M})$ is equivalent to the categorical gluing condition. This provides a necessary and sufficient condition for the existence of (unique) pushout complements in **Sets**, which is used in the corresponding facts of the set-based categories **PTINets** and **AHLINets**.

Definition A.2.9 (Gluing Condition in Sets)
Let $l\colon K \to L$ and $f\colon L \to G$ be morphisms in **Sets** with $l \in \mathcal{M}$.
We define the set of identification points $IP = \{x \in L \mid \exists x' \neq x\colon f(x) = f(x')\}$ and the set of gluing points $GP = l(K)$. We say that l and f satisfy the gluing condition if $IP \subseteq GP$. △

The following lemma provides a set-theoretical construction of pushout complements in **Sets**, whereas the category-theoretical construction of pushout complements is defined via a pushout over the boundary (see Theorem 6.4 in [EEPT06]).

Lemma A.2.10 (Pushout Complement in Sets)
Let $l\colon K \to L$ and $f\colon L \to G$ be morphisms in **Sets**.
There is a pushout complement C of l and f, if l and f satisfy the gluing condition.
If a pushout complement exists it can be computed by $C = (G \setminus f(L)) \cup f(l(K))$ together with inclusion $c\colon C \to G$ and a morphism $g\colon K \to C$ with $g(x) = f(l(x))$ for every $x \in K$.

$$\begin{tikzcd} L \arrow[d, "f"'] \arrow[dr, phantom, "(1)"] & K \arrow[l, "l"'] \arrow[d, "g"] \\ G & C \arrow[l, "c"] \end{tikzcd}$$

Proof. We define C, c and g as above. We show that (1) is pushout in **Sets**.

commutativity of (1): Let $x \in L$. Then, we have $f(l(x)) = g(x) = c(g(x))$.

universal property: Let H be a set together with morphisms $c'\colon C \to H$ and $f'\colon L \to H$ with $c' \circ g = f' \circ l$. We define a morphism $h\colon G \to H$ with

$$h(x) = \begin{cases} c'(x) & \text{, if } x \in C; \\ f'(x') & \text{, for } f(x') = x \text{ otherwise.} \end{cases}$$

For the well-definedness of h we have to check if for every $x \in G$ with $x \notin C$ there is a unique $x' \in L$ such that $f(x') = x$.

$$\begin{aligned} x \notin C &\overset{\text{def. } C}{\Leftrightarrow} x \notin (G \setminus f(L)) \cup f(l(K)) \Leftrightarrow \neg x \in (G \setminus f(L)) \cup f(l(K)) \\ &\Leftrightarrow \neg(x \in G \setminus f(L) \vee x \in f(l(K))) \Leftrightarrow x \notin G \setminus f(L) \wedge x \notin f(l(K)) \end{aligned}$$

From the fact that $x \in G$ and $x \notin G \setminus f(L)$ we have $x \in f(L)$ which means that there is $x' \in L$ with $f(x') = x$.

Let us assume that x' is not unique, i. e., there is $x'' \in L$ with $x' \neq x''$ and $f(x'') = x$. Then x' is an identification point, which implies that $x' \in GP = f(l(K))$ because l and f satisfy the gluing condition. This is a contradiction and hence x' is unique.

$$\begin{tikzcd}
 & L \arrow[d, "f"'] \arrow[dd, phantom, "(2)"', shift right=6] \arrow[ddl, bend right, "f'"'] \arrow[dr, phantom, "(1)"] & K \arrow[l, "l"'] \arrow[d, "g"] \\
 & G \arrow[dl, "h"] \arrow[d, phantom, "(3)"] & C \arrow[l, "c"] \arrow[dll, bend left, "c'"] \\
H & &
\end{tikzcd}$$

Let $x \in C$. Then there is $h \circ c(x) = h(x) = c'(x)$ which means that diagram (2) commutes.
Let $x \in L$. Then we distinguish the following cases:

- **Case 1** $x \in l(K)$: Then there is $k \in K$ with $l(k) = x$ and
$$h(f(x)) = h(f(l(k))) = h(c(g(k))) = c'(g(k)) = f'(l(k)) = f'(x)$$
- **Case 2** $x \notin l(K)$: Then there is $f(x) \notin C$ and therefore $h(f(x)) = f'(x)$.

Hence diagram (3) commutes.
For the uniqueness of h let $h' : G \to H$ with $h' \circ c = c'$ and $h' \circ f = f'$. Let $x \in G$.

- **Case 1** $x \in C$: Then $h'(x) = h'(c(x)) = c'(x) = h(x)$
- **Case 2** $x \notin C$: As we have shown above in this case there is a unique $x' \in L$ with $f(x') = x$ and we obtain $h'(x) = h'(f(x')) = f'(x') = h(x)$.

So we have for all $x \in G$ that $h'(x) = h(x)$ and hence $h' = h$. □

In the following definition and facts we define the boundary and initial pushout over a morphism $f : L \to G$ in **Sets** and show that the satisfaction of the gluing condition in **Sets** is equivalent to the satisfaction of the categorical gluing condition in an $\mathcal{M}$-adhesive category $(\mathbf{Sets}, \mathcal{M})$ where the class of monomorphisms $\mathcal{M}$ contains inclusions. A suitable class $\mathcal{M}$ is the class containing all monomorphisms in **Sets**, because **Sets** is an adhesive category (see Theorem 4.6 in [EEPT06]).

Definition A.2.11 (Boundary in Sets)
Given a morphism $f : L \to G$ in **Sets**. The boundary B of f is the set $B = IP$ of identification points together with an inclusion $b : B \to L$. △

Fact A.2.12 (Initial Pushout in Sets)
Given a morphism $f : L \to G$ in **Sets**, the boundary B of f and the pushout complement C of f and b, defined as $C = (G \setminus f(L)) \cup f(l(K))$ together with inclusion $c : C \to G$ and a morphism $g : K \to C$ with $g(x) = f(l(x))$ for every $x \in K$. Then diagram (1) is initial pushout in **Sets**.

$$\begin{tikzcd} B \arrow[r, "b"] \arrow[d, "g"'] \arrow[dr, phantom, "(\mathbf{1})"] & L \arrow[d, "f"] \\ C \arrow[r, "c"'] & G \end{tikzcd}$$

Proof.

(1) is pushout: The fact that (1) is pushout follows from Lemma A.2.10 by the fact that b and f satisfy the gluing condition. It remains to show that (1) is initial.
Let (2) be a pushout in **Sets** with $d \in \mathcal{M}$.

$$\begin{tikzcd} B \arrow[r, "b"] \arrow[d, "g"'] \arrow[dr, phantom, "(\mathbf{1})"] & L \arrow[d, "f"'] \arrow[dr, phantom, "(\mathbf{2})"] & D \arrow[l, "d"'] \arrow[d, "h"] \\ C \arrow[r, "c"'] & G & E \arrow[l, "e"] \end{tikzcd}$$

function c^*: We define a function $c^* \colon C \to E$ with $c^*(x) = y$ with $e(y) = c(x)$

well-definedness of c^*: For the well-definedness of c^* we have to show that for every $x \in C$ there is a unique $y \in E$ with $e(y) = c(x)$.
We distinguish the following cases for $x \in C$:

- **Case 1** $x \notin f(L)$: There is $c(x) = x \in G$. Since (2) is pushout in **Sets** the functions f and e are jointly surjective which by the fact that there is no $y \in L$ with $f(y) = x$ implies that there is $y \in E$ with $e(y) = x = c(x)$.

- **Case 2** $x \in f(b(B))$: Then there is $z \in B$ with $f(b(z)) = x$. From $z \in B = IP$ and the fact that E is a pushout complement of d and f follows that $z \in d(D)$, i. e., there is $z' \in D$ with $d(z') = z$. Let $y = h(z')$. Then we have

$$e(y) = e(h(z')) = f(d(z')) = f(z) = f(b(z)) = x = c(x)$$

So for every $x \in C$ there is a suitable element $y \in E$ with $e(y) = c(x)$. Since $d \in \mathcal{M}$ and pushouts in **Sets** are closed under $\mathcal{M}$-morphisms there is also $e \in \mathcal{M}$, i. e., e is injective which implies the uniqueness of y.
So c^* is well-defined. The fact that $e \circ c^* = c$ follows directly from the definition of c^*.

uniqueness of c^***, existence of** b^* **and pushout**: By Lemma A.2.4 the morphism c^* is the unique morphism with $c = e \circ c^*$ and there is a unique morphism $b^* \colon B \to D$ with $b = d \circ b^*$ such that (3) is a pushout in **Sets**.

$$\begin{tikzcd} B \arrow[r, "b^*"'] \arrow[rr, bend left, "b"] \arrow[d, "g"'] \arrow[dr, phantom, "(\mathbf{3})"] & D \arrow[r, "d"'] \arrow[d, "h"'] \arrow[dr, phantom, "(\mathbf{2})"] & L \arrow[d, "f"] \\ C \arrow[r, "c^*"] \arrow[rr, bend right, "c"'] & E \arrow[r, "e"] & G \end{tikzcd}$$

Hence diagram (1) is an initial pushout in **Sets**. □

Fact A.2.13 (Characterization of Gluing Condition in Sets)
Let $l\colon K \to L$ and $f\colon L \to G$ be morphisms in **Sets** with $l \in \mathcal{M}$.
The morphisms l and f satisfy the gluing condition in **Sets** if and only if they satisfy the categorical gluing condition.

Proof.

If. Let l and f satisfy the categorical gluing condition, i. e., for initial pushout (1) there is a function $b^*\colon B \to K$ with $l \circ b^* = b$.

$$\begin{tikzcd}
B \arrow[r, "b"'] \arrow[d, "g"'] \arrow[rr, bend left, "b^*"] \arrow[dr, phantom, "(\mathbf{1})"] & L \arrow[d, "f"] & K \arrow[l, "l"] \\
C \arrow[r, "c"'] & G &
\end{tikzcd}$$

Let $x \in IP$. Then there is $x \in B$ and $y = b^*(x) \in K$ with $l(y) = l(b^*(x)) = b(x) = x$ which means that $x \in l(K) = GP$. Hence l and f satisfy the gluing condition in **Sets**.

Only If. Let l and f satisfy the gluing condition in **Sets**, i. e., there is $IP \subseteq GP = l(K)$. We define a function $b^*\colon B \to K$ with $b^*(x) = l^{-1}(x)$. For every $x \in B$ there is $x \in IP$ which by the fact that l and f satisfy the gluing condition implies that $x \in GP = l(K)$, i. e., there is a preimage of x with respect to l. The preimage is unique because l is injective because $l \in \mathcal{M}$. Hence b is well-defined and there is $l(b^*(x)) = l(l^{-1}(x)) = x = b(x)$ which means that l and f satisfy the categorical gluing condition. □

A.2.3. Initial Pushouts in PTINets

In order to construct the initial pushout for a match $f\colon L \to NI$, we define the *boundary* over the match f, which is the minimal subnet containing all places, transitions, and individual tokens that must not be deleted by the application of a rule with left hand side L such that there exists a pushout complement.

The following facts about the gluing condition and initial pushouts hold in all $\mathcal{M}$-adhesive categories $(\mathbf{PTINets}, \mathcal{M})$ whose morphism class $\mathcal{M}$ of monomorphisms contains at least inclusions (for concrete instantiations see Sect. 4.3). The next fact completes the construction of initial pushouts for matches and shows that the construction of the boundary in Def. A.2.14 complies to the categorical notion of boundaries in Def. A.2.1.

From the construction and proof of the gluing condition for PTI nets (see Theorem 4.1.9), the corresponding result for P/T nets can be derived as a special case where the set of individual tokens is empty, which we omit.

Definition A.2.14 (Boundary in PTINets)
Given a morphism $f\colon L \to NI$ in **PTINets**. The boundary of f is a PTI net

$$B = (P_B, T_B, pre_B, post_B, I_B, m_B), \text{ where}$$

- $P_B = DP_T \cup DP_I \cup IP_P \cup P_{IP_T} \cup P_{IP_I}$

- $P_{IP_T} = \{p \in P_L \mid \exists t \in IP_T \colon p \in ENV(t)\}$
- $P_{IP_I} = \{p \in P_L \mid \exists i \in IP_I \colon p = m_L(i)\}$
- $T_B = IP_T, pre_B(t) = pre_L(t), post_B(t) = post_L(t)$
- $I_B = IP_I, m_B(i) = m_L(i)$

together with an inclusion $b \colon B \to L$.

Proof (Well-definedness).

$pre_B, post_B \colon T_B \to P_B^{\oplus}$: Let $t \in T_B$ and let $p \leq pre_B(t)$. Then there is $p \leq pre_L(t)$ which means that $p \in P_L$. Then there is $t \in IP_T$ which, by the fact that $p \in ENV(t)$, means that $p \in P_{IP_T} \subseteq P_B$.
So pre_B is well-defined. The proof for $post_B$ works completely analogously.

$m_B \colon I_B \to P_B$: Let $i \in I_B$. Then we have $i \in IP_I$ and hence $m_B(i) = m_L(i) \in P_{IP_I} \subseteq P_B$.

$b \colon B \to L$: We obtain an inclusion morphism $b \colon B \to L$ from the fact that $pre_B, post_B$ and m_B are restrictions of the respective functions in L. □

Fact A.2.15 (Initial Pushout in PTINets)
Given a morphism $f \colon L \to NI$ in **PTINets**, the boundary B of f and the PTI net

$$C = (P_C, T_C, pre_C, post_C, I_C, m_C), \text{ with}$$

- $P_C = (P_{NI} \setminus f_P(P_L)) \cup f_P(b_P(P_B))$
- $T_C = (T_{NI} \setminus f_T(T_L)) \cup f_T(b_T(T_B))$
- $I_C = (I_{NI} \setminus f_I(I_L)) \cup f_I(b_I(I_B))$
- $pre_C(t) = pre_{NI}(t), post_C(t) = post_{NI}(t)$
- $m_C(i) = m_{NI}(i)$

Then diagram (1) where $g := f|_B$ is an initial pushout in **PTINets**.

$$\begin{tikzcd}
B \arrow[r, "b"] \arrow[d, "g"'] \arrow[dr, phantom, "(1)"] & L \arrow[d, "f"] \\
C \arrow[r, "c"'] & NI
\end{tikzcd}$$

Proof. We prove the well-definedness of the PTI net C and that the construction leads to an initial pushout in the category **PTINets**.

well-definedness of C:

$pre_C, post_C \colon T_C \to P_C^{\oplus}$: Let $t \in T_C$ and let $p \leq pre_C(t)$. Due to the fact that $t \in (T_G \setminus f_T(T_L)) \cup f_T(b_T(T_B))$ we can distinguish the following cases:

Case 1 $t \notin f_T(T_L)$:

Case 1.1 $\exists p' \in P_L \colon f_P(p') = p$: Then due to the fact that there is $p \leq pre_G(t)$ there is $p' \in DP_T \subseteq P_B$ which means that $p = f_P(b_P(p')) \in P_C$.

Case 1.2 $\nexists p' \in P_L \colon f_P(p') = p$: This means that $p \in P_G \setminus f_P(P_L) \subseteq P_C$.

Case 2 $t \in f_T(b_T(T_B))$: Then there exists $t' \in T_B$ with $f_T(b_T(t')) = t$ then

$$pre_G(t) = pre_G(f_T(b_T(t'))) = f_P^{\oplus}(pre_L(b_T(t'))) = f_P^{\oplus}(b_P^{\oplus}(pre_B(t')))$$

which for $p \leq pre_G(t)$ means that $p \leq f_P^{\oplus}(b_P^{\oplus}(pre_B(t')))$ and hence $p \in f_P(b_P(P_B)) \subseteq P_C$.

The proof for $post_C$ works analogously.

well-definedness of $m_C \colon I_C \to P_C$: Let $i \in I_C$. Then there is

$$i \in (I_{NI} \setminus f_I(I_L)) \cup f_I(b_I(I_B))$$

Case 1 $i \notin f_I(I_L)$:

Case 1.1 $\exists p \in P_L \colon f_P(p) = m_{NI}(i)$: This means that $p \in DP$ and therefore $p \in P_B$ and $f_P(p) \in P_C$. So we have $m_C(i) = m_{NI}(i) = f_P(p) \in P_C$.

Case 1.2 $\nexists p \in P_L \colon f_P(p) = m_{NI}(i)$: This means that $m_{NI}(i) \notin f_P(P_L)$ and hence $m_C(i) = m_{NI}(i) \in P_C$.

Case 2 $i \in f_I(b_I(I_B))$: Then there exists $j \in I_B$ with $f_I(b_I(j)) = i$ and we have for $m_C(i) = m_{NI}(i)$ that

$$m_{NI}(i) = m_{NI}(f_I(b_I(j))) = f_P(m_L(b_I(j))) = f_P(b_P(m_B(j)))$$

and because $f_P(b_P(P_B)) \subseteq P_C$ there is $m_C(i) \in P_C$.

well-definedness of c: We obtain an inclusion morphism $c \colon C \to NI$ from the fact that $pre_C, post_C$ and m_C are restrictions of the respective functions in NI.

well-definedness of g: For $J = \{P, T, I\}$ we obtain well-defined functions $g_J \colon J_B \to J_C$ because for $j \in J_B$ there is $g_J(j) = f_J(j) = f_J(b_J(j)) \in J_C$ The morphism g preserves pre and post domains and markings because it is a restriction of f which is a well-defined **PTINets** morphism.

(1) is pushout: Due to Lemma A.2.10 the diagrams (2)-(4) are pushouts in **Sets** which implies that (1) is pushout in **PTINets** because the pushout in **PTINets** can be constructed componentwise.

$$\begin{tikzcd}
P_B \arrow[r, "b_P"] \arrow[d, "g_P"'] \arrow[dr, phantom, "(2)"] & P_L \arrow[d, "f_P"] & T_B \arrow[r, "b_T"] \arrow[d, "g_T"'] \arrow[dr, phantom, "(3)"] & T_L \arrow[d, "f_T"] & I_B \arrow[r, "b_I"] \arrow[d, "g_I"'] \arrow[dr, phantom, "(4)"] & I_L \arrow[d, "f_I"] \\
P_C \arrow[r, "c_P"'] & P_{NI} & T_C \arrow[r, "c_T"'] & T_{NI} & I_C \arrow[r, "c_I"'] & I_{NI}
\end{tikzcd}$$

initiality of (1): Given pushout (5) in **PTINets** with $d \in \mathcal{M}$.

$$\begin{tikzcd}
B \arrow[r, "b"] \arrow[d, "g"'] \arrow[dr, phantom, "\textbf{(1)}"] & L \arrow[d, "f"'] \arrow[dr, phantom, "\textbf{(5)}"] & D \arrow[l, "d"'] \arrow[d, "h"] \\
C \arrow[r, "c"'] & NI & E \arrow[l, "e"]
\end{tikzcd}$$

The sets T_B and I_B are exactly the boundaries of f_T and f_I, respectively, in **Sets**. So pushouts (3) and (4) are initial and because pushouts in **PTINets** can be constructed componentwise in **Sets** there are pushouts (6) and (7) in **Sets** leading to unique suitable functions $b_T^*, c_T^*, b_I^*, c_I^* \in \mathcal{M}_{\mathbf{Sets}}$ such that (8) and (9) are pushouts in **Sets**.

$$\begin{tikzcd}
T_B \arrow[r, "b_T^*"'] \arrow[rr, bend left, "b_T"] \arrow[d, "g_T"'] \arrow[dr, phantom, "\textbf{(8)}"] & T_D \arrow[r, "d_T"'] \arrow[d, "h_T"'] \arrow[dr, phantom, "\textbf{(6)}"] & T_L \arrow[d, "f_T"] \\
T_C \arrow[r, "c_T^*"] \arrow[rr, bend right, "c_T"'] & T_E \arrow[r, "e_T"] & NI_T
\end{tikzcd}
\qquad
\begin{tikzcd}
I_B \arrow[r, "b_I^*"'] \arrow[rr, bend left, "b_I"] \arrow[d, "g_I"'] \arrow[dr, phantom, "\textbf{(9)}"] & I_D \arrow[r, "d_I"'] \arrow[d, "h_I"'] \arrow[dr, phantom, "\textbf{(7)}"] & I_L \arrow[d, "f_I"] \\
I_C \arrow[r, "c_I^*"] \arrow[rr, bend right, "c_I"'] & I_E \arrow[r, "e_I"] & NI_I
\end{tikzcd}$$

We define a function $c_P^* \colon P_C \to P_E$ with $c_P^*(x) = y$ with $e_P(y) = c_P(x)$.

well-definedness of c_P^*: For the well-definedness of c_P^* we have to show that for every $x \in P_C$ there is a unique $y \in P_E$ with $e_P(y) = c_P(x)$. Let $x \in P_C$. We use in the following the fact that from pushout (5) in **PTINets** follows that (10) is a pushout in **Sets**.

$$\begin{tikzcd}
P_D \arrow[r, "d_P"] \arrow[d, "h_P"'] \arrow[dr, phantom, "\textbf{(10)}"] & P_L \arrow[d, "f_P"'] \\
P_C \arrow[r, "e_P"'] & P_{NI}
\end{tikzcd}$$

Case 1 $x \notin f_P(P_L)$: Because (10) is pushout in **Sets** the functions f_P and e_P are jointly surjective. So $x \notin f_P(P_L)$ implies $x \in e_P(P_E)$ and hence there exists $y \in P_E$ with $e_P(y) = x$.

Case 2 $x \in f_P(b_P(P_B))$: Then there exists $z \in P_B$ with $f_P(b_P(z)) = x$.

Case 2.1 $z \in IP_P$: Then from the fact that (10) is pushout in **Sets** together with Fact A.2.13 follows that d_P and f_P satisfy the gluing condition which means that $z \in d_P(P_D)$, i. e., there exists $z' \in P_D$ with $d_P(z') = z$. Let $y = h_P(z')$. Then we have $e_P(y) = e_P(h_P(z')) = f_P(d_P(z')) = f_P(z) = x$ which means that y is a suitable element.

Case 2.2 $z \in DP_T$: This means that there is $t \in T_G \setminus f_T(T_L)$ with $f_P(z) \in ENV_P(t)$. Because (6) is pushout in **Sets** the functions f_T and e_T are jointly surjective which by the fact that $t \notin f_T(T_L)$ implies that $t \in e_T(T_E)$, i. e., there exists $t' \in T_E$ with $e_T(t') = t$.
Then there is $y \in P_E$ with $y \in ENV_P(t')$ and $e_P(y) = f_P(z) = x$ because P/T morphisms preserve pre and post conditions.

Case 2.3 $z \in DP_I$: Then there exists $i \in I_{NI} \setminus f_I(I_L)$ with $f_P(z) = m_{NI}(i)$. Because (7) is pushout in **Sets** the functions f_I and e_I are jointly surjective which due to the fact that $i \notin f_I(I_L)$ implies that there is $i' \in I_E$ with $e_I(i') = i$. Hence we have $x = f_P(z) = m_{NI}(i) = m_{NI}(e_I(i')) = e_P(m_E(i'))$ which means that $y = m_E(i')$ is a suitable place.

Case 2.4 $z \in P_{IP_T}$: This means that there is $t \in IP_T$ with $z \in ENV_P(t)$. By Fact A.2.13 there is $t \in d_T(T_D)$ because T_E is a pushout complement of d_T and f_T. So there is $t' \in T_D$ with $t = d_T(t')$. Then we have

$$\begin{aligned} e_P^{\oplus}(pre_E(h_T(t'))) &= pre_G(e_T(h_T(t'))) = pre_G(f_T(d_T(t'))) \\ &= pre_G(f_T(t)) = f_P^{\oplus}(pre_L(t)) \end{aligned}$$

and analogously $e_P^{\oplus}(post_E(h_T(t'))) = f_P^{\oplus}(post_L(t))$ which means that there is $y \in ENV_P(h_T(t')) \subseteq P_E$ with $e_P(y) = f_P(z) = x$.

Case 2.5 $z \in P_{IP_I}$: Then there exists $i \in IP_I$ with $z = m_L(i)$ which by Fact A.2.13 implies that there is $i' \in I_D$ with $d_I(i') = i$ because I_E is a pushout complement of d_I and f_I. Then we have

$$\begin{aligned} e_P(m_E(h_I(i'))) &= e_P(h_P(m_D(i'))) = f_P(d_P(m_D(i'))) \\ &= f_P(m_L(d_I(i'))) = f_P(m_L(i)) = f_P(z) = x \end{aligned}$$

and hence $y = m_E(h_I(i'))$ is a suitable place.

So for every $x \in P_C$ there is a suitable $y \in P_E$ with $e_P(y) = c_P(x)$. Let us assume that y is not unique, i. e.there is $y' \in P_E$ with $e_P(y') = c_P(x)$. Because $d \in \mathcal{M}$ and pushouts preserve $\mathcal{M}$-morphisms we also have that $e \in \mathcal{M}$ which means that e_P is injective implying that $y = y'$. Hence c_P is well-defined.

morphism c^*: We define a **PTINets** morphism $c^* = (c_P^*, c_T^*, c_I^*)\colon C \to E$. In order to show that c^* is a well-defined **PTINets** morphism we have to show that it preserves pre and post domains and markings.
Let $t \in T_C$ and let $t' \in T_E$ with $e_T(t') = c_T(t)$, i. e., $t' = c_T^*(t)$.
Then we have $c_P^{\oplus}(pre_C(t)) = pre_{NI}(c_T(t)) = pre_{NI}(e_T(t')) = e_P^{\oplus}(pre_E(t'))$
which means that for $pre_C(t) = \sum_{i=1}^{n} p_i$ and $pre_E(t') = \sum_{i=1}^{m} p'_i$ there is $n = m$ because c_P and e_P are injective and therefore also $c_P^{\oplus}$ and $e_P^{\oplus}$ are injective.
So we have $\sum_{i=1}^{n} c_P(p_i) = c_P^{\oplus}(pre_C(t)) = e_P^{\oplus}(pre_E(t')) = \sum_{i=1}^{n} e_P(p'_i)$, which by the definition of c_P^* means that

$$c_P^{*\oplus}(pre_C(t)) = c_P^{*\oplus}(\sum_{i=1}^{n} p_i) = \sum_{i=1}^{n} c_P^*(p_i) = \sum_{i=1}^{n} p'_i = pre_E(t') = pre_E(c_T^*(t)) \qquad \square$$

The proof that c^* preserves post domains works analogously.
Let $i \in I_C$ and let $i' \in I_E$ with $e_I(i') = c_I(i)$, i. e., $i' = c_I^*(i)$.

Then we have $c_P(m_C(i)) = m_{NI}(c_I(i)) = m_{NI}(e_I(i')) = e_P(m_E(i'))$, which by definition of c^*_P means that $c^*_P(m_C(i)) = m_E(i') = m_E(c^*_I(i))$ Hence c^* is a well-defined **PTINets** morphism. The fact that $e \circ c^* = c$ follows directly from the definition of c^*_P and the initiality of (3) and (4).

uniqueness of c^*, existence of b^* and pushout: By Lemma A.2.4 the morphism c^* is the unique morphism with $c = e \circ c^*$ and there is a unique morphism $b^* \colon B \to D$ with $b = d \circ b^*$ such that (11) is a pushout in **PTINets**.

$$\begin{tikzcd}
B \arrow[r, "b^*"'] \arrow[d, "g"'] \arrow[rr, bend left, "b"] \arrow[dr, phantom, "\mathbf{(11)}"] & D \arrow[r, "d"'] \arrow[d, "h"'] \arrow[dr, phantom, "\mathbf{(5)}"] & L \arrow[d, "f"] \\
C \arrow[r, "c^*"] \arrow[rr, bend right, "c"'] & E \arrow[r, "e"] & NI
\end{tikzcd}$$

Fact A.2.16 (Characterization of Gluing Condition in PTINets)
Let $l \colon K \to L$ and $f \colon L \to NI$ be morphisms in **PTINets** with $l \in \mathcal{M}$.
The morphisms l and f satisfy the gluing condition in **PTINets** if and only if they satisfy the categorical gluing condition.

Proof. We prove the equivalence of the fact that **PTINets** morphisms l and f satisfy the gluing condition, and the fact that l and f satisfy the categorical gluing condition.

If. Let l and f satisfy the categorical gluing condition, i. e., for initial pushout (1) there is a morphism $b^* \colon B \to K$ with $l \circ b^* = b$.

$$\begin{tikzcd}
B \arrow[r, "b"'] \arrow[d, "g"'] \arrow[rr, bend left, "b^*"] \arrow[dr, phantom, "\mathbf{(1)}"] & L \arrow[d, "f"] & K \arrow[l, "l"] \\
C \arrow[r, "c"'] & NI &
\end{tikzcd}$$

Let $x \in IP \cup DP$. We have to show that $x \in GP = l_P(P_K) \cup l_T(T_K) \cup l_I(I_K)$.

Case 1 $x \in IP_P \cup IP_I \cup DP$: Then there is $x \in P_B$ and $y \in P_K$ with $b^*(x) = y$ and $l(y) = l(b^*(x)) = b(x) = x$ which means that $x \in l_P(P_K) \subseteq GP$.

Case 2 $x \in IP_T$: Then there is $x \in T_B$ and $y \in T_K$ with $b^*(x) = y$ and $l(y) = l(b^*(x)) = b(x) = x$ which means that $x \in l_T(T_K) \subseteq GP$.

Case 3 $x \in IP_I$: Analogously to Case 2.

Hence l and f satisfy the gluing condition in **PTINets**.

Only If. Let l and f satisfy the gluing condition in **PTINets**, i. e., there is

$$IP \cup DP \subseteq GP.$$

This means that there is $IP_P \subseteq l_P(P_K)$, $IP_T \subseteq l_T(T_K)$ and $IP_I \subseteq l_I(I_K)$ implying that l_P, f_P and l_T, f_T and l_I, f_I satisfy the gluing condition in **Sets**.
From Fact A.2.13 follows that there are functions $b_P^*\colon P_B \to P_K$ with $l_P \circ b_P^* = b_P$, $b_T^*\colon T_B \to T_K$ with $l_T \circ b_T^* = b_T$, and $l_I\colon I_B \to I_K$ with $l_I \circ b_I^* = b_I$.
We define a **PTINets** morphism $b^* = (b_P^*, b_T^*, b_I^*)$. For the well-definedness we have to show that b^* preserves pre and post domains as well as markings.
Let $t \in T_B$. Then there is

$$\begin{aligned} l_P^{\oplus}(b_P^{*\oplus}(pre_B(t))) &= (l_P \circ b_P^*)^{\oplus}(pre_B(t)) = b_P^{\oplus}(pre_B(t)) \\ &= pre_L(b_T(t)) = pre_L(l_T(b_T^*(t))) = l_P^{\oplus}(pre_K(b_T^*(t))) \end{aligned}$$

and because $\_^{\oplus}$ preserves monomorphisms in **Sets** because monomorphisms in **Sets** are exactly the coequalizers in **Sets** and $\_^{\oplus}$ is a free functor, the morphism $l_P^{\oplus}$ is a monomorphism implying that $b_P^{*\oplus}(pre_B(t)) = pre_K(b_T^*(t))$.
The proof that b^* preserves post domains works analogously.
Let $i \in I_B$. Then there is

$$l_P(b_P^*(m_B(i))) = b_P(m_B(i)) = m_L(b_I(i)) = m_L(l_I(b_I^*(i))) = l_P(m_K(b_I^*(i)))$$

which by the fact that l_P is a monomorphism implies that $b_P^*(m_B(i)) = m_K(b_I^*(i))$
Hence b^* is a well-defined **PTINets** morphism which means that l and f satisfy the categorical gluing condition. □

A.2.4. Initial Pushouts in AHLINets

In order to construct the initial pushout for a match $f\colon L \to ANI$, we define the *boundary* over the match f, which is the minimal subnet containing all places, transitions, and individual tokens that must not be deleted by the application of a rule with left hand side L such that there exists a pushout complement.

Definition A.2.17 (Boundary in AHLINets)
Given two AHLI nets

$$\begin{aligned} L &= (\Sigma_L, P_L, T_L, pre_L, post_L, cond_L, type_L, A_L, I_L, m_L), \\ ANI &= (\Sigma_{ANI}, P_{ANI}, T_{ANI}, pre_{ANI}, post_{ANI}, cond_{ANI}, type_{ANI}, A_{ANI}, I_{ANI}, m_{ANI}) \end{aligned}$$

and an AHLI net morphism $f\colon L \to ANI$. The boundary of f is an AHLI net

$$B = (\Sigma_B, P_B, T_B, pre_B, post_B, cond_B, type_B, A_B, I_B, m_B), \text{ with}$$

- $\Sigma_B = \Sigma_L, A_B = A_L, type_B(p) = type_L(p)$
- $P_B = DP_T \cup DP_I \cup IP_P \cup P_{IP_T} \cup P_{IP_I}$
- $P_{IP_T} = \{p \in P_L \mid \exists t \in IP_T\colon p \in ENV_P(t)\}$

- $P_{IP_I} = \{p \in P_L \mid \exists i \in IP_I \colon p = \pi_P(m_L(i))\}$
- $T_B = IP_T, pre_B(t) = pre_L(t), post_B(t) = post_L(t), cond_B(t) = cond_L(t)$
- $I_B = IP_I, m_B(i) = m_L(i)$

together with $b \colon B \to L = (b_\Sigma, b_P, b_T, b_A, b_I)$ where $b_\Sigma = id_{\Sigma_B}$, $b_A = id_{A_B}$, and the remaining parts are inclusions.

Proof (Well-definedness).

$pre_B, post_B \colon T_B \to (T_{\Sigma_B}(X_B) \otimes P_B)^{\oplus}$: Let $t \in T_B$ and let $(term, p) \leq pre_B(t)$. Then there is $(term, p) \leq pre_L(t)$ which means that $term \in T_{\Sigma_L}(X_L)$ and $p \in P_L$. Then from $\Sigma_B = \Sigma_L$ follows that $term \in T_{\Sigma_B}(X_B)$. Furthermore there is $t \in IP_T$ which by the fact that $p \in ENV_P(t)$ means that $p \in P_{IP_T} \subseteq P_B$.
So pre_B is well-defined. The proof for $post_B$ works completely analogously.

$cond_B \colon T_B \to \mathcal{P}_{fin}(Eqns(\Sigma_B))$: Let $t \in T_B$. Then we have

$$cond_B(t) = cond_L(t) \in \mathcal{P}_{fin}(Eqns(\Sigma_L)) = \mathcal{P}_{fin}(Eqns(\Sigma_B))$$

$type_B \colon P_B \to S_B$: Let $p \in P_B$. Then we have $type_B(p) = type_L(p) \in S_L = S_B$.

$m_B \colon I_B \to A_B \otimes P_B$: Let $i \in I_B$ and let $(a, p) = m_B(i)$. Then there is $(a, p) = m_L(i)$ which means that $a \in A_{type_L(p)} = A_{type_B(p)}$. The fact that $p \in P_B$ follows from the fact that $i \in IP_I$ and hence $p \in P_{IP_I} \subseteq P_B$.

inclusion $b \colon B \to L$: We obtain an inclusion morphism $b \colon B \to L$ from the fact that pre_B,$post_B$,$cond_B$,$type_B$, and m_B are restrictions of the respective functions in L. □

The following facts about the gluing condition and initial pushouts hold in all $\mathcal{M}$-adhesive categories $(\mathbf{AHLINets}, \mathcal{M})$ whose morphism class $\mathcal{M}$ of monomorphisms contains at least inclusions with identities for signature and algebra parts (for concrete instantiations see Sect. 4.3). The next fact completes the construction of initial pushouts for matches and shows that the construction of the boundary in Def. A.2.17 complies to the categorical notion of boundaries in Def. A.2.1.

Fact A.2.18 (Initial Pushout in AHLINets)
Given a morphism $f \colon L \to ANI$ in **AHLINets**, the boundary B of f and the AHLI net

$$C = (\Sigma_C, P_C, T_C, pre_C, post_C, cond_C, type_C, A_C, I_C, m_C) \text{ with}$$

- $\Sigma_C = \Sigma_{ANI}, A_C = A_{ANI}, type_C(p) = type_{ANI}(p)$
- $P_C = (P_{ANI} \setminus f_P(P_L)) \cup f_P(b_P(P_B))$
- $T_C = (T_{ANI} \setminus f_T(T_L)) \cup f_T(b_T(T_B))$
- $pre_C(t) = pre_{ANI}(t), post_C(t) = post_{ANI}(t), cond_C(t) = cond_{ANI}(t)$
- $I_C = (I_{ANI} \setminus f_I(I_L)) \cup f_I(b_I(I_B)), m_C(i) = m_{ANI}(i)$

Then diagram (1) is an initial pushout in **AHLINets**, where $g := f|_B$ and c is an inclusion with $c_\Sigma = id_{\Sigma_C}$ and $c_A = id_{A_C}$.

$$\begin{tikzcd}
B \arrow[r, "b"] \arrow[d, "g"'] \arrow[dr, phantom, "(1)"] & L \arrow[d, "f"] \\
C \arrow[r, "c"'] & ANI
\end{tikzcd}$$

Proof. We prove the well-definedness of the AHLI net C and that the construction leads to an initial pushout in the category **AHLINets**.

well-definedness of C:

$pre_C, post_C \colon T_C \to (T_{\Sigma_C}(X_C) \otimes P_C)^\oplus$:
Let $t \in T_C$ and let $(term, p) \leq pre_C(t)$. Then there is

$$term \in T_{\Sigma_{ANI}}(X_{ANI})_{type_{ANI}(p)} = T_{\Sigma_C}(X_C)_{type_C(p)}.$$

It remains to show that $p \in P_C$. Due to the fact that $t \in (T_{ANI} \setminus f_T(T_L)) \cup f_T(b_T(T_B))$ we can distinguish the following cases:

Case 1 $t \notin f_T(T_L)$:

Case 1.1 $\exists p' \in P_L \colon f_P(p') = p$: Then due to the fact that there is $(term, p) \leq pre_{ANI}(t)$ there is $p' \in DP_T \subseteq P_B$ which means that $p = f_P(b_P(p')) \in P_C$.

Case 1.2 $\nexists p' \in P_L \colon f_P(p') = p$: This means that $p \in P_{ANI} \setminus f_P(P_L) \subseteq P_C$.

Case 2 $t \in f_T(b_T(T_B))$: Then there exists $t' \in T_B$ with $f_T(b_T(t')) = t$ and because AHL morphisms preserve pre conditions there is

$$\begin{aligned}
&pre_{ANI}(t) = pre_{ANI}(f_T(b_T(t'))) \\
&= (f_\Sigma^\sharp \otimes f_P)^\oplus(pre_L(b_T(t'))) = (f_\Sigma^\sharp \otimes f_P)^\oplus((b_\Sigma^\sharp \otimes b_P)^\oplus(pre_B(t')))
\end{aligned}$$

By Lemma A.2.5 there is $pre_{ANI}(t) = ((f_\Sigma \circ b_\Sigma)^\sharp \otimes (f_P \circ b_P))^\oplus(pre_B(t'))$ which for $(term, p) \leq pre_{ANI}(t)$ means that $(term, p) \leq ((f_\Sigma \circ b_\Sigma)^\sharp \otimes (f_P \circ b_P))^\oplus(pre_B(t'))$ and hence $p \in f_P(b_P(P_B)) \subseteq P_C$.

The proof for $post_C$ works analogously.

$m_C \colon I_C \to A_C \otimes P_C$: Let $i \in I_C$ and let $(a, p) = m_C(i)$. Then there is $a \in A_{ANI,type_{ANI}(p)} = A_{C,type_C(p)}$. It remains to show that $p \in P_C$. Due to the fact that $i \in (I_{ANI} \setminus f_I(I_L)) \cup f_I(b_I(I_B))$ we can distinguish the following cases:

Case 1 $i \notin f_I(I_L)$:

Case 1.1 $\exists p' \in P_L \colon f_P(p') = p$: This means that $p' \in DP$ and thus $p' \in P_B$ and $p = f_P(b_P(p')) \in P_C$.

Case 1.2 $\nexists p' \in P_L \colon f_P(p) = p$: This means that $p \notin f_P(P_L)$, i. e., $p \in ANI \setminus f_P(P_L)$ and hence $p \in P_C$.

Case 2 $i \in f_I(b_I(I_B))$: Then there exists $j \in I_B$ with $f_I(b_I(j)) = i$ and we have

$$\begin{aligned} m_C(i) &= m_{ANI}(i) = m_{ANI}(f_I(b_I(j))) = (f_A \otimes f_P)(m_L(b_I(j))) \\ &= (f_A \otimes f_P)((b_A \otimes b_P)(m_B(j))) \end{aligned}$$

which means that $p \in f_P(b_P(P_B))$ and hence $p \in P_C$.

well-definedness of c: We obtain an inclusion morphism $c\colon C \to ANI$ from the fact that pre_C, $post_C$, $cond_C$, $type_C$ and m_C are restrictions of the respective functions in ANI. Furthermore there is $c_\Sigma = id_{\Sigma_C}$ and $c_A = id_{A_C}$ which are well-defined signature and algebra morphisms, respectively.

well-definedness of g: For $J = \{P, T, I\}$ we obtain well-defined functions $g_J\colon J_B \to J_C$ because for $j \in J_B$ there is $g_J(j) = f_J(j) = f_J(b_J(j)) \in J_C$. The morphism g preserves pre and post domains, conditions, types and markings because it is a restriction of f which is a well-defined AHLI net morphism. Furthermore there is $g_\Sigma = f_\Sigma$ and $g_A = f_A$.

(1) is pushout: Due to Lemma A.2.10 the diagrams (2)-(4) are pushouts in **Sets**.

$$\begin{tikzcd} P_B \arrow[r, "b_P"] \arrow[d, "g_P"'] \arrow[dr, phantom, "(2)"] & P_L \arrow[d, "f_P"] \\ P_C \arrow[r, "c_P"'] & P_{ANI} \end{tikzcd} \qquad \begin{tikzcd} T_B \arrow[r, "b_T"] \arrow[d, "g_T"'] \arrow[dr, phantom, "(3)"] & T_L \arrow[d, "f_T"] \\ T_C \arrow[r, "c_T"'] & T_{ANI} \end{tikzcd} \qquad \begin{tikzcd} I_B \arrow[r, "b_I"] \arrow[d, "g_I"'] \arrow[dr, phantom, "(4)"] & I_L \arrow[d, "f_I"] \\ I_C \arrow[r, "c_I"'] & I_{ANI} \end{tikzcd}$$

Moreover, diagram (5) is pushout in **Sig** and (6) is pushout in **Algs**.

$$\begin{tikzcd} \Sigma_L \arrow[r, "id_{\Sigma_L}"] \arrow[d, "f_\Sigma"'] \arrow[dr, phantom, "(5)"] & \Sigma_L \arrow[d, "f_\Sigma"] \\ \Sigma_{ANI} \arrow[r, "id_{\Sigma_{ANI}}"'] & \Sigma_{ANI} \end{tikzcd} \qquad \begin{tikzcd} (\Sigma_L, A_L) \arrow[r, "id_{(\Sigma_L, A_L)}"] \arrow[d, "(f_\Sigma, f_A)"'] \arrow[dr, phantom, "(6)"] & (\Sigma_L, A_L) \arrow[d, "(f_\Sigma, f_A)"] \\ (\Sigma_{ANI}, A_{ANI}) \arrow[r, "id_{(\Sigma_{ANI}, A_{ANI})}"'] & (\Sigma_{ANI}, A_{ANI}) \end{tikzcd}$$

The pushouts (2)-(6) imply that (1) is pushout in **AHLINets** because the pushout in **AHLINets** can be constructed componentwise.

In the remaining proof, we show the **initiality of (1)**: Given pushout (7) in **AHLINets** with $d \in \mathcal{M}$.

$$\begin{tikzcd} B \arrow[r, "b"] \arrow[d, "g"'] \arrow[dr, phantom, "(1)"] & L \arrow[d, "f"] \arrow[dr, phantom, "(7)"] & D \arrow[l, "d"'] \arrow[d, "h"] \\ C \arrow[r, "c"'] & ANI & E \arrow[l, "e"] \end{tikzcd}$$

The sets T_B and I_B are exactly the boundaries of f_T and f_I, respectively, in **Sets**. So pushouts (3) and (4) are initial and because pushouts in **AHLINets** can be constructed componentwise in **Sets** there are pushouts (8) and (9) in **Sets** leading to unique suitable functions $b_T^*, c_T^*, b_I^*, c_I^* \in \mathcal{M}_{\mathbf{Sets}}$ such that (10) and (11) are pushouts in **Sets**.

$$\begin{tikzcd}
T_B \arrow[r, "b_T^*"'] \arrow[rr, bend left, "b_T"] \arrow[d, "g_T"'] \arrow[dr, phantom, "(\mathbf{10})"] & T_D \arrow[r, "d_T"'] \arrow[d, "h_T"'] \arrow[dr, phantom, "(\mathbf{8})"] & T_L \arrow[d, "f_T"] \\
T_C \arrow[r, "c_T^*"] \arrow[rr, bend right, "c_T"'] & T_E \arrow[r, "e_T"] & ANI_T
\end{tikzcd}
\qquad
\begin{tikzcd}
I_B \arrow[r, "b_I^*"'] \arrow[rr, bend left, "b_I"] \arrow[d, "g_I"'] \arrow[dr, phantom, "(\mathbf{11})"] & I_D \arrow[r, "d_I"'] \arrow[d, "h_I"'] \arrow[dr, phantom, "(\mathbf{9})"] & I_L \arrow[d, "f_I"] \\
I_C \arrow[r, "c_I^*"] \arrow[rr, bend right, "c_I"'] & I_E \arrow[r, "e_I"] & ANI_I
\end{tikzcd}$$

function c_P^*: We define a function $c_P^*\colon P_C \to P_E$ with $c_P^*(x) = y$ with $e_P(y) = c_P(x)$. For the well-definedness of c_P^* we have to show that for every $x \in P_C$ there is a unique $y \in P_E$ with $e_P(y) = c_P(x)$. Let $x \in P_C$. We need in the following that pushout (7) in **AHLINets** implies pushout (12) in **Sets**.

$$\begin{tikzcd}
P_D \arrow[r, "d_P"] \arrow[d, "h_P"'] \arrow[dr, phantom, "(\mathbf{12})"] & P_L \arrow[d, "f_P"'] \\
P_E \arrow[r, "e_P"'] & P_{ANI}
\end{tikzcd}$$

Case 1 $x \notin f_P(P_L)$: Because (12) is pushout in **Sets** the functions f_P and e_P are jointly surjective. So $x \notin f_P(P_L)$ implies $x \in e_P(P_E)$ and hence there exists $y \in P_E$ with $e_P(y) = x$.

Case 2 $x \in f_P(b_P(P_B))$: Then there exists $z \in P_B$ with $f_P(b_P(z)) = x$.

Case 2.1 $z \in IP_P$: Then from the fact that (12) is pushout in **Sets** together with Fact A.2.13 follows that d and f satisfy the gluing condition which means that $z \in d_P(P_D)$, i. e., there exists $z' \in P_D$ with $d_P(z') = z$. Let $y = h_P(z')$. Then we have $e_P(y) = e_P(h_P(z')) = f_P(d_P(z')) = f_P(z) = x$ which means that y is a suitable element.

Case 2.2 $z \in DP_T$: This means that there is $t \in T_{ANI} \setminus f_T(T_L)$ with $f_P(z) \in ENV_P(t)$. Because (8) is pushout in **Sets** the functions f_T and e_T are jointly surjective which by the fact that $t \notin f_T(T_L)$ implies that $t \in e_T(T_E)$, i. e., there exists $t' \in T_E$ with $e_T(t') = t$.
Then there is $y \in P_E$ with $y \in ENV_P(t')$ and $e_P(y) = f_P(z) = x$ because AHLI net morphisms preserve pre and post conditions.

Case 2.3 $z \in DP_I$: Then there exists $i \in I_{ANI} \setminus f_I(I_L)$ with $f_P(z) = \pi_P(m_{ANI}(i))$. Because (9) is pushout in **Sets** the functions f_I and e_I are jointly surjective which due to the fact that $i \notin f_I(I_L)$ implies that there is $i' \in I_E$ with $e_I(i') = i$. Hence we have

$$\begin{aligned} x &= f_P(z) = \pi_P(m_{ANI}(i)) = \pi_P(m_{ANI}(e_I(i'))) \\ &= \pi_P((e_A \otimes e_P)(m_E(i'))) = e_P(m_E(i')) \end{aligned}$$

which means that $y = m_E(i')$ is a suitable place.

Case 2.4 $z \in P_{IP_T}$: This means that there is $t \in IP_T$ with $z \in ENV_P(t)$. By Fact A.2.13 there is $t \in d_T(T_D)$ because T_E is a pushout complement of d_T and f_T. So there is $t' \in T_D$ with $t = d_T(t')$. Then we have

$$\begin{aligned}(e_\Sigma^\sharp \otimes e_P)^\oplus(pre_E(h_T(t'))) &= pre_{ANI}(e_T(h_T(t'))) = pre_{ANI}(f_T(d_T(t')))\\ &= pre_{ANI}(f_T(t)) = (f_\Sigma^\sharp \otimes f_P)^\oplus(pre_L(t))\end{aligned}$$

and analogously $(e_\Sigma^\sharp \otimes e_P)^\oplus(post_E(h_T(t'))) = (f_\Sigma^\sharp \otimes f_P)^\oplus(post_L(t))$ which means that there is $y \in ENV_P(h_T(t')) \subseteq P_E$ with $e_P(y) = f_P(z) = x$.

Case 2.5 $z \in P_{IP_I}$: Then there exists $i \in IP_I$ with $z = \pi_P(m_L(i))$ which by Fact A.2.13 implies that there is $i' \in I_D$ with $d_I(i') = i$ because I_E is a pushout complement of d_I and f_I. Then we have

$$\begin{aligned}(e_A \otimes e_P)(m_E(h_I(i'))) &= m_{ANI}(e_I(h_I(i')))\\ &= m_{ANI}(f_I(d_I(i'))) = (f_A \otimes f_P)(m_L(i))\end{aligned}$$

which means that there is $y = \pi_P(m_E(h_I(i'))) \subseteq P_E$ with $e_P(y) = f_P(z) = x$.

So for every $x \in P_C$ there is a suitable $y \in P_E$ with $e_P(y) = c_P(x)$. Let us assume that y is not unique, i. e., there is $y' \in P_E$ with $e_P(y') = c_P(x)$. Because $d \in \mathcal{M}$ and pushouts preserve $\mathcal{M}$-morphisms there is also $e \in \mathcal{M}$ which means that e_P is injective implying that $y = y'$. Hence c_P is well-defined.

signature morphism c_Σ^*: Due to the fact that $e \in \mathcal{M}$ the signature morphism e_Σ is an isomorphism. We define $c_\Sigma^* = e_\Sigma^{-1}$. Because $\Sigma_C = \Sigma_{ANI}$ the morphism c_Σ^* is well-defined.

algebra morphism c_A^*: Also the algebra morphism e_A is an isomorphism because $e \in \mathcal{M}$. So we define $c_A^* = e_A^{-1}$, leading to a well-defined algebra morphism because $A_C = A_{ANI}$.

morphism c^*: We define an AHLI net morphism $c^* = (c_\Sigma^*, c_P^*, c_T^*, c_A^*, c_I^*)\colon C \to E$. In order to show that c^* is a well-defined AHLI net morphism we have to show that it preserves pre and post conditions, conditions, types and markings.

types: Let $p \in P_C$ and let $p' \in P_E$ with $c_P(p) = e_P(p')$, i. e., $c_P^*(p) = p'$. Furthermore let $c_\Sigma^* = (c_S^*, c_{OP}^*)$ and $e_\Sigma = (e_S, e_{OP})$. Then we have

$$\begin{aligned}&c_S^*(type_C(p)) = e_S^{-1}(type_C(p)) = e_S^{-1}(type_{ANI}(p)) = e_S^{-1}(type_{ANI}(c_P(p)))\\ &= e_S^{-1}(type_{ANI}(e_P(p'))) = e_S^{-1}(e_S(type_E(p'))) = type_E(p') = type_E(c_P^*(p))\end{aligned}$$

pre and post conditions: Let $t \in T_C$. Due to the definition of c_P^* there is $c_P = e_P \circ c_P^*$. So we have

$$\begin{aligned}&(e_\Sigma^\sharp \otimes e_P)^\oplus((c_\Sigma^{*\sharp} \otimes c_P^*)^\oplus(pre_C(t))) = (e_\Sigma^\sharp \otimes e_P)^\oplus((c_\Sigma^{*\sharp} \otimes c_P^*)^\oplus(pre_C(t)))\\ &\overset{\text{Lemma A.2.5}}{=} ((e_\Sigma \circ c_\Sigma^*)^\sharp \otimes (e_P \circ c_P^*))^\oplus(pre_C(t)) = ((e_\Sigma \circ e_\Sigma^{-1})^\sharp \otimes c_P)^\oplus(pre_C(t))\\ &= ((id_{\Sigma_{ANI}})^\sharp \otimes c_P)^\oplus(pre_C(t)) = (c_\Sigma^\sharp \otimes c_P)^\oplus(pre_C(t))\\ &= pre_{ANI}(c_T(t)) = pre_{ANI}(e_T \circ c_T^*(t)) = (e_\Sigma^\sharp \otimes e_P)^\oplus(pre_E(c_T^*(t)))\end{aligned}$$

Because $e \in \mathcal{M}$ is a monomorphism and $\_ \otimes \_$, $\_^\sharp$ and $\_^\oplus$ preserve monomorphisms, there is also $(e_\Sigma^\sharp \otimes e_P)^\oplus$ a monomorphism. So the above equation implies $(c_\Sigma^{*\sharp} \otimes c_P^*)^\oplus(pre_C(t)) = pre_E(c_T^*(t))$ The proof that c^* preserves post conditions works analogously.

conditions: Let $t \in T_C$ and let $t' \in T_E$ with $e_T(t') = c_T(t)$, i. e., $t' = c_T^*(t)$. Due to the fact that $c_\Sigma = id_{\Sigma_C}$ there is

$$\begin{aligned} cond_C(t) &= \mathcal{P}_{fin}(c_\Sigma^\sharp)(cond_C(t)) = cond_{ANI}(c_T(t)) \\ &= cond_{ANI}(e_T(t')) = \mathcal{P}_{fin}(e_\Sigma^\sharp)(cond_E(t')) \end{aligned}$$

which by the definition of c_Σ^* implies that

$$\begin{aligned} &\mathcal{P}_{fin}(c_\Sigma^{*\sharp})(cond_C(t)) = \mathcal{P}_{fin}(c_\Sigma^{*\sharp})(\mathcal{P}_{fin}(e_\Sigma^\sharp)(cond_E(t'))) \\ &\overset{\text{Lemma } A.2.6}{=} \mathcal{P}_{fin}((c_\Sigma^* \circ e_\Sigma)^\sharp)(cond_E(t')) = \mathcal{P}_{fin}((e_\Sigma^{-1} \circ e_\Sigma)^\sharp)(cond_E(t')) \\ &= \mathcal{P}_{fin}(id_{\Sigma_E}^\sharp)(cond_E(t')) = \mathcal{P}_{fin}(id_{T_{\Sigma_E}(X_E)})(cond_E(t')) \\ &= cond_E(t') = cond_E(c_T^*(t)) \end{aligned}$$

markings: Let $i \in I_C$ and $m_C(i) = (a, p)$. Then we have

$$\begin{aligned} &(e_A \otimes e_P)((c_A^* \otimes c_P^*)(m_C(i))) = (e_A \otimes e_P)((c_A^* \otimes c_P^*)(a, p)) \\ &= (e_A \otimes e_P)(c^*_{A,type_C(p)}(a), c_P^*(p)) = (e_{A,type_E(c_P^*(p))}(c^*_{A,type_C(p)}(a)), e_P(c_P^*(p))) \\ &= (e_{A,c_S^*(type_C(p))}(c^*_{A,type_C(p)}(a)), e_P(c_P^*(p))) \\ &= (V_{c_\Sigma^*}(e_A)_{type_C(p)}(c^*_{A,type_C(p)}(a)), e_P(c_P^*(p))) \\ &= (V_{c_\Sigma^*}(e_A)_{type_C(p)}(e^{-1}_{A,type_C(p)}(a), c_P(p))) = ((V_{c_\Sigma^*}(e_A) \circ e_A^{-1})_{type_C(p)}(a), c_P(p)) \\ &= (a, c_P(p)) = (c_A(a), c_P(p)) = (c_A \otimes c_P)(a, p) = (c_A \otimes c_P)(m_C(i)) \\ &= m_{ANI}(c_I(i)) = m_{ANI}(e_I \circ c_I^*(i)) = (e_A \otimes e_P)(m_E(c_I^*(i))) \qquad \square \end{aligned}$$

and because $e \in \mathcal{M}$ is a monomorphism and $\_ \otimes \_$ preserves monomorphisms, there is also $(e_A \otimes e_P)$ a monomorphism. So the equation above implies $(c_A^* \otimes c_P^*)(m_C(i)) = m_E(c_I^*(i))$. Hence c^* is a well-defined AHLI net morphism. The fact that $e \circ c^* = c$ follows directly from the definitions of c_P^*, c_Σ^* and c_A^* and the initiality of (3) and (4).

uniqueness of c^*, existence of b^* and pushout: By Lemma A.2.4 the morphism c^* is the unique morphism with $c = e \circ c^*$ and there is a unique morphism $b^*: B \to D$ with $b = d \circ b^*$ such that (13) is a pushout in **AHLINets**.

$$\begin{tikzcd}
B \arrow[r, "b^*"'] \arrow[rr, bend left, "b"] \arrow[d, "g"'] \arrow[dr, phantom, "(\mathbf{13})"] & D \arrow[r, "d"'] \arrow[d, "h"] \arrow[dr, phantom, "(\mathbf{7})"] & L \arrow[d, "f"] \\
C \arrow[r, "c^*"] \arrow[rr, bend right, "c"'] & E \arrow[r, "e"] & ANI
\end{tikzcd}$$

Fact A.2.19 (Characterization of Gluing Condition in AHLINets)
Let $l\colon K \to L$ and $f\colon L \to ANI$ be morphisms in **AHLINets** with $l \in \mathcal{M}$.
The morphisms l and f satisfy the gluing condition in **AHLINets** if and only if they satisfy the categorical gluing condition.

Proof. We prove that the fact that AHLI net morphisms f and l satisfy the gluing condition in **AHLINets** is equivalent to the fact that f and l satisfy the categorical gluing condition.

If. Let l and f satisfy the categorical gluing condition, i. e., for initial pushout (1) there is a morphism $b^*\colon B \to K$ with $l \circ b^* = b$.

$$\begin{tikzcd}
B \arrow[r, "b"'] \arrow[rr, bend left, "b^*"] \arrow[d, "g"'] \arrow[dr, phantom, "(1)"] & L \arrow[d, "f"] & K \arrow[l, "l"] \\
C \arrow[r, "c"'] & ANI &
\end{tikzcd}$$

Let $x \in IP \cup DP$. We have to show that $x \in GP = l_P(P_K) \cup l_T(T_K) \cup l_I(I_K)$. The proof works completely analogously to the proof in Fact A.2.16. Hence l and f satisfy the gluing condition in **AHLINets**.

Only If. Let l and f satisfy the gluing condition in **AHLINets**, i. e., there is

$$IP \cup DP \subseteq GP.$$

This means that there is $IP_P \subseteq l_P(P_K)$, $IP_T \subseteq l_T(T_K)$ and $IP_I \subseteq l_I(I_K)$ implying that l_P, f_P and l_T, f_T and l_I, f_I satisfy the gluing condition in **Sets**.
From Fact A.2.13 follows that there are functions $b_P^*\colon P_B \to P_K$ with $l_P \circ b_P^* = b_P$, $b_T^*\colon T_B \to T_K$ with $l_T \circ b_T^* = b_T$, and $l_I\colon I_B \to I_K$ with $l_I \circ b_I^* = b_I$.
We define an AHLI net morphism $b^* = (b_P^*, b_T^*, b_\Sigma^*, b_A^*, b_I^*)$ with $b_\Sigma^* = l_\Sigma^{-1}$ and $b_A^* = l_A^{-1}$. The signature morphism l_Σ^{-1} and algebra morphism l_A^{-1} exist because $l \in \mathcal{M}$ and hence l_Σ and l_A are isomorphisms.
For the well-definedness of b^* it remains to show that b^* preserves pre and post conditions, conditions, types, and markings.

types: Let $p \in P_B$ and let $p' \in P_K$ with $b_P(p) = l_P(p')$, i. e., $b_P^*(p) = p'$.
Then we have

$$\begin{aligned}
&b_S^*(type_C(p)) = b_S^*(type_{ANI}(p)) = b_S^*(type_{ANI}(c_P(p)))\\
&= b_S^*(type_{ANI}(e_P(p'))) = b_S^*(e_S(type_E(p'))) = e_S^{-1}(e_S(type_E(p')))\\
&= type_E(p') = type_E(c_P^*(p))
\end{aligned}$$

pre and post conditions: Let $t \in T_B$. Then there is

$$
\begin{aligned}
&(l_\Sigma^\sharp \otimes l_P)^\oplus((b_\Sigma^{*\sharp} \otimes b_P^*)^\oplus(pre_B(t)))\\
&\overset{\text{Lemma A.2.5}}{=} ((l_\Sigma \circ b_\Sigma^*)^\sharp \otimes (l_P \circ b_P^*))^\oplus(pre_B(t))\\
&= ((l_\Sigma \circ b_\Sigma^*)^\sharp \otimes (l_P \circ b_P^*))^\oplus(pre_B(t))\\
&= ((l_\Sigma \circ l_\Sigma^{-1})^\sharp \otimes (l_P \circ b_P^*))^\oplus(pre_B(t))\\
&= (id_{\Sigma_L}^\sharp \otimes b_P)^\oplus(pre_B(t)) = (b_\Sigma^\sharp \otimes b_P)^\oplus(pre_B(t))\\
&= (pre_{ANI}(b_T(t))) = (pre_{ANI}(l_T \circ b_T^*(t)))\\
&= (l_\Sigma^\sharp \otimes l_P)^\oplus(pre_K(b_T^*(t)))
\end{aligned}
$$

Because $l \in \mathcal{M}$ is a monomorphism and $\_ \otimes \_$, $\_^\sharp$ and $\_^\oplus$ preserve monomorphisms, there is also $(l_\Sigma^\sharp \otimes l_P)^\oplus$ a monomorphism. So the above equation implies $(b_\Sigma^{*\sharp} \otimes b_P^*)^\oplus(pre_B(t)) = pre_K(b_T^*(t))$. The proof for the post conditions works analogously.

conditions: Let $t \in T_B$ and let $t' \in T_K$ with $b_T(t) = l_T(t')$, i. e., $t' = b_T^*(t)$. Then we have

$$
\begin{aligned}
&\mathcal{P}_{fin}(b_\Sigma^{*\sharp})(cond_B(t)) = \mathcal{P}_{fin}(b_\Sigma^{*\sharp})(cond_L(t))\\
&= \mathcal{P}_{fin}(b_\Sigma^{*\sharp})(cond_L(b_T(t))) = \mathcal{P}_{fin}(b_\Sigma^{*\sharp})(cond_L(l_T(t')))\\
&= \mathcal{P}_{fin}(b_\Sigma^{*\sharp})(\mathcal{P}_{fin}(l_\Sigma^\sharp)(cond_K(t')))\\
&\overset{\text{Lemma A.2.6}}{=} \mathcal{P}_{fin}((b_\Sigma^* \circ l_\Sigma)^\sharp)(cond_K(t'))\\
&= \mathcal{P}_{fin}((l_\Sigma^{-1} \circ l_\Sigma)^\sharp)(cond_K(t')) = \mathcal{P}_{fin}(id_{\Sigma_K})^\sharp)(cond_K(t'))\\
&= cond_K(t') = cond_K(b_T^*(t))
\end{aligned}
$$

markings: Let $i \in I_B$ and $m_B(i) = (a, p)$. Then we have

$$
\begin{aligned}
&(l_A \otimes l_P)((b_A^* \otimes b_P^*)(m_B(i))) = (l_A \otimes l_P)((b_A^* \otimes b_P^*)(a, p))\\
&= (l_A \otimes l_P)(b_{A,type_B(p)}^*(a), b_P^*(p))\\
&= (l_{A,type_K(b_P^*(p))}(b_{A,type_B(p)}^*(a)), l_P(b_P^*(p)))\\
&= (l_{A,b_S^*(type_B(p))}(b_{A,type_B(p)}^*(a)), l_P(b_P^*(p)))\\
&= (V_{b_\Sigma^*}(l_A)_{type_B(p)}(b_{A,type_B(p)}^*(a)), l_P(b_P^*(p)))\\
&= (V_{b_\Sigma^*}(l_A)_{type_B(p)}(l_{A,type_B(p)}^{-1}(a), b_P(p)))\\
&= ((V_{b_\Sigma^*}(l_A) \circ l_A^{-1})_{type_B(p)}(a), b_P(p))\\
&= (a, b_P(p)) = (b_A(a), b_P(p)) = (b_A \otimes b_P)(a, p) = (b_A \otimes b_P)(m_B(i))\\
&= m_{ANI}(b_I(i)) = m_{ANI}(l_I \circ b_I^*(i)) = (l_A \otimes l_P)(m_K(b_I^*(i)))
\end{aligned}
$$

and because $l \in \mathcal{M}$ is a monomorphism and $\_ \otimes \_$ preserves monomorphisms, there is also $(l_A \otimes l_P)$ a monomorphism. So the equation above implies $(b_A^* \otimes b_P^*)(m_B(i)) = m_K(b_I^*(i))$.

Hence b^* is a well-defined AHLI net morphism. The required commutativity follows from the commutativity of its components. So l and f satisfy the categorical gluing condition. □

A.3. Lemmas and Proofs

A.3.1. Pushouts and Pullbacks in Categories of Individual Petri Nets

In this section, we first prove a general lemma for individual Petri systems as defined in Def. 4.3.1, which gives conditions on the underlying kind of Petri net and the individual marking functor that lead to componentwise constructions of pushouts and pullbacks in the categories of the corresponding individual Petri systems. By instantiating the individual Petri system construction with concrete marking functors to categories of PTI nets and AHLI nets, we use this lemma to show that pullbacks along certain injective morphisms and all pushouts can be constructed componentwise in the categories **PTINets** and **AHLINets**.

Lemma A.3.1 (Componentwise Pullbacks/Pushouts of Individual Systems) *Given a category* **Nets** *of nets that has pullbacks along a class $\mathcal{M}$ of morphisms and all pushouts, as well as a functor $M\colon$* **Nets** $\to$ **Sets** *assigning a marking set to each net object,*

1. *the category* **ISystems**$(\mathbf{Nets}, M)$ *has pushouts that are constructed componentwise over the nets and the individual sets, and*
2. *if M preserves pullbacks along $\mathcal{M}$ morphisms then pullbacks in* **ISystems**$(\mathbf{Nets}, M)$ *along morphisms in $\mathcal{M} \times Mor_{\mathbf{Sets}}$ are constructed componentwise over the nets and the individual sets.*

Proof. As stated in Theorem 4.3.2, **ISystems**$(\mathbf{Nets}, M)$ is isomorphic to the comma category $ComCat(ID_{\mathbf{Sets}}, M, \{m\})$ consisting of objects $(I, N, op_m\colon I \to M(N))$.

1. From **Nets** and **Sets** having pushouts, $ID_{\mathbf{Sets}}$ preserving pushouts, and item 2 of Fact A.43 (constructions in comma categories) in [EEPT06], it follows that $ComCat(ID_{\mathbf{Sets}}, M, \{m\})$ has pushouts that can be constructed componentwise (over the I and N components).
2. Although item 3 of Fact A.43 (constructions in comma categories) in [EEPT06] would be applicable only if M preserved all pullbacks, the construction for the proof can be done for pullbacks in **Nets** along $\mathcal{M}$-morphisms, which shows that in **ISystems**$(\mathbf{Nets}, M)$ pullbacks along morphisms in $\mathcal{M} \times Mor_{\mathbf{Sets}}$ can be constructed componentwise over the nets and the individual sets. □

A.3.1.1. Proof of Fact 4.1.4 and Fact 4.1.5

Fact (Componentwise Pushouts and Pullbacks in PTINets)
The category **PTINets** has pullbacks along injective morphisms and all pushouts, and these constructions are componentwise over the net and the individual sets.

Proof. **PTNets** is known to be complete and cocomplete. Given the functor $M_{PT}\colon \mathbf{PTNets} \to \mathbf{Sets}$ defined by $M_{PT}(P, T, pre, post) = P$, we have $\mathbf{PTINets} \cong \mathbf{ISystems}(\mathbf{PTNets}, M_{PT})$ (cf. the proof for Theorem 4.3.3). Consider the class $\mathcal{M}$ of injective **PTNets** morphisms. M_{PT} preserves pullbacks along $\mathcal{M}$-morphisms because pullbacks along injective morphisms are constructed componentwise in **PTNets**. With this, the proposition follows directly from Lemma A.3.1. □

A.3.1.2. Proof of Fact 4.2.5 and Fact 4.2.6

Fact (Componentwise Pushouts and Pullbacks in AHLINets)
The category **AHINets** has pullbacks along injective morphisms with isomorphic algebra component and all pushouts, and these constructions are componentwise over the net and the individual sets.

Proof. **AHLNets** is known to be complete and cocomplete. It is easy to see that this holds also for its full subcategory $\mathbf{AHLNets}_{\emptyset}$ of generalized AHL nets with empty sets of specification equations. Given the functor $M_{AHL}\colon \mathbf{AHLNets}_{\emptyset} \to \mathbf{Sets}$ with

$$M_{AHL}(SP, P, T, pre, post, cond, type, A) = A \otimes P,$$

we have $\mathbf{AHLINets} \cong \mathbf{ISystems}(\mathbf{AHLNets}_{\emptyset}, M_{AHL})$ (cf. the proof of Theorem 4.3.4). Consider the class $\mathcal{M}$ of injective $\mathbf{AHLNets}_{\emptyset}$ morphisms with isomorphic algebra components. M_{AHL} preserves pullbacks along $\mathcal{M}$-morphisms because pullbacks along injective morphisms are constructed componentwise in $\mathbf{AHLNets}_{\emptyset}$ and we have only algebra isomorphisms in $\mathcal{M}$. With this, the proposition follows directly from Lemma A.3.1. □

A.3.2. Proof of Theorem 4.1.11

Theorem (Correspondence of PTI Firings and Transformations)

1. *Each firing step* $NI \overset{t,S}{\rightarrowtail} NI'$ *via token selection* $S = (M, m, N, n)$ *corresponds to an induced direct transformation* $NI \xRightarrow{\varrho(t,S),f} NI'$ *via the transition rule* $\varrho(t, S)$*, where the match* $f : L_{\varrho(t,S)} \to NI$ *is an inclusion.*

2. *Each direct transformation* $NI \xRightarrow{\varrho(t,S),f} NI_1$ *via some transition rule* $\varrho(t, S)$ *with* $t \in T_{NI}$*, token selection* $S = (M, m, N, n)$*, and token-injective match* $f : L_{\varrho(t,S)} \to NI$*, implies that the transition* $f_T(t)$ *is enabled in* NI *under some token selection* $\bar{S}$ *with firing step* $NI \overset{f_T(t),\bar{S}}{\rightarrowtail} NI^*$ *such that* $NI^* \cong NI_1$.

Proof. In the following let $NI = (PN, I, m)$, $NI' = (PN', I', m')$, and $NI_i = (PN_i, I_i, m_i)$. *Item 1.* Consider the DPO diagram in Fig. A.1 with inclusions d and d', i. e., $PN = PN_0 = PN_1$. This diagram exists by Theorem 4.1.9 because there are no identification points (f is injective) and all dangling points are gluing points ($l_P = id_{P_t}$, i. e., no places are deleted). Because pushouts in **PTINets** can be constructed componentwise for the net and the token components (see appendix A.3.1.2), we have $I_0 = I \setminus M$ and

$$\begin{tikzcd}
L=(PN_t,M,m_t) \arrow[d, "f"'] \arrow[rd, phantom, "(\mathbf{PO})"] & K=(PN_t,\emptyset,\emptyset) \arrow[l, hook', "l"'] \arrow[r, hook, "r"] \arrow[d] \arrow[rd, phantom, "(\mathbf{PO})"] & R=(PN_t,N,n_t) \arrow[d, "f^*"] \\
NI=(PN,I,m) & NI_0=(PN_0,I_0,m_0) \arrow[l, "d"] \arrow[r, "d'"'] & NI_1=(PN_1,I_1,m_1)
\end{tikzcd}$$

Figure A.1.: DPO transformation diagram in **PTINets** for $\varrho(t,S)$ applied to NI

$$\begin{tikzcd}
M \arrow[d, "f_I"'] \arrow[rd, phantom, "(\mathbf{PO})"] & \emptyset \arrow[l, hook'] \arrow[r, hook] \arrow[d] \arrow[rd, phantom, "(\mathbf{PO})"] & N \arrow[d, "f_I^*"] \arrow[rd, "n_t"] & \\
I \arrow[rd, "m"'] & I_0 \arrow[l, hook', "d_I"] \arrow[r, hook, "d'_I"] \arrow[d, dashed, "m_0=m\circ d_I"] & I_1 \arrow[rd, dashed, "m_1"'] & P_t \arrow[d, "f_P^*=f_P"] \\
 & P_{PN} \arrow[rr, equal, "id"'] & & P_{PN}
\end{tikzcd}$$

Figure A.2.: DPO diagram in **Sets** for the token components in Fig. A.1

$I_1 = I_0 \uplus (N \setminus \emptyset)$ as in the DPO diagram of the **Sets** components in Fig. A.2. By assumption, t is enabled under S, so we have that $(I \setminus M) \cap N = \emptyset$ and therefore $I_1 = (I \setminus M) \cup N$. For m_1 as induced morphism for the pushout object I_1 follows that

$$m_1(x) = \begin{cases} m_0(x) = m(x) \text{ for } x \in I \setminus M \\ n_t(x) = n(x) \text{ for } x \in N \end{cases}$$

hence $I_1 = I', m_1 = m'$ according to Def. 4.1.2 and therefore $NI_1 = NI'$. This proves the existence of the direct transformation $NI \xRightarrow{\varrho(t,S),f} NI'$.

Item 2. Given a direct transformation $NI \xRightarrow{\varrho(t,S),f} NI_1$ as in the DPO diagrams in Fig. A.1 and Fig. A.2, there is also a direct transformation $NI \xRightarrow{\varrho(t,S),f} \overline{NI}$ with $\overline{NI} = (PN, (I \setminus f_I(M)) + N)$ given by the componentwise DPOs in Fig. A.3a by standard category theory and Fig. A.3b by construction of pushout complements and pushouts in **Sets** (see [EEPT06]) where we choose the injection $\overline{d'_I}$ to be an inclusion. Then there is $\overline{NI} \cong NI_1$ by uniqueness of pushouts and pushout complements in **PTINets**.

Then $f_T(t) \in T_{NI}$ is enabled under a token selection $\bar{S} = (\bar{M}, \bar{m}, \bar{N}, \bar{n})$ with $\bar{M} = f_I(M)$, $\bar{m} = m|_{f_I(M)}$, $\bar{N} = \bar{f}_I(N)$ and $\bar{n} = \bar{m}_1|_{\bar{N}}$ if the following five conditions are met:

1. $\bar{M} \subseteq I$,
2. $\bar{n} \colon \bar{N} \to P_{PN}$,
3. $(I \setminus \bar{M}) \cap \bar{N} = \emptyset$,
4. $\sum_{i \in \bar{M}} \bar{m}(i) = pre_{NI}(f_T(t))$
5. $\sum_{i \in \bar{N}} \bar{n}(i) = post_{NI}(f_T(t))$

Items 1 and 2 hold by construction via image and restriction. Item 3 follows from the fact that the coproduct $(I \setminus \bar{M}) + N$ is a disjoint union in **Sets** and $\bar{N} = \bar{f}_I(N)$ is exactly

$$\begin{tikzcd}
PN_t \arrow[d, "{(f_P,f_T)}"'] \arrow[dr, phantom, "(\mathbf{PO})"] & PN_t \arrow[l, "id"'] \arrow[r, "id"] \arrow[d] \arrow[dr, phantom, "(\mathbf{PO})"] & PN_t \arrow[d, "{(\bar{f}_P,\bar{f}_T)}"] \\
PN & PN \arrow[l, "id"] \arrow[r, "id"'] & PN
\end{tikzcd}$$

(a) DPO in **PTNets**

$$\begin{tikzcd}
M \arrow[d, "f_I"'] \arrow[dr, phantom, "(\mathbf{PO})"] & \emptyset \arrow[l, hook'] \arrow[rr, hook] \arrow[d] \arrow[drr, phantom, "(\mathbf{PO})"] & & N \arrow[d, "\bar{f}_I"] \arrow[dr, "n_t"] & \\
I \arrow[drr, "m"'] & I \setminus f_I(M) \arrow[l, hook', "\bar{d}_I"'] \arrow[rr, hook, "\bar{d}'_1"] \arrow[dr, "\bar{m}_0"] & & (I \setminus f_I(M)) + N \arrow[dr, "\bar{m}_1"'] & P_t \arrow[d, "\bar{f}_P=f_P"] \\
& & P_{PN} \arrow[rr, equal, "id"'] & & P_{PN}
\end{tikzcd}$$

(b) DPO in **Sets**

Figure A.3.: Componentwise DPO diagrams in **PTNets** and **Sets**

the part of that set which is not in $I \setminus \bar{M}$. It remains to show items 4 and 5:

$$\begin{aligned}
&\sum_{i \in \bar{M}} \bar{m}(i) = \sum_{i \in f_I(M)} \bar{m}(i) \\
=&\sum_{i \in M} m \circ f_I(i) && (f_I \text{ inj.}, \bar{m} = m|_{f_I(M)}) \\
=&\sum_{i \in M} f_P \circ m_t(i) = f_P^{\oplus} \circ \sum_{i \in M} m_t(i) && (f \ \mathbf{PTINets}\text{-morphism}) \\
=&f_P^{\oplus} \circ pre_{PN_t}(t) && (t \text{ enabled under } S \text{ in } L_{\varrho(t,S)}) \\
=&pre_{NI} \circ f_T(t) && (f \ \mathbf{PTINets}\text{-morphism})
\end{aligned}$$

and analogously,

$$\begin{aligned}
&\sum_{i \in \bar{N}} \bar{n}(i) = \sum_{i \in \bar{f}_I(N)} \bar{n}(i) \\
=&\sum_{i \in N} \bar{m}_1 \circ \bar{f}_I(i) && (\bar{f}_I \text{ inj.}, \bar{n} = \bar{m}_1|_{\bar{f}_I(N)}) \\
=&\sum_{i \in N} f_P \circ n_t(i) = f_P^{\oplus} \circ \sum_{i \in N} n_t(i) && (\bar{f} \ \mathbf{PTINets}\text{-morphism}, \bar{f}_P = f_P) \\
=&f_P^{\oplus} \circ post_{PN_t}(t) && (t \text{ enabled under } S \text{ in } L_{\varrho(t,S)}) \\
=&post_{NI} \circ f_T(t) && (f \ \mathbf{PTINets}\text{-morphism})
\end{aligned}$$

So we have that $f_T(t)$ is enabled under $\bar{S}$ and we obtain a firing step $NI \xrightarrow{f_T(t),\bar{S}} NI^*$ where NI^* has the same net part PN and the follower marking (I^*, m^*) with $I^* =$

$(I \setminus \bar{M}) \cup \bar{N}$ and

$$m^*(x) = \begin{cases} \bar{m}(x) = m(x) = \bar{m}_1(x)|_{I \setminus \bar{M}} & \text{, if } x \in I \setminus \bar{M}; \\ \bar{n}(x) = \bar{m}_1(x)|_{\bar{N}} & \text{, if } x \in \bar{N}. \end{cases}$$

Now, by the fact that $\bar{d}'_I$ is an inclusion we have

$$I^* = (I \setminus \bar{M}) \cup \bar{N} = (I \setminus f_I(M)) \cup \bar{f}_I(N) = (I \setminus f_I(M)) + N$$

and the marking function $m^* \colon I^* \to P_{PN}$ maps the individuals exactly as $\bar{m}_1 \colon (I \setminus f_I(M)) + N \to P_{PN}$. So we have $NI^* = \overline{NI}$ and hence $NI^* \cong NI_1$. □

A.3.3. Proof of Theorem 4.2.12

Theorem (Correspondence of AHLI Firings and Transformations)

1. *Each firing step* $ANI \overset{t,asg,S}{\rightarrowtail} ANI'$ *via token selection* $S = (M, m, N, n)$ *corresponds to an induced direct transformation* $ANI \xRightarrow{\varrho(t,asg,S),f} ANI'$ *via the transition rule* $\varrho(t, asg, S)$*, where the match* $f \colon L \to ANI$ *is an inclusion.*

2. *Each direct transformation* $ANI \xRightarrow{\varrho(t,asg,S),f} ANI_1$*, via some transition rule* $\varrho(t, asg, S)$ *with* $t \in T_{ANI}$*, token selection* $S = (M, m, N, n)$*, and token-injective match* $f : L_{\varrho(t,asg,S)} \to ANI$*, implies a consistent transition assignment* $(f_T(t), \bar{asg})$ *being enabled in ANI under some token selection* $\bar{S}$ *with firing step* $ANI \overset{f_T(t),\bar{asg},\bar{S}}{\rightarrowtail} ANI^*$ *such that* $ANI^* \cong ANI_1$*.*

Proof. In the following let $ANI = (AN, I, m)$, $ANI' = (AN', I', m')$, and $ANI_i = (AN_i, I_i, m_i)$.
Item 1. Consider the DPO diagram in Fig. A.4 with inclusions d and d', i.e., $AN =$

$$\begin{tikzcd}
L = (AN_t, M, m_t) \arrow[d, "f"'] \arrow[dr, phantom, "\textbf{(PO)}"] & K = (AN_t, \emptyset, \emptyset) \arrow[l, hook', "l"'] \arrow[r, hook, "r"] \arrow[d] \arrow[dr, phantom, "\textbf{(PO)}"] & R = (AN_t, N, n_t) \arrow[d, "f^*"] \\
ANI = (AN, I, m) & ANI_0 = (AN_0, I_0, m_0) \arrow[l, "d"] \arrow[r, "d'"'] & ANI_1 = (AN_1, I_1, m_1)
\end{tikzcd}$$

Figure A.4.: DPO transformation diagram in **AHLINets** for $\varrho(t, asg, S)$ applied to ANI

$AN_0 = AN_1$. This diagram exists by Theorem 4.2.10 because there are no identification points (f is injective) and all dangling points are gluing points ($l_P = id_{P_t}$, i. e., no places are deleted). Because pushouts in **AHLINets** can be constructed componentwise for the net and the token components (see appendix A.3.1.2), we have $I_0 = I \setminus M$ and $I_1 = I_0 \uplus (N \setminus \emptyset)$ as in the DPO diagram of the **Sets** components in Fig. A.5. By assumption, (t, asg) is enabled under S, so we have that $(I \setminus M) \cap N = \emptyset$ and therefore $I_1 = (I \setminus M) \cup N$. For m_1 as induced morphism for the pushout object I_1 follows that

$$\begin{tikzcd}
M \arrow[d, "f_I"'] \arrow[dr, phantom, "\textbf{(PO)}"] & \emptyset \arrow[l, hook'] \arrow[r, hook] \arrow[d] \arrow[dr, phantom, "\textbf{(PO)}"] & N \arrow[d, "f_I^*"] \arrow[dr, "n_t"] & \\
I \arrow[dr, "m"'] & I_0 \arrow[l, hook', "d_I"] \arrow[r, hook, "d'_I"'] \arrow[d, dashed, "m_0=m\circ d_I"] & I_1 \arrow[dr, dashed, "m_1"'] & A_t \otimes P_t \arrow[d, "f_A^*\otimes f_P^*=f_A\otimes f_P"] \\
& A_{AN} \otimes P_{AN} \arrow[rr, equal, "id"'] & & A_{AN} \otimes P_{AN}
\end{tikzcd}$$

Figure A.5.: DPO diagram in **Sets** for the token components in Fig. A.4

$$m_1(x) = \begin{cases} m_0(x) = m(x) \text{ for } x \in I \setminus M \\ n_t(x) = n(x) \text{ for } x \in N \end{cases}$$

hence $I_1 = I', m_1 = m'$ according to Def. 4.2.2 and therefore $ANI_1 = ANI'$. This proves the existence of the direct transformation $ANI \xRightarrow{\varrho(t,asg,S),f} ANI'$.

Item 2. Given a direct transformation $ANI \xRightarrow{\varrho(t,asg,S),f} ANI_1$ as in the DPO diagrams in Fig. A.4 and Fig. A.5, there is also a direct transformation $ANI \xRightarrow{\varrho(t,asg,S),f} \overline{ANI}$ with $\overline{ANI} = (AN, (I \setminus f_I(M)) + N)$ given by the componentwise DPOs in Fig. A.6a by standard category theory and Fig. A.6b by construction of pushout complements and pushouts in **Sets** (see [EEPT06]) where we choose the injection $\bar{d}'_I$ to be an inclusion. Then there is $\overline{ANI} \cong ANI_1$ by uniqueness of pushouts and pushout complements in **AHLINets**.

$$\begin{tikzcd}
AN_t \arrow[d, "{(f_P,f_T)}"'] \arrow[dr, phantom, "\textbf{(PO)}"] & AN_t \arrow[l, "id"'] \arrow[r, "id"] \arrow[d] \arrow[dr, phantom, "\textbf{(PO)}"] & AN_t \arrow[d, "{(\bar{f}_P,\bar{f}_T)}"] \\
AN & AN \arrow[l, "id"] \arrow[r, "id"'] & AN
\end{tikzcd}$$

(a) DPO in **AHLNets**

$$\begin{tikzcd}
M \arrow[d, "f_I"'] \arrow[dr, phantom, "\textbf{(PO)}"] & \emptyset \arrow[l, hook'] \arrow[r, hook] \arrow[d] \arrow[dr, phantom, "\textbf{(PO)}"] & N \arrow[d, "\bar{f}_I"] \arrow[dr, "n_t"] & \\
I \arrow[dr, "m"'] & I \setminus f_I(M) \arrow[l, hook', "\bar{d}_I"'] \arrow[r, hook, "\bar{d}'_1"] \arrow[d, "\bar{m}_0"] & (I \setminus f_I(M)) + N \arrow[dr, "\bar{m}_1"'] & A_t \otimes P_t \arrow[d, "\bar{f}_A\otimes\bar{f}_P=f_A\otimes f_P"] \\
& A_{AN} \otimes P_{AN} \arrow[rr, equal, "id"'] & & A_{AN} \otimes P_{AN}
\end{tikzcd}$$

(b) DPO in **Sets**

Figure A.6.: Componentwise DPO diagrams in **AHLNets** and **Sets**

Then $(f_T(t), asg_f) \in CT_{ANI}$ is enabled under a token selection $\bar{S} = (\bar{M}, \bar{m}, \bar{N}, \bar{n})$ with $\bar{M} = f_I(M)$, $\bar{m} = m|_{f_I(M)}$, $\bar{N} = \bar{f}_I(N)$ and $\bar{n} = \bar{m}_1|_{\bar{N}}$ if the following five conditions are met:

1. $\bar{M} \subseteq I$, 2. $\bar{n} \colon \bar{N} \to A_{AN} \otimes P_{AN}$, 3. $(I \setminus \bar{M}) \cap \bar{N} = \emptyset$,

4. $\sum_{i \in \bar{M}} \bar{m}(i) = pre_A(f_T(t), asg_f)$ 5. $\sum_{i \in \bar{N}} \bar{n}(i) = post_A(f_T(t), asg_f)$

Items 1 and 2 hold by construction via image and restriction. Item 3 follows from the fact that the coproduct $(I \setminus \bar{M}) + N$ is a disjoint union in **Sets** and $\bar{N} = \bar{f}_I(N)$ is exactly the part of that set which is not in $I \setminus \bar{M}$. It remains to show items 4 and 5:

$$
\begin{aligned}
&\sum_{i \in \bar{M}} \bar{m}(i) = \sum_{i \in f_I(M)} \bar{m}(i) && \\
=&\sum_{i \in M} m \circ f_I(i) && (f_I \text{ inj.}, \bar{m} = m|_{f_I(M)}) \\
=&\sum_{i \in M} (f_A \otimes f_P) \circ m_t(i) && (f\ \textbf{AHLINets}\text{-morphism}) \\
=&(f_A \otimes f_P)^{\oplus} \sum_{i \in M} m_t(i) && \\
=&(f_A \otimes f_P)^{\oplus} \circ pre_A(t, asg) && ((t, asg) \text{ enabled under} \\
& && S \text{ in } L_{\varrho(t,asg,S)}) \\
=&(f_A \otimes f_P)^{\oplus} \circ (\overline{asg} \otimes id_P)^{\oplus} \circ pre_{AN_t}(t) && (\text{def. } pre_A) \\
=&(\overline{asg_f} \times id_P)^{\oplus} \circ (f_\Sigma^{\#} \otimes f_P)^{\oplus} \circ pre_{AN_t}(t) && (\text{Lemma A.3.2}) \\
=&(\overline{asg_f} \times id_P)^{\oplus} \circ pre_{AN} \circ f_T(t) && (f \text{ AHLI morph.}) \\
=&pre_A(f_T(t), asg_f) && (\text{def. } pre_A)
\end{aligned}
$$

and analogously,

$$
\begin{aligned}
&\sum_{i \in \bar{N}} \bar{n}(i) = \sum_{i \in \bar{f}_I(N)} \bar{n}(i) && \\
=&\sum_{i \in N} \bar{m}_1 \circ \bar{f}_I(i) && (\bar{f}_I \text{ inj.}, \bar{n} = \bar{m}_1|_{\bar{f}_I(N)}) \\
=&\sum_{i \in N} (\bar{f}_A \otimes \bar{f}_P) \circ n_t(i) && (\bar{f}\ \textbf{AHLINets}\text{-morphism}) \\
=&\sum_{i \in N} (f_A \otimes f_P) \circ n_t(i) && (\bar{f}_P = f_P, \bar{f}_A = f_A) \\
=&(f_A \otimes f_P)^{\oplus} \sum_{i \in N} n_t(i) && \\
=&(f_A \otimes f_P)^{\oplus} \circ post_A(t, asg) && ((t, asg) \text{ enabled under} \\
& && S \text{ in } L_{\varrho(t,asg,S)}) \\
=&(f_A \otimes f_P)^{\oplus} \circ (\overline{asg} \otimes id_P)^{\oplus} \circ post_{AN_t}(t) && (\text{def. } post_A) \\
=&(\overline{asg_f} \times id_P)^{\oplus} \circ (f_\Sigma^{\#} \otimes f_P)^{\oplus} \circ post_{AN_t}(t) && (\text{Lemma A.3.2}) \\
=&(\overline{asg_f} \times id_P)^{\oplus} \circ post_{AN} \circ f_T(t) && (f \text{ AHLI morph.}) \\
=&post_A(f_T(t), asg_f) && (\text{def. } post_A)
\end{aligned}
$$

So we have that $(f_T(t), asg_f)$ is enabled under $\bar{S}$ and we obtain a firing step $ANI \xrightarrow{f_T(t),\bar{S},asg_f} ANI^*$ where ANI^* has the same net part AN and the follower marking (I^*, m^*) with $I^* = (I \setminus \bar{M}) \cup \bar{N}$ and

$$m^*(x) = \begin{cases} \bar{m}(x) = m(x) = \bar{m}_1(x)|_{I\setminus\bar{M}} & \text{, if } x \in I \setminus \bar{M}; \\ \bar{n}(x) = \bar{m}_1(x)|_{\bar{N}} & \text{, if } x \in \bar{N}. \end{cases}$$

Now, by the fact that $\bar{d}'_I$ is an inclusion we have

$$I^* = (I \setminus \bar{M}) \cup \bar{N} = (I \setminus f_I(M)) \cup \bar{f}_I(N) = (I \setminus f_I(M)) + N$$

and the marking function $m^*\colon I^* \to A_{AN} \otimes P_{AN}$ maps the individuals exactly as $\bar{m}_1\colon (I \setminus f_I(M)) + N \to A_{AN} \otimes P_{AN}$. So we have $ANI^* = \overline{ANI}$ and hence $ANI^* \cong ANI_1$. □

A.3.4. Lemma of Translated Assignments

Lemma A.3.2 (Translated Assignments)
Given two AHLI nets

$$ANI_i = (\Sigma_i = (S_i, OP_i, X_i), P_i, T_i, pre_i, post_i, cond_i, type_i, A_i, I_i, m_i), i \in \{1, 2\}$$

a transition assignment $(t, asg\colon Var(t) \to A_1) \in CT_{ANI_1}$, and an AHLI net morphism $f = (f_\Sigma, f_P, f_T, f_A, f_I)\colon ANI_1 \to ANI_2$ with f_Σ injective on the variables of $Var(t)$, it holds for all terms $term \in T_{\Sigma_1}(Var(t))$ that

$$\overline{asg_f} \circ f^\#_\Sigma(term) = f_A \circ \overline{asg}(term)$$

where $asg_f = f_A \circ asg \circ f^{-1}_{\Sigma|Var(t)}\colon Var(f_T(t)) \to A_2$ is the translation of asg along f according to Def. 4.2.4.

Proof (by structural induction over all Σ_1-terms over variables of $Var(t)$). [1] First, note that because (f_Σ, f_A) is a generalized algebra homomorphism, we have for all constants c and operations op in OP_1

$$f_A(c_{A_1}) \overset{f_A \text{ homomorph.}}{=} c_{V_{f_\Sigma}(A_2)} \overset{\text{def. } V_{f_\Sigma}}{=} (f_\Sigma(c))_{A_2} \tag{A.1}$$

$$f_A \circ op_{A_1} \overset{f_A \text{ homomorph.}}{=} op_{V_{f_\Sigma}(A_2)} \circ f_A \overset{\text{def. } V_{f_\Sigma}}{=} (f_\Sigma(op))_{A_2} \circ f_A \tag{A.2}$$

Case 1 $t = x \in Var(t)$:
$f_A \circ \overline{asg}(x) = \overline{f_A \circ asg}(x) = \overline{f_A \circ asg \circ f^{-1}_{\Sigma|Var(t)}} \circ f^\#_\Sigma(x)$
because $f_{\Sigma|Var(t)}\colon Var(t) \to Var(f_T(t))$ is surjective due to f being a net morphism and hence f_Σ bijective on variables in $Var(t)$.

Case 2 $t = (c\colon \to s) \in T_{\Sigma_1}$:
$f_A \circ \overline{asg}(c) = f_A(c_{A_1}) \overset{(A.1)}{=} (f_\Sigma(c))_{A_2} = (f^\#_\Sigma(c))_{A_2} = \overline{asg_f} \circ f^\#_\Sigma(c)$
The last equality holds because a constant c would be evaluated by the extension of just any variable assignment to c_{A_2}.

[1] A categorical proof using free constructions can be found in [EP97].

Case 3 $t = op(t_1, \dots, t_n) \in T_{\Sigma_1}(Var(t))$ with $t_1, \dots, t_n$ satisfying the property to be proven:

$$\begin{aligned}
f_A \circ \overline{asg}(op(t_1, \dots, t_n)) &= f_A \circ op_{A_1}(\overline{asg}(t_1), \dots, \overline{asg}(t_n)) \\
&\overset{(A.2)}{=} (f_\Sigma(op))_{V_{f_\Sigma}(A_2)}(f_A \circ \overline{asg}(t_1), \dots, f_A \circ \overline{asg}(t_n)) \\
&\overset{ind.assumpt.}{=} (f_\Sigma(op))_{V_{f_\Sigma}(A_2)}(\overline{asg_f} \circ f_\Sigma^\#(t_1), \dots, \overline{asg_f} \circ f_\Sigma^\#(t_n)) \\
&= \overline{asg_f}\left((f_\Sigma(op))\,(f_\Sigma^\#(t_1), \dots, f_\Sigma^\#(t_n))\right) \\
&= \overline{asg_f} \circ f_\Sigma^\#\,(op(t_1, \dots, t_n))
\end{aligned} \qquad \square$$

A.3.5. Proof of Theorem 5.2.10

Theorem (Structural Satisfaction of Variable Application Conditions) *Given a variable application condition ac over P. For all morphisms $p\colon P \to N$ holds the equivalence*

$$p \vDash_{\mathrm{S}} ac \Leftrightarrow p^S \vDash DShift(p_A, ac)$$

Proof (by structural induction). For $ac = true$ the equivalence holds. For the inductive step over the case $ac = \exists(a, ac')$ with $a\colon P \to C$, we assume as hypothesis that the equivalence holds for ac' and all morphisms $c\colon C \to N$.

"$\Rightarrow$:" Assume that $p \vDash_{\mathrm{S}} ac$, i. e., there exists a morphism $q\colon C \to N$ such that $q^S \in \mathcal{M}$ with $q \circ a = p$ and $q \vDash_{\mathrm{S}} ac'$. Because ac is a VAC, we have $a_A = id_{T_\Sigma(Y)}$ and therefore $q_A = p_A$, $q_A^D = p_A^D$ and $q^D \circ a = a^{\uparrow p_A} \circ p^D$. From this and the decomposition property follows that

$$\begin{aligned}
p^S &= (q \circ a)^S \\
&= q^S \circ \left(a^S\right)^{\uparrow q_A} \\
&= q^S \circ a^{\uparrow p_A}
\end{aligned}$$

$$\begin{tikzcd}
ac \rhd P \arrow[r, "a"] \arrow[d, "p^D"'] & C \lhd ac' \arrow[d, "q^D"] \\
DShift(p_A, ac) \rhd P^{\uparrow p_A} \arrow[r, "a^{\uparrow p_A}"] \arrow[d, "p^S"'] & C^{\uparrow q_A} \lhd DShift(q_A, ac') \arrow[dl, "q^S \in \mathcal{M}"] \\
N &
\end{tikzcd}$$

Because $q \vDash_{\mathrm{S}} ac'$, the induction hypothesis states that $q^S \vDash DShift(q_A, ac')$ and because of $q_A = p_A$ also $q^S \vDash DShift(p_A, ac')$. Hence, $p^S \vDash \exists(a^{\uparrow p_A}, DShift(p_A, ac')) = DShift(p_A, ac)$.

"$\Leftarrow$:" Assume that $p^S \vDash DShift(p_A, ac)$, i. e., there exists a morphism $q\colon C^{\uparrow p_A} \to N$ such that $q \in \mathcal{M}$ with $q \circ a^{\uparrow p_A} = p^S$ and $q \vDash DShift(p_A, ac')$. From isomophic q_A and Lemma A.3.3, we get that $q^S \vDash DShift(q_A \circ p_A, ac')$. Choose $q' := q \circ id_C^{+p_A}\colon C \to N$, from which follows that

$$\begin{aligned} q' \circ a &= q \circ id_C^{+p_A} \circ a \\ &= q \circ a^{\uparrow p_A} \circ p^D \\ &= p^S \circ p^D \\ &= p \end{aligned}$$

$$\begin{tikzcd}
ac \rhd & P \arrow[r, "a"] \arrow[d, "p^D"'] & C \arrow[d, "id_C^{+p_A}"] & \lhd ac' \\
DShift(p_A, ac) \rhd & P^{\uparrow p_A} \arrow[r, "a^{\uparrow p_A}"] \arrow[d, "p^S"'] & C^{\uparrow p_A} \arrow[dl, "q \in \mathcal{M}"] & \lhd DShift(p_A, ac') \\
 & N & &
\end{tikzcd}$$

Because $q \in \mathcal{M}$ and $q^S = q'^S$, we have also $q'^S \in \mathcal{M}$ and with $q'_A = q_A \circ p_A$ that $q'^S \models DShift(q'_A, ac')$. From the latter and the induction hypothesis, we get that $q' \models_{\mathrm{S}} ac'$ and finally $p \models_{\mathrm{S}} \exists(a, ac') = ac$. □

Lemma A.3.3

For all variable application conditions $ac = \exists(a, ac')$ of some variable set Y with $a\colon P \to C$, all algebra Σ-homomorphisms $h\colon T_\Sigma(Y) \to A_1$, and all AHLI net morphisms $p\colon P^{\uparrow h} \to N$ with isomorphic p_A holds the equivalence

$$p \models DShift(h, ac) \Leftrightarrow p^S \models DShift(p_A \circ h, ac)$$

Proof. By definition, $DShift(h, true) = true$ over $P^{\uparrow h}$ and $DShift(p_A \circ h, true) = true$ over $P^{+p_A \circ h}$ and hence $p \models true \Leftrightarrow p^S \models true$, which we use as inductive base. For the inductive step over the case $ac = \exists(a, ac')$, we assume as hypothesis that for ac' over C, all algebra Σ-homomorphisms $h'\colon T_\Sigma(Y) \to A_1$, and all AHLI net morphisms $p'\colon C^{\uparrow h} \to N$ holds the equivalence

$$p' \models DShift(h', ac') \Leftrightarrow p'^S \models DShift(p'_A \circ h', ac')$$

Note that $id_P^{+p_A \circ h} = id_{P+h}^{+p_A} \circ id_P^{+h} = p^D \circ id_P^{+h}$ and that the following diagram commutes:

$$\begin{tikzcd}
ac \rhd & P \arrow[r, "a"] \arrow[d, "id_P^{+h}"'] \arrow[dr, phantom, "\mathbf{(1)}"] & C \arrow[d, "id_C^{+h}"] & \lhd ac' \\
DShift(h, ac) \rhd & P^{\uparrow h} \arrow[r, "a^{\uparrow h}"] \arrow[d, "p^D = id_{P^{\uparrow h}}^{+p_A}"'] \arrow[dr, phantom, "\mathbf{(2)}"] & C^{\uparrow h} \arrow[d, "id_{C^{\uparrow h}}^{+p_A}"] & \lhd DShift(h, ac') \\
DShift(p_A \circ h, ac) \rhd & P^{\uparrow p_A \circ h} \arrow[r, "a^{\uparrow p_A \circ h}"] & C^{\uparrow p_A \circ h} & \lhd DShift(p_A \circ h, ac')
\end{tikzcd}$$

"$\Rightarrow$:" Assume that $p \models DShift(h, ac)$, i. e., there exists a morphism $q\colon C^{\uparrow h} \to N$ such that $q \in \mathcal{M}$ with $q \circ a^{\uparrow h} = p$ and $q \models DShift(h, ac')$. Because of VAC ac and $a_A^{\uparrow h} = id_{A_1}$

and the decomposition property, we have $p_A = q_A$ and that

$$\begin{aligned}
p^S &= (q \circ a^{\uparrow h})^S \\
&= q^S \circ \left(\left(a^{\uparrow h}\right)^S\right)^{\uparrow q_A} && \text{(decomposition)} \\
&= q^S \circ \left(a^{\uparrow h}\right)^{\uparrow q_A} && ((a^{\uparrow h})^S = (a^{\uparrow h})) \\
&= q^S \circ \left(a^{\uparrow h}\right)^{\uparrow p_A} && (q_A = p_A) \\
&= q^S \circ a^{\uparrow p_A \circ h} && \text{(comm. diag.(2))}
\end{aligned}$$

Because of $q \vDash \mathit{DShift}(h, ac')$ and isomorphic $p_A = q_A$, we get from the induction hypothesis that $q^S \vDash \mathit{DShift}(p_A \circ h, ac')$ and moreover $q^S \in \mathcal{M}$. Therefore, we have $p^S \vDash \exists(a^{\uparrow p_A \circ h}, \mathit{DShift}(p_A \circ h, ac')) = \mathit{DShift}(p_A \circ h, ac)$.

"$\Leftarrow$:" Assume that $p^S \vDash \mathit{DShift}(p_A \circ h, ac)$, i. e., there exists a morphism $q\colon C^{\uparrow p_A \circ h} \to N$ such that $q \in \mathcal{M}$ with $q \circ a^{\uparrow p_A \circ h} = p^S$ and $q \vDash \mathit{DShift}(p_A \circ h, ac')$.

Consider the morphism

$$q' = q \circ id^{p_A}_{C^{\uparrow h}} = q^S \circ id^{q_A}_{C^{\uparrow p_A \circ h}} \circ id^{p_A}_{C^{\uparrow h}} = q^S \circ id^{q_A \circ p_A}_{C^{\uparrow h}} \colon C^{\uparrow h} \to N$$

p_A and q_A are isomorphic, therefore also $q_A \circ p_A$ is and we have $q' \in \mathcal{M}$ and $q'^S = q^S$. Because of $q \vDash \mathit{DShift}(p_A \circ h, ac')$, we get from the inductive hypothesis that $q'^S = q^S \vDash \mathit{DShift}(q_A \circ p_A \circ h, ac')$ and also $q' = q^S \circ id^{q_A \circ p_A}_{C^{\uparrow h}} \vDash \mathit{DShift}(h, ac')$.

Finally, we have that

$$\begin{aligned}
q' \circ a^{\uparrow h} &= q \circ id^{p_A}_{C^{\uparrow h}} \circ a^{\uparrow h} \\
&= q \circ a^{\uparrow p_A \circ h} \circ id^{p_A}_{P^{\uparrow h}} && \text{(comm. diag.(2))} \\
&= p^S \circ id^{p_A}_{P^{\uparrow h}} && (q \circ a^{\uparrow p_A \circ h} = p^S) \\
&= p && \text{(factorization)}
\end{aligned}$$

and hence $p \vDash \exists(a^{\uparrow h}, ac') = \mathit{DShift}(h, ac)$. □

A.3.6. Proof of Theorem 6.3.3

Theorem (Compatibility of Parallel and Sequential Permutability)

1. *Given two parallel permutable firing steps as shown in the top of Fig. 6.4*

$$(NI, I_1, m_1) \xleftarrow{t_1, asg_1, S_1} (NI, I_0, m_0) \xrightarrow{t_2, asg_2, S_2'} (NI, I_2, m_2)$$

then there exist two firing steps $(NI, I_1, m_1) \xrightarrow{t_2, asg_2, S_2} (NI, I_3, m_3)$ *and* $(NI, I_2, m_2) \xrightarrow{t_1, asg_1, S_1'} (NI, I_3', m_3')$ *— called* complement *firing steps — such that both the firing sequences* $(NI, I_0, m_0) \xrightarrow{t_1, asg_1, S_1} (NI, I_1, m_1) \xrightarrow{t_2, asg_2, S_2} (NI, I_3, m_3)$ *and* $(NI, I_0, m_0) \xrightarrow{t_1, asg_1, S_1'} (NI, I_2, m_2) \xrightarrow{t_2, asg_2, S_2'} (NI, I_3', m_3')$ *are sequentially permutable and* $(NI, I_3, m_3) \approx (NI, I_3', m_3')$.

2. *Given two sequentially permutable firing steps as shown in the left of Fig. 6.4*

$$(NI, I_0, m_0) \overset{t_1, asg_1, S_1}{\rightarrowtail} (NI, I_1, m_1) \overset{t_2, asg_2, S_2}{\rightarrowtail} (NI, I_3, m_3)$$

then there exist a firing sequence $(NI, I_0, m_0) \overset{t_2, asg_2, S_2'}{\rightarrowtail} (NI, I_2, m_2) \overset{t_1, asg_1, S_1'}{\rightarrowtail} (NI, I_3', m_3')$ *— called* complement *firing steps — such that* $(NI, I_3, m_3) \approx (NI, I_3', m_3')$ *and that the firing steps* $(NI, I_1, m_1) \overset{t_1, asg_1, S_1}{\leftarrowtail} (NI, I_0, m_0) \overset{t_2, asg_2, S_2'}{\rightarrowtail} (NI, I_2, m_2)$ *are parallel permutable.*

Proof.

1. We first construct a token selection $S_2 = (M_2, m_1, N_2, n_2)$ that enables the complement firing step (t_2, asg_2, S_2) in the lower left of Fig. 6.4.

existence of suitable M_2: Because (t_2, asg_2) is enabled under $S_2' = (M_2', m_0, N_2', n_2')$ in (NI, I_0, m_0), we have

$$(1) \quad pre_A(t_2, asg_2) = \sum_{i \in M_2'} m_0(i) \leq \sum_{i \in I_0} m_0(i)$$

We use the permutability property (Pa) and the selection $S_1 = (M_1, m_0, N_1, n_1)$ to get

$$\begin{aligned} pre_A(t_2, asg_2) &\leq \sum_{i \in I_0} m_0(i) \ominus pre_A(t_1, asg_1) \oplus post_A(t_1, asg_1) && ((1), (\text{Pa})) \\ &= \sum_{i \in I_0} m_0(i) \ominus \sum_{i \in M_1} m_0(i) \oplus \sum_{i \in N_1} n_1(i) && (S_1 \text{ enables } (t_1, asg_1)) \\ &= \sum_{i \in I_1} m_1(i) && (\text{Def. 4.2.2}) \end{aligned}$$

Therefore, there exists a set $M_2 \subseteq I_1$ with $\sum_{i \in M_2} m_1(i) = pre_A(t_2, asg_2)$.

construction of N_2, n_2**:** Select any set N_2 of the same cardinality as N_2' such that $I_1 \setminus M_2 \cap N_2 = \emptyset$. With any bijection $\iota \colon N_2 \to N_2'$, we define $n_2 := n_2' \circ \iota$ and get

$$\sum_{i \in N_2} n_2(i) = \sum_{i \in N_2} n_2' \circ \iota(i) \overset{\iota \text{ bij.}}{=} \sum_{i \in N_2'} n_2'(i) = post_A(t_2, asg_2)$$

because (t_2, asg_2) is enabled under S_2'.

With this, the selection $S_2 = (M_2, m_1, N_2, n_2)$ satisfies all conditions for enabling the firing of (t_2, asg_2) in (NI, I_1, m_1).

sequential permutability: For the firing steps (t_1, asg_1, S_1) and (t_2, asg_2, S_2), the permutability condition (Sa) is the equation (1) above, which follows from the enabling of (t_2, asg_2) under S_2'. The second condition (Sb) is the first inequation that we get by using (1) and (Pa) above.

The construction of the token selection $S_1' = (M_1', m_2, N_1', n_1')$ for the other complement firing step (t_1, asg_1, S_1') and showing sequential permutability with the firing step (t_2, asg_2, S_2') works analogously.

equivalence of firing results: To prove that $(NI, I_3, m_3) \approx (NI, I_3', m_3')$, we show

$$Coll_{AHL}(NI, I_3, m_3) = \left(NI, \sum_{i \in I_3} m_3(i)\right) = \left(NI, \sum_{i \in I_3'} m_3'(i)\right) = Coll_{AHL}(NI, I_3', m_3')$$

with the collective construction $Coll_{AHL}$ for AHLI nets from Def. 4.4.1. It is sufficient to show the equality of the collective markings:

$$\begin{aligned}
&\sum_{i \in I_3} m_3(i) \\
&= \sum_{i \in I_1} m_1(i) \ominus \sum_{i \in M_2} m_1(i) \oplus \sum_{i \in N_2} n_2(i) && \text{(Def. 4.2.2, firing with } S_2\text{)} \\
&= \sum_{i \in I_0} m_0(i) \ominus \sum_{i \in M_1} m_0(i) \oplus \sum_{i \in N_1} n_1(i) \\
&\qquad \ominus \sum_{i \in M_2} m_1(i) \oplus \sum_{i \in N_2} n_2(i) && \text{(Def. 4.2.2, firing with } S_1\text{)} \\
&= \sum_{i \in I_0} m_0(i) \ominus pre_A(t_1, asg_1) \oplus post_A(t_1, asg_1) && (S_1 \text{ enables } (t_1, asg_1), \\
&\qquad \ominus pre_A(t_2, asg_2) \oplus post_A(t_2, asg_2) && S_2 \text{ enables } (t_2, asg_2)) \\
&= \sum_{i \in I_0} m_0(i) \ominus \sum_{i \in M_1'} m_2(i) \oplus \sum_{i \in N_1'} n_1'(i) && (S_1' \text{ enables } (t_1, asg_1), \\
&\qquad \ominus \sum_{i \in M_2'} m_0(i) \oplus \sum_{i \in N_2'} n_2'(i) && S_2' \text{ enables } (t_2, asg_2)) \\
&= \sum_{i \in I_2} m_2(i) \ominus \sum_{i \in M_1'} m_2(i) \oplus \sum_{i \in N_2'} n_2'(i) && \text{(Def. 4.2.2, firing with } S_2'\text{)} \\
&= \sum_{i \in I_3'} m_3'(i) && \text{(Def. 4.2.2, firing with } S_1'\text{)}
\end{aligned}$$

Using Lemma 4.4.4 yields the equivalence of the firing results.

2. We first construct a token selection $S_2' = (M_2', m_0, N_2', n_2')$ that enables the complement firing step (t_2, asg_2, S_2') in the upper right of Fig. 6.4.

existence of suitable M_2': From the permutability property (Sa) for the firing steps $(NI, I_0, m_0) \xrightarrow{t_1, asg_1, S_1} (NI, I_1, m_1) \xrightarrow{t_2, asg_2, S_2} (NI, I_3, m_3)$, we get

$$(1) \quad pre_A(t_2, asg_2) \leq \sum_{i \in I_0} m_0(i)$$

Therefore, there exists a set $M_2' \subseteq I_0$ with $\sum_{i \in M_2'} m_0(i) = pre_A(t_2, asg_2)$.

construction of N_2', n_2': Select any set N_2' of the same cardinality as N_2 such that $I_0 \setminus M_2' \cap N_2' = \emptyset$. With any bijection $\iota \colon N_2' \to N_2$, we define $n_2' := n_2 \circ \iota$ and get

$$\sum_{i \in N_2'} n_2'(i) = \sum_{i \in N_2'} n_2 \circ \iota(i) \overset{\iota \text{ bij.}}{=} \sum_{i \in N_2} n_2(i) = post_A(t_2, asg_2)$$

because (t_2, asg_2) is enabled under S_2.

With this, the selection $S_2' = (M_2', m_0, N_2', n_2')$ satisfies all conditions for enabling the firing of (t_2, asg_2) in (NI, I_0, m_0).

parallel permutability: For the firing steps (t_1, asg_1, S_1) and (t_2, asg_2, S_2'), the permutability condition (Pb) follows directly from adding $pre_A(t_2, asg_2)$ to both sides of the inequation given by the permutability property (Sb). For showing that also the permutability condition (Pa) holds, we use that

$$pre_A(t_2, asg_2) \leq \sum_{i \in I_1} m_1(i) = \sum_{i \in I_0} m_0(i) \ominus pre_A(t_1, asg_1) \oplus post_A(t_1, asg_1)$$

because (t_2, asg_2) is enabled under S_2 in (NI, I_1, m_1), which results as a firing of (t_1, asg_1, S_1) in (NI, I_0, m_0). By adding $pre_A(t_1, asg_1)$ to both (outermost) sides of this inequation, we get (Pa).

equivalence of firing results: The existence and construction of the token selection $S_1' = (M_1', m_2, N_1', n_1')$ for the second complement firing step (t_1, asg_1, S_1') follows from item 1 of this theorem. The proof for $(NI, I_3, m_3) \approx (NI, I_3', m_3')$ is analogous to the one of item 1. □

A.3.7. Proof of Theorem 6.3.8

Theorem (Local Church-Rosser Theorem)

1. *Given a direct transformation and firing step $ANI_1 \xLeftarrow{\varrho_1, o_1} ANI_0 \overset{t,asg,S}{\rightarrowtail} ANI_0'$ that are parallel independent (see the top of Fig. 6.7), then there exist the firing step $ANI_1 \overset{t,asg,S}{\rightarrowtail} ANI_1'$ and a direct transformation $ANI_0' \xRightarrow{\varrho_1, \bar{o}_1} ANI_1'$ — called* complement *firing step and transformation — such that $ANI_0 \xRightarrow{\varrho_1, o_1} ANI_1 \overset{t,asg,S}{\rightarrowtail} ANI_1'$ and $ANI_0 \overset{t,asg,S}{\rightarrowtail} ANI_0' \xRightarrow{\varrho_1, \bar{o}_1} ANI_1'$ both are sequentially independent.*

2a. *Given a direct transformation followed by a firing step $ANI_0 \xRightarrow{\varrho_1, o_1} ANI_1 \overset{t,asg,S}{\rightarrowtail} ANI_1'$ that are sequentially independent (see the left of Fig. 6.7), then there exist the firing step $ANI_0 \overset{t,asg,S}{\rightarrowtail} ANI_0'$ and a direct transformation $ANI_0' \xRightarrow{\varrho_1, \bar{o}_1} ANI_1'$ — called* complement *firing step and transformation — such that $ANI_1 \xLeftarrow{\varrho_1, o_1} ANI_0 \overset{t,asg,S}{\rightarrowtail} ANI_0'$ are parallel independent.*

2b. *Given a firing step followed by a direct transformation $ANI_0 \overset{t,asg,S}{\rightarrowtail} ANI_0' \xRightarrow{\varrho_1, \bar{o}_1} ANI_1'$ that are sequentially independent (see the right of Fig. 6.7), then there exists a direct transformation $ANI_0 \xRightarrow{\varrho_1, o_1} ANI_1$ and a firing step $ANI_1 \overset{t,asg,S}{\rightarrowtail} ANI_1'$ — called* complement *firing step and transformation — such that $ANI_1 \xLeftarrow{\varrho_1, o_1} ANI_0 \overset{t,asg,S}{\rightarrowtail} ANI_0'$ are parallel independent.*

Proof.

1. For the transformation $ANI_1 \xRightarrow{\varrho_1, o_1} ANI_0$, we have the pushouts (1) and (2), and for the firing step $ANI_0 \overset{t,asg,S}{\rightarrowtail} ANI_0'$, we get by item 1 of Theorem 4.2.12 the pushouts (3) and (4) for the application of the firing rule $\varrho(t, asg, S) = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ with the inclusion match $o_t \colon L \to ANI_0$ and the inclusion comatch $o_t' \colon R \to ANI_0'$.

$$\begin{tikzcd}
R_1 \arrow[d, "o'_1"'] \arrow[dr, phantom, "\mathbf{(2)}"] & K_1 \arrow[l, "r_1"'] \arrow[r, "l_1"] \arrow[d] \arrow[dr, phantom, "\mathbf{(1)}"] & L_1 \arrow[dr, "o_1"] & & L \arrow[dl, "o_t"'] \arrow[dr, phantom, "\mathbf{(3)}"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d] \arrow[dr, phantom, "\mathbf{(4)}"] & R \arrow[d, "o'_t"] \\
ANI_1 & D_1 \arrow[l, "g_1"] \arrow[rr, "f_1"'] & & ANI_0 & & D \arrow[ll, "f"] \arrow[r, "g"'] & ANI'_0
\end{tikzcd}$$

From the parallel independence of the transformation and the firing step, we also have the morphisms

- $i_1\colon L_1 \to ANI_0{}'$ with $i_{1,P} = o_{1,P}$, $i_{1,T} = o_{1,T}$, and $i_{1,A} = o_{1,A}$, and
- $i\colon L \to D_1$ with $f_1 \circ i = o_t$.

construction of rule ϱ^*: We construct the extended firing rule $\varrho^* = (L \overset{l^*}{\leftarrow} K^* \overset{r^*}{\rightarrow} R)$ with the interface AHLI net $K^* := (AN_K, I_{K^*}, m_{K^*})$, where

- AN_K is the AHL net component of K,
- $I_{K^*} := \{x \in I_{L_1} \mid o_{1,I}(x) \notin I_D \wedge i_{1,I}(x) \notin I_D\}$ are the individual tokens of L_1 that are not in the images of the inclusions f and g, i. e., the tokens that are deleted and created in the application of $\varrho(t, asg, S)$ via the pushouts (3) and (4),
- $m_{K^*}\colon I_{K^*} \to A_0 \otimes P_0$ is the marking function defined as $m_{K^*}(x) := m_0(o_{1,I}(x))$. Note that $A_0 \otimes P_0$ is the codomain of the marking function m_0 of ANI_0 and also of all marking functions in $\varrho(t, asg, S)$.
- $l^*\colon K^* \to L$ is defined as $l^* := (l_P, l_T, l_A, l^*_i)$ with $l^*_I(x) := o_{1,I}(x)$.
- $r^*\colon K^* \to R$ is defined as $r^* := (r_P, r_T, r_A, r^*_i)$ with $r^*_I(x) := i_{1,I}(x)$.

To show that l^*_I is well-defined, we consider the pushout (3). By Fact 4.2.5, we have a pushout in **Sets** on the token-components of (3). We have that $o_{1,I}(I_{K^*}) \cap f_I(D_I) = \emptyset$. Because f_I and $o_{t,I}$ are jointly surjective there must be a preimage via $o_{t,I}$ in L_I for each element in $o_{1,I}(I_{K^*})$. $o_{t,I}$ is an inclusion so that $o_{1,I}(I_{K^*}) \subseteq L_I$. l^* is a proper AHLI net morphism, because for all $x \in I_{K^*}$ holds

$$\begin{aligned}
&m_L \circ l^*_I(x) \\
=&m_0(o_{1,I}(x)) && \text{(defs. } \varrho(t, asg, S), l^*_I) \\
=&m_{K^*}(x) && \text{(def. } m_{K^*}(x)) \\
=&(l^*_A \otimes l^*_P) \circ m_{K^*}(x) && \text{(def. } \varrho(t, asg, S)\colon l^*_A = l_A = id_{A_0}, l^*_P = l_P = id_{P_0})
\end{aligned}$$

The proof for the well-definedness of r^* works analogously for the elements in pushout (4) and i_1.

token-firing along rule ϱ^*: We construct the pushout (5), where k is the obvious inclusion of K into K^* with $l^* \circ k = l$, because $I_K = \emptyset$. Because of the empty token set I_K, we also have that I_{D^*} is the coproduct of I_{K^*} and I_D with the inclusions u_I and v_I such that $m_{D^*} \circ u_I = m_{K^*}$ and $m_{D^*} \circ v_I = m_D$. From pushout (3), we get f^* induced and that (6) is a pushout because (5)+(6) = (3) and (5) is a pushout.

$$\begin{tikzcd}
K \arrow[r, "k"'] \arrow[rr, bend left, "l"] \arrow[d] \arrow[dr, phantom, "(\mathbf{5})"] & K^* \arrow[r, "l^*"'] \arrow[d, "u"] \arrow[dr, phantom, "(\mathbf{6})"] & L \arrow[d, "o_t"] \\
D \arrow[r, "v"] \arrow[rr, bend right, "f"'] & D^* \arrow[r, dashed, "f^*"] & \mathit{ANI}_0
\end{tikzcd}$$

Analogously, we construct the pushout (7). With the double pushout diagram via (6) and (7), we have the transformation $\mathit{ANI}_0 \overset{\varrho^*,o_t}{\Longrightarrow} \mathit{ANI}'_0$. Note that the net component of D^* is the same as of ANI_0 and ANI_1 and that f^* and g^* are inclusions with identities as components for places, transitions, and algebras.

$$\begin{tikzcd}
L \arrow[d, "o_t"'] \arrow[dr, phantom, "(\mathbf{6})"] & K^* \arrow[l, "l^*"'] \arrow[r, "r^*"] \arrow[d, "u"'] \arrow[dr, phantom, "(\mathbf{7})"] & R \arrow[d, "o'_t"] \\
\mathit{ANI}_0 & D^* \arrow[l, "f^*"] \arrow[r, "g^*"'] & \mathit{ANI}'_0
\end{tikzcd}$$

parallel independency of transformations with ϱ^* and ϱ_1: We already have $i\colon L \to D_1$ with $f_1 \circ i = o_t$. To show that $\mathit{ANI}_1 \overset{\varrho_1,o_1}{\Longleftarrow} \mathit{ANI}_0 \overset{\varrho^*,o_t}{\Longrightarrow} \mathit{ANI}'_0$ are parallel independent, we need a morphism $i_1^*\colon L_1 \to D^*$ such that $f^* \circ i_1^* = o_1$.

Consider the morphism $i_1^* = (o_{1,P}, o_{1,T}, l_{1,A}, i^*_{1,I})$ with $i^*_{1,I}\colon I_{L_1} \to I_{D^*}$ defined as $i^*_{1,I}(x) := \begin{cases} u_I(x) & \text{if } x \in I_{K^*} \\ v_I(o_{1,I}(x)) & \text{if } x \in I_{L_1} \setminus I_{K^*} \end{cases}$, which is well-defined because for all $x \in I_{L_1} \setminus I_{K^*}$ holds that $o_{1,I}(x) \in I_D$. i_1^* is a proper AHLI net morphism because the components of o_1 satisfy the structural conditions and furthermore

Case 1 $x \in I_{K^*}$: $m_{D^*} \circ i^*_{1,I}(x) = m_{D^*} \circ u_I(x) = m_{K^*}(x) = m_0(o_{1,I}(x))$

Case 2 $x \in I_{L_1} \setminus I_{K^*}$:
$m_{D^*} \circ i^*_{1,I}(x) = m_{D^*} \circ v_I(o_{1,I}(x)) = m_D(o_{1,I}(x)) = m_0 \circ f_I(o_{1,I}(x)) = m_0(o_{1,I}(x))$

and for both cases $m_0(o_{1,I}(x)) = (o_{1,A} \otimes o_{1,P}) \circ m_{L_1}(x) = (i_{1,A} \otimes i_{1,P}) \circ m_{L_1}(x)$.

Because the components for places, transitions, and algebra of f^* are identities, it is obvious that $f^* \circ i_1^*$ is equal to o_1 on these components. For the token components hold that

Case 1 $x \in I_{K^*}$: $f_I^* \circ i^*_{1,I}(x) = f_I^*(u_I(x)) \overset{PO(6)}{=} o_t(o_{1,I}(x)) \overset{o_t \text{ incl.}}{=} o_{1,I}(x)$

Case 2 $x \in I_{L_1} \setminus I_{K^*}$: $f_I^* \circ i^*_{1,I}(x) = f_I^*(v_I(o_{1,I}(x))) = f(o_{1,I}(x)) \overset{f \text{ incl.}}{=} o_{1,I}(x)$

construction of complement transformations with ϱ^* and ϱ_1: By the local Church-Rosser Theorem 5.12 in [EEPT06], we get for the parallel independent transformations $\mathit{ANI}_1 \overset{\varrho_1,o_1}{\Longleftarrow} \mathit{ANI}_0 \overset{\varrho^*,o_t}{\Longrightarrow} \mathit{ANI}'_0$ the complement transformations $\mathit{ANI}_1 \overset{\varrho^*,\overline{o}_t}{\Longrightarrow} \mathit{ANI}'_1 \overset{\varrho_1,\overline{o}_1}{\Longleftarrow} \mathit{ANI}'_0$ with the matches $\overline{o}_t = g_1 \circ i$ and $\overline{o}_1 = g^* \circ i_1^*$ along the pushouts (8),(9),(10), and (11).

$$\begin{tikzcd}
L \arrow[d, "\overline{o}_t"'] \arrow[dr, phantom, "(\mathbf{8})"] & K^* \arrow[l, "l^*"'] \arrow[r, "r^*"] \arrow[d] \arrow[dr, phantom, "(\mathbf{9})"] & R \arrow[dr, "\overline{o}'_t"] & & R_1 \arrow[dl, "\overline{o}'_1"'] & K_1 \arrow[l, "r"'] \arrow[r, "l"] \arrow[d] \arrow[dl, phantom, "(\mathbf{11})"] \arrow[dr, phantom, "(\mathbf{10})"] & L_1 \arrow[d, "\overline{o}_1"] \\
ANI_1 & \overline{D}^* \arrow[l, "\overline{f}^*"] \arrow[rr, "\overline{g}^*"'] & & ANI'_1 & & \overline{D}_1 \arrow[ll, "\overline{g}_1"] \arrow[r, "\overline{f}_1"'] & ANI'_0
\end{tikzcd}$$

Moreover, $ANI_0 \overset{\varrho_1,o_1}{\Longrightarrow} ANI_1 \overset{\varrho^*,\overline{o}_t}{\Longrightarrow} ANI'_1$ and $ANI_0 \overset{\varrho^*,o_t}{\Longrightarrow} ANI'_0 \overset{\varrho_1,\overline{o}_1}{\Longrightarrow} ANI'_1$ are both sequentially independent so that there exist morphisms

- $\overline{i}^*\colon L \to D_1$ with $g_1 \circ \overline{i}^* = \overline{o}_t$ and $\overline{i}_1\colon R_1 \to \overline{D}^*$ with $\overline{f}^* \circ \overline{i}_1 = o'_t$
- $\overline{j}_1\colon L_1 \to D$ with $g \circ \overline{j}_1 = \overline{o}_1$ and $\overline{j}^*\colon R \to \overline{D}_1$ with $\overline{f}_1 \circ \overline{j}^* = o'_t$

construction of complement transformation with $\varrho(t, asg, S)$: We first show that u and v satisfy the gluing condition (see Def. 4.2.9). There are no identification points because u is injective. Furthermore, we have for the sets DP and GP of dangling points that $DP \subseteq P_{K^*} = k_P(P_K) \subseteq GP$. Therefore, there exists a unique pushout complement in the pushout (12). With this, we get the double pushout diagram with the composed pushouts (6+12) and (12+7) and the transformation $ANI_1 \overset{\varrho(t,asg,S),\overline{o}_t}{\Longrightarrow} ANI'_1$.

$$\begin{tikzcd}
K^* \arrow[d, "u"'] \arrow[dr, phantom, "(\mathbf{12})"] & K \arrow[l, "k"'] \arrow[d] \\
\overline{D}^* & \overline{D} \arrow[l]
\end{tikzcd}
\qquad
\begin{tikzcd}
L \arrow[d, "\overline{o}_t"'] \arrow[dr, phantom, "(\mathbf{6{+}12})"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d] \arrow[dr, phantom, "(\mathbf{12{+}7})"] & R \arrow[d, "\overline{o}'_t"] \\
ANI_1 & \overline{D} \arrow[l, "\overline{f}"] \arrow[r, "\overline{g}"'] & ANI'_1
\end{tikzcd}$$

From $f_1 \circ i = o_t$ and f_1 and o_t being inclusions, we have that i is an inclusion. Because g_1 is an inclusion, also the match $\overline{o}_t = g_1 \circ i$ is one. By item 2 of Theorem 4.2.12 (and the construction of the token selection in its proof), we get the firing step $ANI_1 \overset{t,asg,S}{\rightarrowtail} ANI^*$ with $ANI^* \cong ANI'_1$. Up to an isomorphism between the tokens of ANI^* and ANI'_1, these nets are identical. Therefore, there also exists a firing step $ANI_1 \overset{t,asg,\overline{S}}{\rightarrowtail} ANI'_1$, where $\overline{S}$ has a different but isomorphic set $N_{\overline{S}}$ of created tokens than S.

sequential independency of complement firing step and transformation: We first show that $ANI_0 \overset{\varrho_1,o_1}{\Longrightarrow} ANI_1 \overset{t,asg,\overline{S}}{\rightarrowtail} ANI'_1$ are sequentially independent. We already have $\overline{i}^*\colon L \to D_1$ with $g_1 \circ \overline{i}^* = \overline{o}_t$. For the morphism $g^* \circ \overline{i}_1\colon R_1 \to ANI'_1$ holds that $(g^* \circ \overline{i}_1)_P = g^*_P \circ \overline{i}_{1,P} \overset{g_P = id_{P_1} = f_P}{=} f^*_P \circ \overline{i}_{1,P} = o'_{t,P}$ and analogously for the transition and algebra components. Hence the transformation and the following complement firing step are sequentially independent.

Similarly, we already have for $ANI_0 \overset{t,asg,S}{\rightarrowtail} ANI'_0 \overset{\varrho_1,\overline{o}_1}{\Longrightarrow} ANI'_1$ the morphism $\overline{j}^*\colon R \to \overline{D}_1$ with $\overline{f}_1 \circ \overline{j}^* = o'_t$. For the morphism $f \circ \overline{j}_1\colon L_1 \to ANI_0$ holds that $(f \circ \overline{j}_1)_P = f_P \circ \overline{j}_{1,P} \overset{f_P = id_{P_0} = g_P}{=} g_P \circ \overline{j}_{1,P} = \overline{o}_{1,P}$ and analogously for the transition and algebra components. Hence the firing step and the following complement transformation are sequentially independent.

2a. From the definitions of sequential and parallel independence, we get immediately that $ANI_0 \xRightarrow{\varrho_1,o_1} ANI_1 \overset{t,asg,S}{\rightarrowtail} ANI'_1$ are sequentially independent iff $ANI_0 \xLeftarrow{\varrho_1^{-1},o'_1} ANI_1 \overset{t,asg,S}{\rightarrowtail} ANI'_1$ are parallel independent, where ϱ_1^{-1} is the inverse rule of ϱ_1 and o'_1 the comatch of the original transformation (analogous to Remark 5.10 in [EEPT06]). Applying item 1 of this theorem yields the complements $ANI_0 \overset{t,asg,S}{\rightarrowtail} ANI'_0 \xLeftarrow{\varrho_1^{-1},\overline{o}'_1} ANI'_1$. Moreover, $ANI_1 \xRightarrow{\varrho_1^{-1},o'_1} ANI_0 \overset{t,asg,S}{\rightarrowtail} ANI'_0$ are sequentially independent and therefore $ANI_1 \xLeftarrow{\varrho_1,o_1} ANI_0 \overset{t,asg,S}{\rightarrowtail} ANI'_0$ are parallel independent.

2b. We cannot directly apply the equivalence of parallel and sequential independence to $ANI_0 \overset{t,asg,S}{\rightarrowtail} ANI'_0 \xRightarrow{\varrho_1,\overline{o}_1} ANI'_1$ as in the proof for item 2a, because in general there is no inverse firing step. Nevertheless, we can construct the inverse firing rule $\varrho(t,asg,S)^{-1}$ and the application of the inverse extended firing rule $ANI_0 \xLeftarrow{\varrho^{*-1},o'_t} ANI'_0$ as in the proof for item 1 that is parallel independent to the transformation $(\varrho_1,\overline{o}_1)$. Finishing the construction as for item 1 yields the complements $ANI_0 \xRightarrow{\varrho_1,o_1} ANI_1 \xLeftarrow{\varrho(t,asg,S)^{-1},\overline{o}'_t} ANI'_1$ with sequentially independent $ANI'_0 \xRightarrow{\varrho^{*-1},o'_t} ANI_0 \xRightarrow{\varrho_1,o_1} ANI_1$ so that $ANI'_0 \xLeftarrow{\varrho^*,o_t} ANI_0 \xRightarrow{\varrho_1,o_1} ANI_1$ are parallel independent. From this we can show that $ANI'_0 \overset{t,asg,S}{\leftarrowtail} ANI_0 \xRightarrow{\varrho_1,o_1} ANI_1$ are parallel independent by using the arguments from the last step of the proof for item 1 and relation of parallel and sequential independence w. r. t. inversion. □

Bibliography

There are two motives for reading a book:
one, that you enjoy it; the other, that you can boast about it.

(Bertrand Russell)

[AAD07] ABDULEZER, Loren ; ABDULEZER, Susan ; DAMMOND, Howard: *Skype For Dummies*. Wiley Publishing, Inc., 2007

[AB09] ASPERTI, Andrea ; BUSI, Nadia: Mobile Petri nets. In: *Mathematical Structures in Computer Science* 19 (2009), Nr. 6, S. 1265–1278

[Agh86] AGHA, Gul: *ACTORS: A Model of Concurrent Computation in Distributed Systems*, Massachusetts Institute of Technology, Cambridge, Diss., 1986. – MIT Press

[AMSPZ01] ARBANOWSKI, Stefan ; MEER, Sven V. D. ; STEGLICH, Stephan ; POPESCU-ZELETIN, Radu: The Human Communication Space: Towards I-centric Communications. In: *Personal and Ubiquitous Computing* Bd. 5, Springer, January 2001. – ISSN 1617–4909, 34–37

[Arb03] ARBANOWSKI, Stefan: *I-Centric Communications*, Technische Universität Berlin, PhD thesis, November 2003

[Bau90] BAUMGARTEN, Bernd: *Petrinetze, Grundlagen und Anwendungen.* Spektrum Akademischer Verlag, 1990. – ISBN 978–3827401755

[BEE$^+$09] BIERMANN, Enrico ; EHRIG, Hartmut ; ERMEL, Claudia ; HOFFMANN, Kathrin ; MODICA, Tony: Modeling Multicasting in Dynamic Communication-based Systems by Reconfigurable High-level Petri Nets. In: *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, IEEE, 2009, S. 47–50

[BEE$^+$10] BIERMANN, E. ; EHRIG, H. ; ERMEL, C. ; GOLAS, U. ; TAENTZER, G.: Parallel Independence of Amalgamated Graph Transformations Applied to Model Transformation. In: ENGELS, G. (Hrsg.) ; LEWERENTZ, C. (Hrsg.) ; SCHÄFER, W. (Hrsg.) ; SCHÜRR, A. (Hrsg.) ; WESTFECHTEL, B. (Hrsg.): *Graph Transformations and Model-Driven Engineering. Essays Dedicated to Manfred Nagl* Bd. 5765. Springer, 2010, S. 121–140

[BEHM07] BIERMANN, Enrico ; ERMEL, Claudia ; HERMANN, Frank ; MODICA, Tony: A Visual Editor for Reconfigurable Object Nets based on the ECLIPSE Graphical Editor Framework. In: JUHÁS, Gabriel (Hrsg.) ; DESEL, Jörg

(Hrsg.): *Proc. 14th Workshop on Algorithms and Tools for Petri Nets (AWPN'07)*. Universität Koblenz-Landau, Germany : GI Special Interest Group on Petri Nets and Related System Models, October 2007

[BFH87] Böhm, P. ; Fonio, H.-R. ; Habel, A.: Amalgamation of graph transformations: a synchronization mechanism. In: *Computer and System Sciences (JCSS)* 34 (1987), S. 377–408

[BGM+03] Boam, Matthew M. ; Gilbert, Jay ; Mathew, Tisson K. ; Rasovky, Karel ; Sistla, Raj: Modular Communications Platform. In: *Intel Technology Journal* 7 (2003), November, Nr. 4, S. 7–16

[BK84] Bergstra, Jan A. ; Klop, Jan W.: Process Algebra for Synchronous Communication. In: *Information and Control* (1984), S. 109–137

[BM08] Biermann, Enrico ; Modica, Tony: Independence Analysis of Firing and Rule-based Net Transformations in Reconfigurable Object Nets. In: Ermel, Claudia (Hrsg.) ; Lara, Juan de (Hrsg.) ; Heckel, Reiko (Hrsg.): *Proc. Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT'08)* Bd. 10, European Association of Software Science and Technology, 2008 (Electronic Communications of the EASST). – ISSN 1863–2122

[BMMS99] Bruni, R. ; Meseguer, J. ; Montanari, U. ; Sassone, V.: Functorial Semantics for Petri Nets under the Individual Token Philosophy. In: *Category Theory and Computer Science, CTCS '99.* Bd. 29, Elsevier, 1999 (ENTCS), 18 pp.

[BS04] Baset, Salman ; Schulzrinne, Henning: An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. In: *Computing Research Repository (CoRR)* abs/cs/0412017 (2004)

[BWGP01] Blackwell, Alan F. ; Whitley, Kirsten N. ; Good, Judith ; Petre, Marian: Cognitive Factors in Programming with Diagrams. In: *Artifical Intelligence Review* 15 (2001), March, S. 95–114. – ISSN 0269–2821

[CM95] Corradini, A. ; Montanari, U.: Specification of Concurrent Systems: From Petri Nets to Graph Grammars. In: Hommel, G. (Hrsg.): *Proc. Workshop on Quality of Communication-Based Systems, Berlin, Germany*, Kluwer Academic Publishers, 1995

[DK06] Desclaux, Fabrice ; Kortchinsky, Kostya: Vanilla Skype. In: *Code-Breakers Journal* RECON2006 Conf. Proc. (2006). – ISSN 1864–7049

[DVLS02] Dulac, Nicolas ; Viguier, Thomas ; Leveson, Nancy G. ; Storey, Margaret-Anne D.: On the Use of Visualization in Formal Requirements Specification. In: *Proc. 10th Anniversary IEEE Joint International Conference on Requirements Engineering (RE)*. Washington, DC, USA : IEEE Computer Society, 2002. – ISBN 0–7695–1465–0, 71–80

[EBO92] Ehrig, Hartmut ; Baldamus, Michael ; Orejas, Fernando: New concepts for amalgamation and extension in the framework of specification logics. In: *Proc. WADT-COMPASS-Workshop Dourdan (1991)* Bd. 655, Springer, 1992 (Lecture Notes in Computer Science), S. 199–221

[EEPT06] Ehrig, Hartmut ; Ehrig, Karsten ; Prange, Ulrike ; Taentzer, Gabriele: *Fundamentals of Algebraic Graph Transformation.* Springer, 2006 (EATCS Monographs in Theoretical Computer Science)

[EFS11] Ermel, Claudia ; Fischer, Winzent ; Shareef, Sarkaft: RONs Revisited: General Approach and Implementation of Reconfigurable Object Nets based on Algebraic High-Level Nets. In: *Electronic Communications of the EASST* 40 (2011)

[EGH10] Ehrig, H. ; Golas, U. ; Hermann, F.: Categorical Frameworks for Graph Transformations and HLR Systems based on the DPO Approach. In: *Bull. EATCS* 102 (2010), S. 111–121

[EGLT11] Ermel, Claudia ; Gall, Jürgen ; Lambers, Leen ; Taentzer, Gabriele: Modeling with Plausibility Checking: Inspecting Favorable and Critical Signs for Consistency between Control Flow and Functional Behavior. In: *Proc. Fundamental Aspects of Software Engineering (FASE'11)* Bd. 6603, Springer, 2011 (Lecture Notes in Computer Science), 156–170

[EH85] Ehrig, H. ; Habel, A.: Graph grammars with application conditions. In: Rozenberg, G. (Hrsg.) ; Salomaa, A. (Hrsg.): *The Book of L.* Springer, 1985, S. 87–100

[EHL+10a] Ehrig, H. ; Habel, A. ; Lambers, L. ; Orejas, F. ; Golas, U.: Local Confluence for Rules with Nested Application Conditions. In: Ehrig, Hartmut (Hrsg.) ; Rensink, Arend (Hrsg.) ; Rozenberg, Grzegorz (Hrsg.) ; Schürr, Andy (Hrsg.): *Proc. of Intern. Conf. on Graph Transformation (ICGT' 10)* Bd. 6372, Springer, 2010 (Lecture Notes in Computer Science). – ISBN 978–3–642–15927–5, 330–345

[EHL10b] Ehrig, Hartmut ; Habel, Annegret ; Lambers, Leen: Parallelism and Concurrency Theorems for Rules with Nested Application Conditions. In: *Electronic Communications of the EASST* 26 (2010), S. 1–24. – ISSN 1863–2122

[EHP+02] Ehrig, Hartmut ; Hoffmann, Kathrin ; Padberg, Julia ; Baldan, Paolo ; Heckel, Reiko: High-Level Net Processes. In: *Formal and Natural Computing* Bd. 2300. Springer, 2002, S. 191 – 219

[EHP+07] Ehrig, H. ; Hoffmann, K. ; Padberg, J. ; Prange, U. ; Ermel, C.: Independence of Net Transformations and Token Firing in Reconfigurable Place/Transition Systems. In: Kleijn, Jetty (Hrsg.) ; Yakovlev, Alex (Hrsg.): *Proc. of 28th International Conference on Application and Theory* *of Petri Nets and Other Models of Concurrency* Bd. 4546, Springer, 2007 (Lecture Notes in Computer Science), S. 104–123

[EHP$^+$08] Ehrig, Hartmut ; Hoffmann, Kathrin ; Padberg, Julia ; Ermel, Claudia ; Prange, Ulrike ; Biermann, Enrico ; Modica, Tony: Petri Net Transformations. In: *Petri Net Theory and Applications*. I-Tech Education and Publication, 2008. – ISBN 978–3–902613–12–7, S. 1–16

[EM85] Ehrig, Hartmut ; Mahr, Bernd: *EATCS Monographs on Theoretical Computer Science*. Bd. 6: *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. Springer, 1985

[EOP06] Ehrig, Harmut ; Orejas, Fernando ; Prange, Ulrike: Categorical Foundations of Distributed Graph Transformation. In: Corradini, Andrea (Hrsg.) ; Ehrig, Hartmut (Hrsg.) ; Montanari, Ugo (Hrsg.) ; Ribeiro, Leila (Hrsg.) ; Rozenberg, Grzegorz (Hrsg.): *Proc. Third International Conference on Graph Transformation (ICGT)* Bd. 4178. Natal, Brazil : Springer, September 2006 (Lecture Notes in Computer Science). – ISBN 3–540–38870–2, 215–229

[EP97] Ermel, Claudia ; Padberg, Julia: Formalization of Variables in Algebraic High-Level Nets / Fakultät IV – Elektrotechnik und Informatik – Technische Universität Berlin. 1997 (97–19). – Technical Report

[EPR94] Ehrig, Hartmut ; Padberg, Julia ; Ribeiro, Leila: Algebraic High-Level Nets: Petri Nets Revisited. In: *Selected papers from the 9th Workshop on Specification of Abstract Data Types Joint with the 4th COMPASS Workshop on Recent Trends in Data Type Specification*. London, UK : Springer, 1994. – ISBN 3–540–57867–6, 188–206

[ERRW03] Ehrig, Hartmut (Hrsg.) ; Reisig, Wolfgang (Hrsg.) ; Rozenberg, Grzegorz (Hrsg.) ; Weber, Herbert (Hrsg.): *Lecture Notes in Computer Science*. Bd. 2472: *Advances in Petri Nets: Petri Net Technology for Communication Based Systems*. Springer, 2003

[Fis10] Fischer, Winzent: *Entwicklung einer Werkzeugumgebung für Algebraische High-Level Netze mit Anwendung auf ein Szenario zur Einsatzsteuerung bei der Berliner Feuerwehr* , Technische Universität Berlin, Diplomarbeit (diploma thesis), 2010

[GEH10] Golas, U. ; Ehrig, H. ; Habel, A.: Multi-Amalgamation in Adhesive Categories. In: Ehrig, Hartmut (Hrsg.) ; Rensink, Arend (Hrsg.) ; Rozenberg, Grzegorz (Hrsg.) ; Schürr, Andy (Hrsg.): *Proceedings of Intern. Conf. on Graph Transformation (ICGT' 10)* Bd. 6372, Springer, 2010 (Lecture Notes in Computer Science). – ISBN 978–3–642–15927–5, 346–361

[Gla05] Glabbeek, Robert J.: The individual and collective token interpretations of Petri nets. In: *Proceedings CONCUR 2005, 16 th International Confer-*

ence on Concurrency Theory. London, UK : Springer-Verlag, 2005. – ISBN 3–540–28309–9, S. 323–337

[Gol11] Golas, Ulrike: *Analysis and Correctness of Algebraic Graph and Model Transformations*, Technische Universität Berlin, Dissertation, 2011. – Vieweg + Teubner – ISBN 978-3-8348-1493-7

[GP95] Glabbeek, R. J. ; Plotkin, G. D.: Configuration Structures. In: *LICS '95: Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science.* Washington, DC, USA : IEEE Computer Society, 1995. – ISBN 0–8186–7050–6, S. 199

[GP96] Green, Thomas R. G. ; Petre, Marian: Usability Analysis of Visual Programming Environments: a 'cognitive dimensions' framework. In: *Journal of Visual Languages and Computing* 7 (1996), June, Nr. 2, S. 131–174

[GP09] Glabbeek, R. J. ; Plotkin, G. D.: Configuration structures, event structures and Petri nets. In: *Theoretical Computer Science* 410 (2009), Nr. 41, S. 4111–4159. – ISSN 0304–3975

[GR83] Goltz, Ursula ; Reisig, Wolfgang: The Non-sequential Behavior of Petri Nets. In: *Information and Control* 57 (1983), Nr. 2/3, S. 125–147

[HB08] Hicheur, Awatef ; Barkaoui, Kamel: Modelling collaborative workflows using recursive ECATNets. In: *NOTERE '08: Proceedings of the 8th international conference on New technologies in distributed systems.* New York, NY, USA : ACM, 2008. – ISBN 978–1–59593–937–1, 1–11

[HEET99] Heckel, Reiko ; Ehrig, Hartmut ; Engels, Gregor ; Taentzer, Gabriele: Classification and Comparison of Modularity Concepts for Graph Transformation Systems. In: Ehrig, H. (Hrsg.) ; Engels, G. (Hrsg.) ; Kreowski, J.-J. (Hrsg.) ; Rozenberg, G. (Hrsg.): *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages and Tools*, World Scientific, 1999, S. 669 – 690

[HEM05] Hoffmann, Kathrin ; Ehrig, Hartmut ; Mossakowski, Till: High-Level Nets with Nets and Rules as Tokens. In: *Proc. of 26th Intern. Conf. on Application and Theory of Petri Nets and other Models of Concurrency* Bd. 3536. Springer, 2005. – ISSN 0302–9743, S. 268–288

[HEOG10] Hermann, Frank ; Ehrig, Hartmut ; Orejas, Fernando ; Golas, Ulrike: Formal Analysis of Functional Behaviour of Model Transformations Based on Triple Graph Grammars. In: Ehrig, Hartmut (Hrsg.) ; Rensink, Arend (Hrsg.) ; Rozenberg, Grzegorz (Hrsg.) ; Schürr, Andy (Hrsg.): *Proceedings of Intern. Conf. on Graph Transformation (ICGT' 10)* Bd. 6372, Springer, 2010 (Lecture Notes in Computer Science). – ISBN 978–3–642–15927–5, 155–170

[Her11] Hermann, Frank: *Analysis and Optimization of Visual Enterprise Models – Based on Graph and Model Transformation*, Technische Universität Berlin, Diss., 2011. – Universitätsverlag der TU Berlin – ISBN 978-3-7983-2321-6

[HHT96] Habel, A. ; Heckel, R. ; Taentzer, G.: Graph Grammars with Negative Application Conditions. In: *Special issue of Fundamenta Informaticae* 26 (1996), Nr. 3,4, S. 287–313

[HK99] Hlavacs, Helmut ; Kotsis, Gabriele: Modeling User Behavior: A Layered Approach. In: *In MASCOTS'99, IEEE Computer Society, Los Alamito*, 1999, S. 218–225

[HM03] Harel, David ; Marelly, Rami: *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine.* Springer, 2003. – ISBN 3–540–00787–3

[HM10] Hoffmann, Kathrin ; Modica, Tony: Formal Modeling of Flexible Processes using Reconfigurable Algebraic High-Level Nets. In: *Electronic Communications of the EASST* 30 (2010), S. 1–25. – ISSN 1863–2122

[Hoa85] Hoare, Charles Antony R.: *Communicating Sequential Processes.* Prentice-Hall, 1985. – ISBN 0–13–153271–5

[HP00] Haddad, Serge ; Poitrenaud, Denis: Modelling and Analyzing Systems with Recursive Petri Nets. In: *Proc. of the Workshop on Discrete Event Systems – Analysis and Control*, Kluwer Academics Publishers, 2000, S. 449–458

[HP05] Habel, A. ; Pennemann, K.-H.: Nested constraints and application conditions for high-level structures. In: Kreowski, H.-J. (Hrsg.) ; Montanari, U. (Hrsg.) ; Orejas, F. (Hrsg.) ; Rozenberg, G. (Hrsg.) ; Taentzer, G. (Hrsg.): *Formal Methods in Software and Systems Modeling* Bd. 3393, Springer, 2005 (Lecture Notes in Computer Science). – ISBN 3–540–24936–2, 294–308

[HP09] Habel, Annegret ; Pennemann, Karl-Heinz: Correctness of high-level transformation systems relative to nested conditions. In: *Mathematical Structures in Computer Science* 19 (2009), S. 1–52

[IOS94] International Organization for Standardization (IOS): *Open Systems Interconnection – Basic Reference Model: Overview: The Basic Model.* http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip. Version: 1994. – Reference number ISO/IEC 7498-1:1994

[IOS96] International Organization for Standardization (IOS): *Open Distributed Processing – Reference model: Foundations.* http:

//standards.iso.org/ittf/PubliclyAvailableStandards/s018836_ISO_IEC_10746-2_1996(E).zip. Version: December 1996. – Reference number ISO/IEC 10746-2:1996(E)

[IOS98] International Organization for Standardization (IOS): *Open Distributed Processing – Reference model: Overview.* http://standards.iso.org/ittf/PubliclyAvailableStandards/c020696_ISO_IEC_10746-1_1998(E).zip. Version: December 1998. – Reference number ISO/IEC 10746-1:1998(E)

[IOS08] International Organization for Standardization (IOS): *Open distributed processing – Use of UML for ODP system specifications.* http://www.rm-odp.net/files/resources/LON-040_UML4ODP_IS/LON-040_UML4ODP_IS.pdf. Version: November 2008. – Reference number ISO/IEC 19793:2008

[Jen92] Jensen, Karl: *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use.* Bd. 1: Basic Concepts. EATCS Monographs in Theoretical Computer Science. Springer, 1992

[Jen94] Jensen, Karl: *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use.* Bd. 2: Analysis Methods. EATCS Monographs in Theoretical Computer Science. Springer, 1994

[Jen97] Jensen, Karl: *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use.* Bd. 3: Practical Use. EATCS Monographs in Theoretical Computer Science. Springer, 1997

[JGJ+00] Jannotti, John ; Gifford, David K. ; Johnson, Kirk L. ; Kaashoek, M. F. ; O'Toole, James W. Jr.: Overcast: reliable multicasting with on overlay network. In: *Proc. 4th Conf. on Operating System Design & Implementation (OSDI)* Bd. 4. Berkeley, CA, USA : USENIX Association, 2000, 14–41

[JR91] Jensen, Karl (Hrsg.) ; Rozenberg, Grzegorz (Hrsg.): *High-Level Petri Nets.* Springer, 1991

[KHM06] Kastenberg, Harmen ; Hermann, Frank ; Modica, Tony: Towards Translating Graph Transformation Systems by Model Transformation. In: *Proc. International Workshop on Graph and Model Transformation (GraMoT'06), Satellite Event of the IEEE Symposium on Visual Languages and Human-Centric Computing* Bd. 4. Brighton, UK, September 2006 (Electronic Communications of the EASST). – ISSN 1863–2122

[KHTR10] Khan, Ajab ; Heckel, Reiko ; Torrini, Paolo ; Ráth, István: Model-Based Stochastic Simulation of P2P VoIP Using Graph Transformation System. In: Al-Begain, Khalid (Hrsg.) ; Fiems, Dieter (Hrsg.) ; Knottenbelt, William J. (Hrsg.) ; Al-Begain, Khalid (Hrsg.) ; Fiems, Dieter

(Hrsg.) ; Knottenbelt, William J. (Hrsg.): *Proc. 17th Int. Conf. Analytical and Stochastic Modeling Techniques and Applications (ASMTA)* Bd. 6148, Springer, 2010 (Lecture Notes in Computer Science). – ISBN 978–3–642–13567–5, 204–217

[KK99] Kreowski, H.-J. ; Kuske, S.: Graph Transformation Units with Interleaving Semantics. In: *Formal Aspects of Computing* 11 (1999), S. 690–723

[KMPP05] Koch, Manuel ; Mancini, L.V. ; Parisi-Presicce, Francesco: Graph-based specification of access control policies. In: *Journal of Computer and System Sciences* 71 (2005), Nr. 1, S. 1 – 33. – ISSN 0022–0000

[KR07] Köhler, Michael ; Rölke, Heiko: Dynamic Transition Refinement. In: *Electronic Notes in Theoretical Computer Science* 175 (2007), Nr. 2, S. 119–134. – ISSN 1571–0661. – Proc. 5th Int. Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA 2006)

[Kre81] Kreowski, H.-J.: A Comparison between Petri Nets and Graph Grammars. In: *5th International Workshop on Graph-Theoretic Concepts in Computer Science* Bd. 100, Springer, 1981 (Lec), S. 1–19

[Kus98] Kuske, Sabine: More about control conditions for transformation units. In: Ehrig, Hartmut (Hrsg.) ; Engels, Gregor (Hrsg.) ; Kreowski, Hans-Jörg (Hrsg.) ; Rozenberg, Grzegorz (Hrsg.): *Proc. Theory and Application of Graph Transformations* Bd. 1764, Springer, 1998 (Lecture Notes in Computer Science), S. 323–337

[LETE04] Lara, Juan de ; Ermel, Claudia ; Taentzer, Gabriele ; Ehrig, Karsten: Parallel Graph Transformation for Model Simulation applied to Timed Transition Petri Nets. In: *Electronic Notes in Theoretical Computer Science* 109 (2004), S. 17–29. – Proc. Workshop on Graph Transformation and Visual Modelling Techniques (GT-VMT 2004)

[LL95] Lee, Jong-Kun ; Lee, Kwang-Hui: Modeling of the Multicast Transport Protocols using Petri Nets. In: *Proc. Conf. on Communications and Networks.*, 1995. – ISBN 0–7803–2579–6, 106–110

[LNW03] Lee, Edward A. ; Neuendorffer, Stephen ; Wirthlin, Michael J.: Actor-oriented Design of Embedded Hardware and Software Systems. In: *Journal of Circuit, Systems, and Computers* 12 (2003), Nr. 3, S. 231–260. – ISSN 0218–1266

[Los03] Loshin, P.: *TCP/IP clearly explained.* 4th edition. Morgan Kaufmann, 2003 (Clearly Explained). – ISBN 978–1558607828

[MBE09a] Modica, Tony ; Biermann, Enrico ; Ermel, Claudia: An ECLIPSE Framework for Rapid Development of Rich-featured GEF Editors based on EMF Models. In: *GI-Workshop on Methodological Development of Modelling Tools*, 2009

[MBE09b] Modica, Tony ; Biermann, Enrico ; Ermel, Claudia: *An ECLIPSE Framework for Rapid Development of Rich-featured GEF Editors based on EMF Models: Long Version*. Internal Report. http://tfs.cs.tu-berlin.de/roneditor/files/MuvitorKit_GI09_long.pdf. Version: 2009

[MEE$^+$10] Modica, Tony ; Ermel, Claudia ; Ehrig, Hartmut ; Hoffmann, Kathrin ; Biermann, Enrico: Modeling Communication Spaces with Higher-Order Petri Nets. In: Lasker, George E. (Hrsg.) ; Pfalzgraf, Jochen (Hrsg.): *Advances in Multiagent Systems, Robotics and Cybernetics: Theory and Practice* Bd. III. Tecumseh, Canada : The International Institute for Advanced Studies in Systems Research and Cybernetics, 2010. – ISBN 978–1–897233–61–0, S. 43–48

[MEE11] Maximova, Maria ; Ehrig, Hartmut ; Ermel, Claudia: Formal Relationship between Petri Net and Graph Transformation Systems based on Functors between $\mathcal{M}$-adhesive Categories. In: *Electronic Communications of the EASST* 40 (2011)

[MGE$^+$10] Modica, Tony ; Gabriel, Karsten ; Ehrig, Hartmut ; Hoffmann, Kathrin ; Shareef, Sarkaft ; Ermel, Claudia ; Golas, Ulrike ; Hermann, Frank ; Biermann, Enrico: Low- and High-Level Petri Nets with Individual Tokens / Fakultät IV – Elektrotechnik und Informatik – Technische Universität Berlin. 2010 (2009/13). – Technical Report

[MGH11] Modica, Tony ; Gabriel, Karsten ; Hoffmann, Kathrin: Formalization of Petri Nets with Individual Tokens as Basis for DPO Net Transformations. In: Ehrig, H. (Hrsg.) ; Ermel, C. (Hrsg.) ; Hoffmann, K. (Hrsg.): *Proc. 4th Workshop on Petri Nets and Graph Transformation (PNGT 2010)* Bd. 40, European Association of Software Science and Technology, 2011

[Mil80] Milner, Robin: *Lecture Notes in Computer Science*. Bd. 92: *A Calculus of Communicating Systems*. Springer, 1980. – ISBN 3–540–10235–3

[Mil99] Milner, Robin: *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, 1999. – ISBN 978–0521658690

[MKL$^+$02] Milojicic, Dejan S. ; Kalogeraki, Vana ; Lukose, Rajan ; Nagaraja, Kiran ; Pruyne, Jim ; Richard, Bruno ; Rollins, Sami ; Xu, Zhichen: Peer-to-Peer Computing / HP Laboratories, Palo Alto. 2002 (HPL-2002-57 (R.1)). – Technical Report

[MM90] Meseguer, José ; Montanari, Ugo: Petri Nets are Monoids. In: *Information and Computation* 88 (1990), Nr. 2, S. 105–155

[Mod10] Modica, Tony: Towards Formal Algebraic Modeling and Analysis of Communication Spaces. In: Haveraaen, Magne (Hrsg.) ; Lenisa, Marina (Hrsg.) ; Power, John (Hrsg.) ; Seisenberger, Monika (Hrsg.): *CALCO*

Young Researchers Workshop (CALCO-jnr 2009) – Selected Papers, Università di Udine – Dipartimento di Matematica e Informatica, 2010 (Technical Report 5-2010), 89–103

[MOM89] Martí-Oliet, Narciso ; Meseguer, José: From Petri Nets to Linear Logic. In: *Proc. Category Theory and Computer Science* Bd. 389, Springer, 1989 (Lecture Notes in Computer Science), 313–340

[MSM97] Makinae, Naoto ; Suzuki, Hikari ; Minamoto, Satoshi: Communication Platform for Service Operations Systems in Advanced Intelligent Network. In: *Proc. Int. Conf. on Communications (ICC)* Bd. 2 IEEE, 1997. – ISBN 0780339258, 872–877

[Nar02] Narayanan, Shrikanth: Towards modeling user behavior in human-machine interactions: Effect of Errors and Emotions. In: *Proc. ISLE Workshop on Multimodal Dialog Tagging*, 2002,

[OMG06] Object Management Group (OMG): *OMG Meta Object Facility (MOF) Core Specification – Version 2.0*. http://www.omg.org/spec/MOF/2.0/PDF/. Version: January 2006. – Document number – formal/06-01-01

[OMG10] Object Management Group (OMG): *OMG Unified Modeling Language (OMG UML), Superstructure – Version 2.3*. http://www.omg.org/spec/UML/2.3/Superstructure/PDF/. Version: May 2010. – Document number – formal/2010-05-05

[PEH07] Padberg, Julia ; Ehrig, Hartmut ; Hoffmann, Kathrin: Formal Modeling and Analysis of flexible Processes in Mobile Ad-Hoc Networks. In: *Bulletin of the European Association for Theoretical Computer Science (EATCS)* 91 (2007), S. 128–132. – ISSN 0252–9742

[PEHP08] Prange, Ulrike ; Ehrig, Hartmut ; Hoffman, Kathrin ; Padberg, Julia: Transformations in Reconfigurable Place/Transition Systems. In: Degano, P. (Hrsg.) ; De Nicola, R. (Hrsg.) ; Meseguer, J. (Hrsg.): *Concurrency, Graphs and Models: Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday* Bd. 5065. Springer, 2008. – ISBN 978–3–540–68676–7, S. 96–113

[PEL08] Prange, Ulrike ; Ehrig, Hartmut ; Lambers, Leen: Construction and Properties of Adhesive and Weak Adhesive High-Level Replacement Categories. In: *Applied Categorical Structures* 16 (2008), Nr. 3, S. 365–388. – ISSN 0927–2852

[PER95] Padberg, Julia ; Ehrig, Hartmut ; Ribeiro, Leila: Algebraic High-Level Net Transformation Systems. In: *Mathematical Structures in Computer Science* 5 (1995), S. 217–256

[Pet62] Petri, Carl A.: *Kommunikation mit Automaten.* Bonn, Schriften des Institutes für Instrumentelle Mathematik, Diss., 1962

[Pra91] Prasad, K.: A calculus of broadcasting systems. In: Abramsky, S. (Hrsg.) ; Maibaum, T. (Hrsg.): *TAPSOFT '91* Bd. 493. Springer Berlin / Heidelberg, 1991. – ISBN 978–3–540–53982–7, S. 338–358

[Pra08] Prange, Ulrike: Towards Algebraic High-Level Systems as Weak Adhesive HLR Categories. In: Ehrig, Hartmut (Hrsg.) ; Pfalzgraf, Jochen (Hrsg.) ; Prange, Ulrike (Hrsg.): *Proc. Workshop of Applied and Computational Category Theory (ACCAT) at ETAPS 2007* Bd. 203 / 6. Amsterdam : Elsevier, 2008 (Electronic Notes in Theoretical Computer Science). – ISSN 1571–0661, 67–88

[PSH06] Platon, Eric ; Sabouret, Nicolas ; Honiden, Shinichi: Overhearing and direct interactions: Point of view of an active environment, a preliminary study. In: Weyns, Danny (Hrsg.) ; Parunak, H. V. (Hrsg.) ; Michel, Fabien (Hrsg.): *Proc. Environments for Multi-Agent Systems II (E4MAS 2005)* Bd. 3830, Springer, 2006 (Lecture Notes in Computer Science), 121–138

[PT07] Pelz, Elisabeth ; Tutsch, Dietmar: Formal models for multicast traffic in network on chip architectures with compositional high-level Petri nets. In: *Proc. 28th Int. Conf. on Applications and Theory of Petri Nets and other Models of Concurrency (ICATPN)*. Berlin, Heidelberg : Springer, 2007. – ISBN 978–3–540–73093–4, 381–401

[Rei85a] Reisig, Wolfgang: *EATCS Monographs on Theoretical Computer Science*. Bd. 4: *Petri Nets: An Introduction*. Springer, 1985

[Rei85b] Reisig, Wolfgang: Petri Nets with Individual Tokens. In: *Theoretical Computer Science* 41 (1985), S. 185–213

[Rei91] Reisig, Wolfgang: Petri nets and algebraic specifications. In: *Theoretical Computer Science* 80 (1991), March, Nr. 1, S. 1–34

[RET11] Runge, Olga ; Ermel, Claudia ; Taentzer, Gabriele: AGG 2.0 – New Features for Specifying and Analyzing Algebraic Graph Transformations. In: Schürr, A. (Hrsg.) ; Várro, D. (Hrsg.): *4th Int. Symp. of Application of Graph Transformation with Industrial Relevance (AGTIVE'11)*, Springer, 2011 (Lecture Notes in Computer Science). – To Appear

[Ric83] Rich, Elaine: Users are individuals: individualizing user models. In: *International Journal of Man-Machine Studies* 18 (1983), S. 199–214

[Roz97] Rozenberg, Grzegorz (Hrsg.): *Handbook of graph grammars and computing by graph transformation: volume I. foundations*. River Edge, NJ, USA : World Scientific Publishing Co., Inc., 1997. – ISBN 98–102288–48

[RSV97] Rowson, James A. ; Sangiovanni-Vincentelli, Alberto: Interface-based design. In: *Proc. of the 34th annual Design Automation Conference (DAC)*. New York, NY, USA : ACM, 1997. – ISBN 0–89791–920–3, 178–183

[RT86] Rozenberg, Grzegorz ; Thiagarajan, P.S.: Petri Nets: Basic notions, structure, behaviour. In: Bakker, J. W. (Hrsg.) ; Roever, W. P. (Hrsg.) ; Rozenberg, G (Hrsg.): *Current Trends in Concurrency* Bd. 224, Springer, 1986 (Lecture Notes in Computer Science), 585–668

[Rud01] Rudolph, Larry: Project Oxygen: Pervasive, Human-Centric Computing – An Initial Experience. In: Dittrich, Klaus (Hrsg.) ; Geppert, Andreas (Hrsg.) ; Norrie, Moira (Hrsg.): *Advanced Information Systems Engineering* Bd. 2068. Springer, 2001, S. 1–12

[SB94] Sibertin-Blanc, Christophe: Cooperative Nets. In: Valette, Robert (Hrsg.): *Application and Theory of Petri Nets (APTN)* Bd. 815, Springer, 1994 (Lecture Notes in Computer Science), 471–490

[SBPM09] Steinberg, David ; Budinsky, Frank ; Paternostro, Marcelo ; Merks, Ed: *EMF: Eclipse Modeling Framework 2.0.* 2nd ed. Addison-Wesley Professional, 2009. – ISBN 0321331885

[Sie04] Siegemund, Frank: A Context-Aware Communication Platform for Smart Objects. In: *Proc. of the 2nd Int. Conf. on Pervasive Computing* Bd. 3001, Springer, 2004 (Lecture Notes in Computer Science), 69–86

[SS01] Schoberth, Thomas ; Schrott, Gregor: Virtual Communities. In: *Wirtschaftsinformatik* 43 (2001), Nr. 5, S. 517–519

[Ste03] Steglich, Stephan: *I-centric User Interaction*, Technische Universität Berlin, PhD thesis, November 2003

[Tae04] Taentzer, Gabriele: AGG: A Graph Transformation Environment for Modeling and Validation of Software. In: Pfaltz, J. (Hrsg.) ; Nagl, M. (Hrsg.) ; Boehlen, B. (Hrsg.): *Application of Graph Transformations with Industrial Relevance (AGTIVE'03)* Bd. 3062. Berlin : Springer, 2004, S. 446 – 456

[TBG91] Tarlecki, Andrzej ; Burstall, Rod M. ; Goguen, Joseph A.: Some Fundamental Algebraic Tools for the Semantics of Computation: Part 3: Indexed Categories. In: *Theoretical Computer Science* 91 (1991), Nr. 2, S. 239–264

[TEG$^+$05] Taentzer, Gabriele ; Ehrig, Karsten ; Guerra, Esther ; Lara, Juan de ; Lengyel, Laszlo ; Levendovsky, Tihamer ; Prange, Ulrike ; Varro, Daniel ; Varro-Gyapay, Szilvia: Model Transformation by Graph Transformation: A Comparative Study. In: *Proc. Workshop Model Transformation in Practice (at MODELS)* ACM/IEEE, 2005

[Val78] Valk, Rüdiger: Self-Modifying Nets, a Natural Extension of Petri Nets. In: Ausiello, Giorgio (Hrsg.) ; Böhm, Corrado (Hrsg.): *5th Coll. Automata, Languages and Programming (ICALP), Udine, Italy* Bd. 62, Springer, 1978 (Lecture Notes in Computer Science), 464–476

[Val98] Valk, Rüdiger: Petri Nets as Token Objects: An Introduction to Elementary Object Nets. In: *Proc. 19th Int. Conf. on Application and Theory of Petri Nets (ICATPN)* Bd. 1420, Springer, 1998 (Lecture Notes in Computer Science). – ISBN 3–540–64677–9, 1–25

[Win87] Winskel, Glynn: Petri nets, algebras, morphisms, and compositionality. In: *Information and Computation* 72 (1987), S. 197–238

[WWR03] Wireless World Research Forum (WWRF): *I-centric Communications – Basic Terminology*. Whitepaper. Working Group 2 "Service Architectures for the Wireless World". http://www.wireless-world-research.org/fileadmin/sites/default/files/about_the_forum/WG/WG2/White%20Papers/WWRF-WG2-WP1-Terminology.pdf. Version: December 2003

[ZZZY10] Zhang, Dongyan ; Zheng, Chao ; Zhang, Hongli ; Yu, Hongliang: Identification and Analysis of Skype Peer-to-Peer Traffic. In: *5th Int. Conf. on Internet and Web Applications and Services*. Los Alamitos, CA, USA : IEEE Computer Society, 2010. – ISBN 978–0–7695–4022–1, 200–206

Index