Virtual Teaching Interface for E-Learning Capable Simulations

Mang Li

München 2004

Virtual Teaching Interface for E-Learning Capable Simulations

Mang Li

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität
München

vorgelegt von
Mang Li

München, den 12.03.2004

Erstgutachter: Prof. Dr. C. Linnhoff-Popien
Zweitgutachter: Prof. Dr. I. Schieferdecker

Zusammenfassung

Diese Dissertation ist ein Beitrag zur Entwicklung von E-Learning Systemen, die generell als Learning Technology Systems (LTSs) bezeichnet werden. Die Arbeit befasst sich mit dem Aufbau und der Integration von *Virtuellen Betreuern* in LTSs im Rahmen von *e-learning-fähigen Simulationen*. Ein virtueller Betreuer automatisiert Funktionen, die ein menschlicher Betreuer normalerweise ausübt und erlaubt so mehr Flexibilität bei geringeren Kosten.

Gemäß dem pädagogischen Konzept des situierten Lernens unterstützt eine e-learning-fähige Simulation die Darstellung von domänen-spezifischem Wissen und stellt eine interaktive und verteilte Kollaborationsumgebung bereit. Sie erlaubt außerdem die automatisierte Verarbeitung von Lehrstrategien, die die pädagogischen Handlungen von menschlichen Betreuern modellieren.

Diese Dissertation präsentiert einen neuen daten-orientierten Ansatz, der die genannten Anforderungen erfüllt. Der in dieser Arbeit beschriebene Ansatz trägt den Namen *VTIS (Virtual Teaching Interface for e-learning capable Simulations)*. Der Ansatz basiert auf der Sammlung historischer Daten über den Verlauf einer interaktiven Simulation, die über eine verteilte Nutzer-Umgebung gesteuert wird. Die Integration von virtuellen Betreuern in eine solche Simulationsumgebung wird durch die Manipulation der historischen Daten mit *aktiven Regeln* erreicht. Im Vergleich zu den bisherigen Ansätzen kann dadurch eine bessere Adaptivität und Erweiterbarkeit der virtuellen Betreuer erzielt werden. Dieser Ansatz erlaubt außerdem eine freundlichere Entwicklungs-Schnittstelle für die menschlichen Betreuer, die in diesem Kontext durch die Formalisierung der Lehrstrategien ihre pädagogischen Erfahrungen in die Funktionalität der virtuellen Betreuer übertragen.

Die hier präsentierten Konzepte wurden prototypisch implementiert, um sie in der Praxis evaluieren zu können. Die Evaluation wird im Rahmen eines Praktikums stattfinden, das von den Universitäten LMU München und RWTH-Aachen zusammen veranstaltet wird. In der vorliegenden Dissertation ist eine Beispielaufgabe für das Praktikum enthalten, die die Anwendung der VTIS-Konzepte und -Werkzeuge demonstriert.

Abstract

This thesis contributes to the development of modern e-learning systems, which are generally referred to as Learning Technology Systems (LTSs). It is concerned with the construction and the integration of *virtual teachers* in LTSs, considered in the technological context of *e-learning capable simulations*. A virtual teacher automates certain functions a human teacher usually performs. Once a virtual teacher has been created, it allows greater flexibility at less expenses.

According to the pedagogical concepts for situated learning, an e-learning capable simulation is required to provide the representation of domain knowledge, as well as an interactive and distributed collaboration environment. In addition, it should support the automated processing of teaching strategies which model the pedagogical handling of human teachers.

This thesis covers the various aspects of the requirements by a data-oriented approach, called *VTIS (Virtual Teaching Interface for e-learning capable Simulations)*. The approach relies on the history data of interactive simulations, that are controlled by users over a distributed environment. It uses active rules operating on the history data, in order to achieve the integration of virtual teachers in e-learning capable simulations. Compared to existing approaches, not only more adaptivity and extensibility of virtual teachers, but also a teacher-friendly development interface can be achieved.

The concepts presented in the thesis have been implemented for an empirical evaluation. The evaluation will be embedded in a practical course that will be organized by the two universities LMU Munich and RWTH Aachen. An example task for the practical course demonstrates how the VTIS concepts and tools can be applied.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

E-learning, as many other application areas of information technology, has experienced tremendous development in recent years. On one side, requirements derived from traditional learning scenarios and learning theories guide the creation of e-learning artifacts. On the other side, the utilization of information technologies opens opportunities for new learning methods. Not only technical expertise, but also non-technical expertise, especially pedagogical knowledge, is essential for well-founded e-learning. This has been shown in numerous interdisciplinary research and development projects, among them the VIP-project[1] [LLPM+03a], which provides the context and examples for this thesis.

This thesis contributes to the development of computer simulation-based e-learning software. The scope, motivation and structure of the thesis are introduced in this chapter. Section 1.1 gives an overview on related terminologies and concepts of e-learning. In Section 1.2, the main problem to be addressed is outlined. The structure of the thesis revolving this problem is then presented in Section 1.3.

1.1 E-Learning

The term *e-learning* emerged just a few years ago, although the idea of information technology-supported learning has been followed already since the beginning of the computer era [O'S82]. Thanks to the research in artificial intelligence starting in the 1980s, and the rapid development of the Internet and multimedia technologies in the past decade, numerous computer-based learning systems of a great variety are available today. There exit also various denotations and definitions for such systems.

Definitions

As other e-terms, such as e-business, e-government, e-library, etc., e-learning is understood as a field involving some kinds of digital information representation. In the literature, several more

[1]The project "Virtuelles Informatik Praktikum (VIP)" is sponsored by the Ministry for Education and Research of the German Government.

or less formal definitions for e-learning can be found. Three representative ones are cited in the following:

> Broad definition of the field of using technology to deliver learning and training programs. Typically used to describe media such as CD-ROM, Internet, Intranet, wireless and mobile learning. [E-L03]
>
> Learning that is accomplished over the Internet, a computer network, via CD-ROM, interactive TV, or satellite broadcast. [Wor03f]
>
> Internet-enabled learning that encompasses training, education, just-in-time information, and communication. [Cis01]

Denotations, such as computer-supported learning, distance learning, web-based learning, multimedia learning, and on-line learning, arose during the evolution of e-learning systems. They are still frequently used as synonyms for e-learning.

This work focuses on software systems for e-learning which are generally denoted as **Learning Technology Systems** (LTSs) . LTSs are characterized by the IEEE Learning Technology Systems Committee (LTSC) as information technology-supported learning, education, and training systems [IEE01].

Initiatives

A considerable number of initiatives has been launched to push forward the research and development of e-learning. Besides IEEE LTSC, the international initiatives include also the IMS Global Learning Consortium [IMS], the Advanced Distributed Learning Initiative (ADL) [Adv], the Aviation Industry CBT (Computer-Based Training) Committee (AICC) [Avi], and the ARIADNE Foundation [ARI]. These organizations provide reference models, frameworks or guidelines for the development of e-learning systems. In addition, national programs, such as the German Government Programme ”New Media in the Education” [Ger], support the cooperative work between universities and the industry. Several websites, e.g. E-Learning Guru [E-L03] and WorldWideLearn [Wor03f], provide also interesting information for developers.

Benefits

In world-wide interdisciplinary projects, professionals of information technologies and pedagogy have been working together on case studies. In the first results published recently [Her01] [FM02] [Sch01] [Wol01] [Sch97] [Sti01], experts agree to the following benefits of e-learning:

1. A great variety of forms for knowledge representation and knowledge transfer is available in addition to conventional techniques.

2. Learning becomes independent from location and time, which allows life-long learning in different situations.

3. Valuable material and human resources can be used more efficiently.
4. Learner-centered learning can be better supported through interactive and adaptive learning environments.

In general, a shift from the traditional classroom learning to interactive learning has been taking place. Some major trends summarized in [Tap98] are cited in Table 1.1.

Classroom learning	**Interactive learning**
Linear	Hypermedia
Instruction	Construction
Teacher-centered	Learner-centered
School learning	Life-long learning

Table 1.1: From classroom learning to interactive learning

Critical Issues

To fully achieve the benefits of e-learning, some critical issues need to be further addressed. One major problem identified in the case studies is, that the huge amount of information sources and the multiplicity of new technologies are overwhelming for some learners. Teachers and tutors have spent even much more time to guide them to fulfil learning objectives than using traditional means. Obviously, this is opposite to the expectation of more efficient use of resources.

This problem motivates the approach presented in this thesis, which aims on effective and efficient teaching support by automated mechanisms. It is believed that the success of e-learning depends very much on the availability and the usefulness of the teaching support. A first analysis of the problem to be solved is given in the following.

1.2 Problem Statement

Traditionally, teaching is carried out by a human teacher. During the evolution of e-learning, two main approaches to teaching support have been formed. One is referred to as *tele-teacher*. The other one is denoted in this thesis as *virtual teacher*.

Tele-Teacher vs. Virtual Teacher

Flexibility in location for students [2] and teachers is a major benefit of the tele-teacher approach. As illustrated in Figure 1.1, the basic concerns in supporting tele-teachers [ILP01] or tele-tutors [KT99] [FM02] are the distribution of audio and video material, e.g. lectures, slides or documents, as well as the person-to-person communication between students and teachers. Well-known multimedia techniques, and the so-called ”groupware” supporting cooperative work, are found to be very useful.

[2] Student and learner are used as synonyms in this thesis.

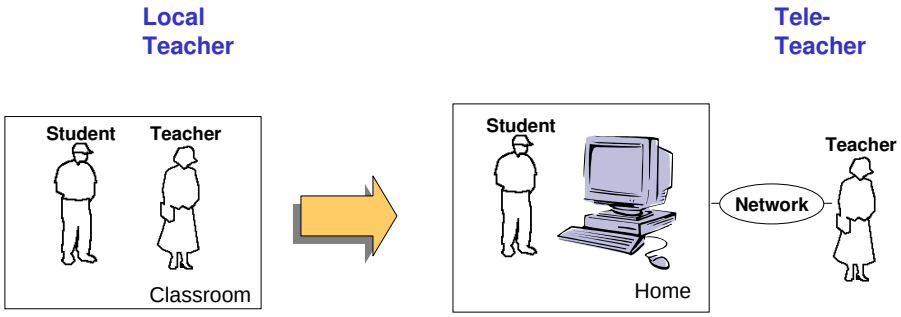

Figure 1.1: From local teacher to tele-teacher

For distributed lectures, or similar broadcasting-like scenarios, in which a large audience can be reached by less frequent interaction, tele-teacher is a rather efficient approach. However, due to the fact, that the face-to-face communication can not be fully imitated, in situations where individual treatment is desired, providing teaching over distance can be more resource-consuming than doing it locally.

Virtual teacher is an approach of automation to save human resources. It is an orthogonal approach to tele-teacher, as represented by Figure 1.2. Generally, efforts in computer-aided instruction [O'S82], intelligent tutoring [JD96] [DHM+93] and pedagogical agent [SJG99] can be classified to the category of virtual teacher. The main idea is to use automated mechanisms, not necessarily to approximate a teacher as a human being (e.g. his/her look and voice), but always to approximate the functions such a human teacher carries out, for example, doing demonstration, giving explanation, providing advice or making assessment.

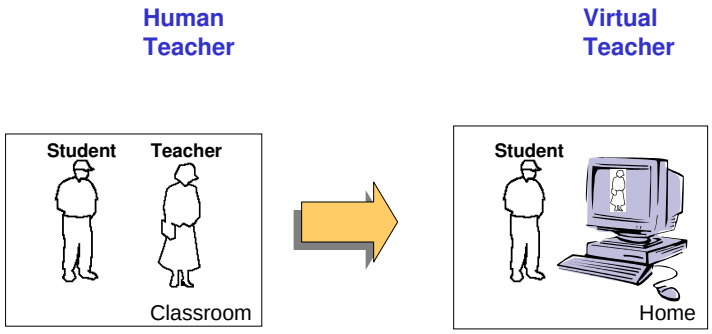

Figure 1.2: From human teacher to virtual teacher

Virtual teachers do not fully replace human teachers. Human contact is in many cases indispensable. However, a virtual teacher can be a useful complement. Thanks to the rapid development of computation power and wide-spread networks, once a virtual teacher has been created, it can be applied at great flexibility and less expense.

Scope of the Work

The critical question is: **How to create a useful virtual teacher?** This work seeks an answer in a concrete technical context which is denoted as **e-learning capable simulation**.

How teaching is accomplished depends very much on the knowledge to be taught. It depends also on the context of teaching, including the representation of the knowledge, and the environment in which teaching is accomplished. Taking language teaching as an example, teaching by book differs to teaching by conversation. A book represents the written language. Depending on the situation, in which the learner may use it, it can be a grammar book, a dictionary, a novel, etc. In contrast, teaching by conversation usually applies the method of articulation. It can be practiced in school, or elsewhere the language is frequently spoken. The harmony of the learner's demand (e.g. learning a language), the means for teaching (e.g. book or conversation), and the pedagogical skill (e.g. how to write a book, or how to conduct a conversation), is crucial for teaching.

This work focuses on the technical support for teaching. It considers in particular computer simulations (subsequently only simulations) as the core technology providing the means for teaching. A simulation is in general a computer program that represents the behavior of a physical system or a process. Simulation techniques are widely used in different areas of research and industry. Using simulations in education is primarily to facilitate "learning by doing". According to the "Cone of Learning" of Edgar Dale [Dal54], we remember 90% of what we do, but only 10% of what we read, 20% of what we hear, and 30 % of what we see. Simulations are used frequently as alternatives to experiments in laboratories that may be expensive, time consuming or even dangerous. Simulations are also very popular for pilot training or similar purposes.

The demand for simulations in e-learning has been growing fast [Adk02]. However, most simulations used so far for learning are actually general-purpose tools, such as pens. Giving someone a pen, who learns first time writing or painting, without instructing how to use it, is not enough. Some e-learning approaches rely on tele-tutors [KT99] [Kle99]. As stated earlier, a resource problem may arise when supporting individual learners.

This thesis presents a concept of e-learning capable simulations with integrated virtual teachers. Key issues to be addressed are the following:

- How to classify the requirements of learners and teachers for such systems, in order to achieve the harmonization with the means for teaching?
- How well are these requirements covered by existing simulation concepts, because simulation is a well-studied area?
- With respect to the requirements, which functions of a human teacher can be automated by a virtual teacher, and how to describe those functions in a machine-readable form?
- How to provide an appropriate means for human teachers to contribute to the development of virtual teachers?
- How to integrate virtual teachers with simulations in a manner that the system developed can be easily adapted and extended?

The starting point of the work is a substantial requirement analysis. Based on this, the thesis comprises concepts from several research areas to cover the entire spectrum of the key issues, as outlined in the following.

1.3 Thesis Structure

The structure of the thesis is illustrated in Figure 1.3. It consists mainly of an analysis (Chapter 2), an approach overview (Chapter 3), a detailed elaboration of the approach consisting of three parts (Chapter 3, 4 and 5), as well as implementation (Chapter 7) and evaluation concepts (Chapter 8). The chapter and section headings in Figure 1.3 are abbreviated by key words.

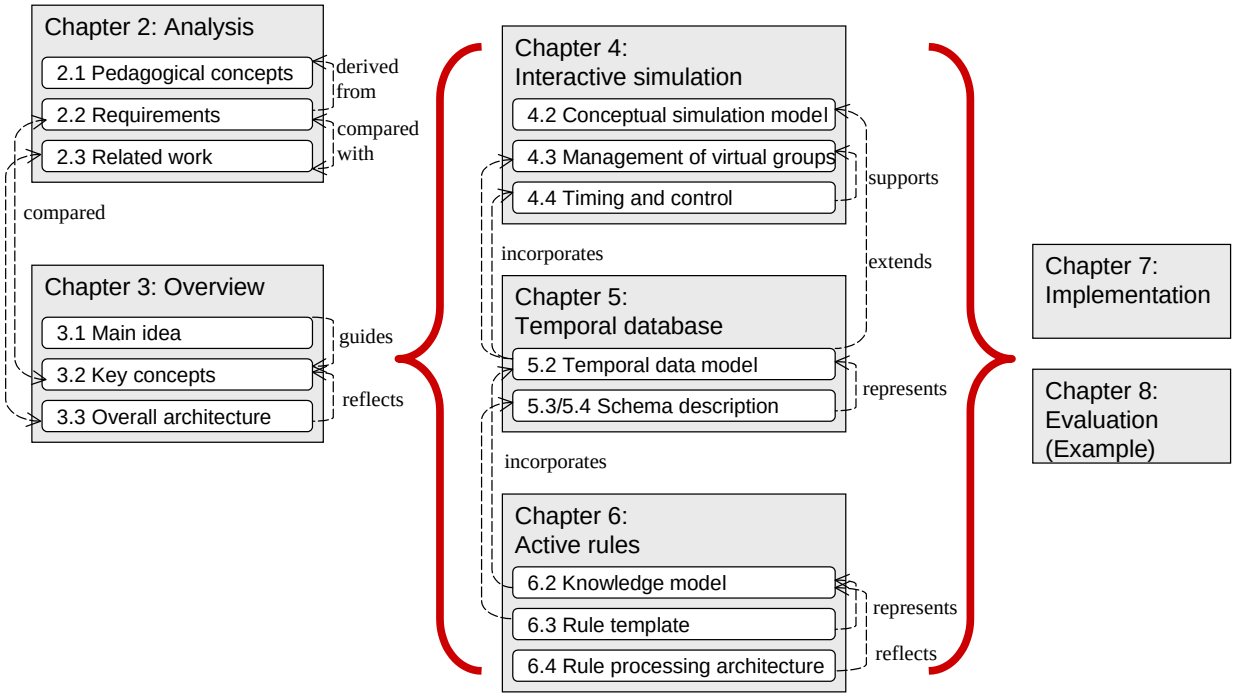

Figure 1.3: Structure of the thesis

In the following, this structure is described in greater detail:

Chapter 2 introduces the pedagogical foundations of this work. The core concept is called *situated learning*, in which students obtain problem-solving knowledge in authentic learning contexts through explorative work. Based on these concepts, technical requirements on e-learning capable simulations are derived. They are classified into requirements for simulations as learning context, and for simulations with integrated virtual teachers. In particular for the latter, the modeling of teaching methods by teaching strategies is a crucial issue. These requirements are compared with known approaches and reference models.

Chapter 3 gives an overview on the main idea and key concepts of the approach presented in this thesis, which is called VTIS (Virtual Teaching Interface for e-learning Simulations). The main idea of VTIS is to use a data-oriented approach to integrate virtual teachers into simulations. That is, automated mechanisms supporting virtual teachers operate on the simulation history data that reflects simulation executions including user interaction. In this manner, the functionality of a virtual teacher can be easily coupled and decoupled

with a simulation execution, in order to achieve greater flexibility. Prerequisite for doing this are an adequate data model for the history data, and proper mechanisms to interpret the history data for automated processing of teaching strategies. These are the core issues discussed in Chapter 3, 4 and 5. The overall VTIS architecture is also included in this chapter.

Chapter 4 presents a conceptual model for simulation applications, which is used as the basis for the temporal data model introduced in Chapter 5. In addition, the modeling of virtual groups in distributed learning environments is elaborated, whereas both the spatial distribution and the temporal distribution of such an environment are taken into account. The spatial distribution is incorporated in a distributed simulation architecture. Timing and control mechanisms supporting the temporal distribution are also carefully examined.

Chapter 5 elaborates the modeling of a temporal simulation database that stores history data for simulation executions. The temporal data model extends the conceptual model for simulation applications with time concepts that incorporate the particular characteristics of simulations as considered in Chapter 4. A generic concept for data schemas based on the data model is also presented.

Chapter 6 discusses the automated processing of teaching strategies using active rules. A knowledge model describes the rules as event-condition-action rules, that are used to formalize the pedagogical handling of human teachers as teaching strategies. The event part of a rule is sensitive to modifications of the simulation database, so that the processing of the rule can be aligned to the simulation execution. This is a crucial point for the data-oriented integration of virtual teachers. The rules description is supported by a rule template based on a data query language. A rule processing architecture is also presented.

Chapter 7 introduces the prototype implementation of the approach.

Chapter 8 includes concepts to evaluate the approach in practice. A comprehensive example demonstrates the usability of the approach.

Chapter 9 closes the thesis with a summary of contributions and an outlook over future work.

Chapter 2

Situated Learning with E-Learning Capable Simulations

This chapter presents an analysis of the technical requirements on e-learning capable simulations. Section 2.1 introduces first the concepts of situated learning. They build the pedagogical foundation for the discussion in Section 2.2, where the characteristics of the desired e-learning systems are described. Section 2.3 gives a review on related reference models and approaches.

2.1 Pedagogical Concepts of Situated Learning

Pedagogy, the art, science, or profession of teaching [Onl04], cooperates with different other disciplines [Kro01] [Dit03]. Two of the most closely related disciplines are psychology and sociology. Psychology is concerned with students and teachers as individuals, while sociology focuses on the social context of students and teachers, whereas the social context may be either region, culture or family related. Aspects of psychology and sociology are incorporated in the concepts of situated learning, as introduced in the following.

2.1.1 Classification of Learning Theories

The empirical branch of pedagogy, often referred to as educational science, emerged in the 1970s. It is characterized by the application of scientific experimental methods to prove and to improve theories. In this realm, two extremes are distinguished: the behaviorism and the constructivism.

Behaviorism

The behaviorism emphasizes on observable human behavior as indicator of learning [FJ98]. In this sense, learning is defined as a sequence of stimulus and response actions in observable cause and effect relationships. A teacher modifies the behavior of students until they exhibit the desired response.

The criticism on behaviorism are the unconditional association between stimuli and responses, and the dominance of teacher's instructions. The learner's mental process is treated as unobservable indication of learning, and is insufficiently considered. However, for some skills the approach of behaviorism works effectively. Therefore, it is still practiced in the education today.

Constructivism

In contrast to behaviorism, constructivism focuses on the motivation and ability of humans to construct learning for themselves (see Table 2.1). Through a process of discovery and problem-solving, a human is believed to be able to construct knowledge in their own minds. A teacher is in the role of a motivator and a supporter.

Constructivism is linked to cognitive learning, which considers learning as a mental process of information processing [Her01]. In this process, a learner selectively absorbs information in order to construct own knowledge. The learner is the center in constructivism. In addition, group work is promoted.

Behaviorism	**Constructivism**
Directed instruction	Non-directed instruction
Objectivist	Constructivist
Teacher-centered	Learner-centered
Behavioral observations	Cognitive operations
Focus on the individual	Group work is emphasized

Table 2.1: Behaviorism and constructivism in learning theories [FJ98]

Constructivism arose in the 1970s and is very popular in the modern pedagogy. But behaviorism is not forgotten. Moreover, a new position called situated learning came out in the 1980s, which in fact re-associates ideas of the two [Her01], as illustrated in Figure 2.1.

Situated Learning

In this approach, learning is not a pure self-organized process, but a combination of instruction and construction. It is "an active, constructive and highly situated process" [MGF00]. It emphasizes on authentic learning contexts, self-regulation, as well as social aspects. Cognitive apprenticeship [CBN89], problem-oriented learning [MGF00] and students-centered learning [LH00] are some representatives in this realm.

Learning and cognition are fundamentally situated, i.e. *situated cognition* [BCD89]. Knowledge cannot be simply transferred from teacher to learner, but can only be fully understood in authentic contexts. For example, the best context for learning a natural language is ordinary conversation with native speakers. Therefore, the situated cognition approach promotes learning in authentic situations, in which students are guided into the "culture of experts" [CBN89] through activities and social interactions.

Learning through cognitive apprenticeship [CBN89] follows the model of traditional apprenticeship, which had been practiced long before schools appeared. The approach is as follows:

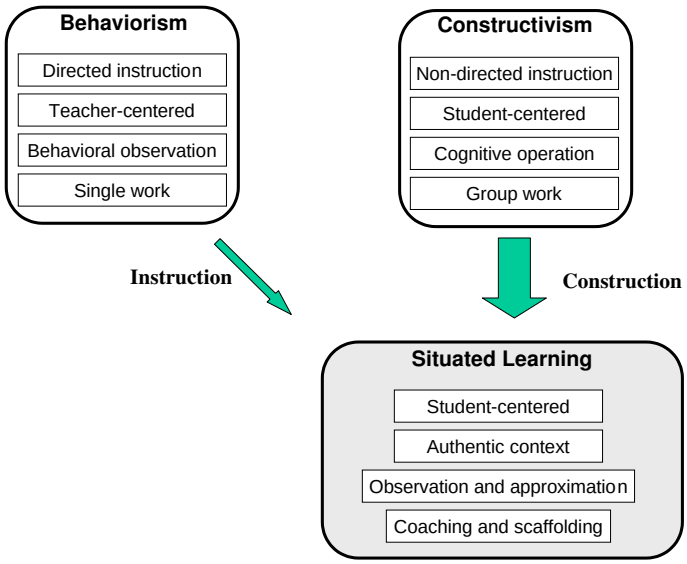

Figure 2.1: Classification of learning theories

teachers first demonstrate the task, then support students doing the task, and finally, the students are encouraged doing the task independently. In apprenticeship, group learning is essential to stress social interaction and collaborative learning. For students, this kind of learning is a process involving observation and successive approximation. From the teachers' point of view, cognitive apprenticeship is a process of modeling, coaching, scaffolding and fading, in order to support students to proceed from observation towards independent-doing.

2.1.2 Frameworks of Learning Environments

Learning environments for situated learning can be denoted as constructivist learning environments. Two representative frameworks are introduced in the following.

Framework by Collins, Brown and Newman

Based on the constructive apprenticeship approach, Collins, Brown and Newman proposed a framework [CBN89] for the design of learning environments. This framework has been guiding a number of research work later, such as [LH00] [FTM01] [MGF00] [BTW+94] and [SJG99]. It is also used as the pedagogical foundation in this work.

This framework (CBN-FM in the following) describes four dimensions of an ideal learning environment: content, methods, sequence and sociology, as summarized in Table 2.2. Each dimension consists of a set of desired characteristics, as described in the following.

Content includes different types of target knowledge for students to learn, which are basically domain knowledge and strategic knowledge.

- **Domain knowledge** denotes explicit conceptual, factual, and procedural knowledge associated with expertise. Although explicated in books, lectures or demonstrations,

Dimension	Characteristics
Content	Domain knowledge
	Heuristic strategies
	Control strategies
	Learning strategies
Methods	Modeling
	Coaching
	Scaffolding and fading
	Articulation
	Reflection
	Exploration
Sequence	Increasing complexity
	Increasing diversity
	Global before local skills
Sociology	Situated learning
	Culture of expert practice
	Intrinsic motivation
	Exploiting cooperation
	Exploiting competition

Table 2.2: Characteristics of ideal learning environments ([CBN89])

this kind of knowledge tends to remain inaccessible unless it is learned through practice in appropriate situations. Some domain knowledge is difficult to formalize, which makes strategic knowledge even more important.

- **Strategic knowledge** refers to experts' ability to make use of domain knowledge to solve problems. It includes heuristic strategies, control strategies and learning strategies. *Heuristic strategies* are gained from best practice. They are "tricks of the trade" that could be quite helpful when they work. *Control strategies* serve the control when processing a task, e.g. to select or to change problem solving strategies. Thereby, monitoring is crucial for diagnosis, which is to make assessment of the current state relative to one's goals. Diagnosis is then followed by remedy to finally achieve the desired goals. *Learning strategies* are knowledge about how to learn domain knowledge and other kinds of strategic knowledge, e.g. how to explore a new domain, how to check own understanding, or how to discuss with others.

Methods refer to teaching methods. Modeling, coaching, scaffolding and fading are core methods to help students to acquire knowledge through guided practice. Articulation and reflection help students to gain their own problem-solving strategies through observation and reasoning. Exploration is to push students to solve problems on their own.

- **Modeling** involves demonstration of processes and activities of a task (which can be usually internal ones), so that students can build conceptual models in order to

accomplish the task on their own. It requires insight to those processes and activities.

- **Coaching** offers hints to students when they carry out a task, for example to direct them to some unnoticed or overlooked aspects. Coaching involves highly interactive and highly situated feedback and suggestions.
- **Scaffolding and fading** are applied to help students to perform a task on their own, whereby scaffolding refers to suggestions or help, and fading means gradual removal of supports until the students are on their own. Key points of this method are accurate diagnosis of the student's current skill level or difficulty, and the availability of an intermediate step at the appropriate level of difficulty in carrying out the target activity.
- **Articulation** is to get students to articulate their knowledge, reasoning, or problem-solving processes in a domain.
- **Reflection** uses different techniques for reproducing and replaying, so that students can compare their own problem-solving processes with those of an expert, another student or a model of expertise.
- **Exploration** is to push students not only to solve problems on their own, but first to identify those problems from general goals. Therefore, focus of this method is to teach students exploration strategies.

Sequence is concerned with the sequencing of learning activities to facilitate the development of robust problem-solving skills. Since learning is a process of knowledge development, learning materials and activities must be adapted to different stages of skill acquisition.

- **Increasing complexity** (i.e. from easy to complex) is a well-understood and widely-applied strategy. The goal is to learn to control complexity.
- **Increasing diversity** (i.e. from single to various) serves to widen and to generalize knowledge.
- **Global before local skills** (i.e. first big picture, then details) helps students to understand the overall concept, in order to be able to put pieces of a task together by themselves.

Sociology is a critical dimension for skills that are learned and performed in communities. It involves different social abilities of students.

- **Situated learning** is the core of constructivist learning. It is to put students into situations that reflect the environment, in which they will apply the acquired knowledge in the future. In various problem-solving contexts, the students learn how, when, and under which conditions to apply knowledge.
- **Culture of expert practice** promotes focused interactions among learners and experts. In this kind of culture, students learn to think and to act like experts.

- **Intrinsic motivation** is difficult to measure, but in many cases essential for the success of teaching methods. Some strategies can be used to awake intrinsic motivation.
- **Exploiting cooperation** refers to learning through cooperative problem solving, in which students can motivate and help each other.
- **Exploiting competition** is to get students comparing their own problem solving processes and results with those of other students. Since competition may cause ill effects, blending cooperation and competition is crucial.

Obviously, the characteristics of the four dimensions are inter-related, at some points even dependent to each other. For example, the method modeling is applied to the representation of domain knowledge and strategic knowledge. And exploration can be carried out in cooperation and competition. The classification helps to focus the consideration on aspects of interest.

CBN-FM is a well-established work but not the only one considering constructivist learning environments. The framework by Land and Hannafin [HL97] is another work worth to mention, because it addresses some issues related to information technology-supported learning environments.

Framework by Land and Hannafin

This framework was intended for *student-centered learning environments*. It is guided by core concepts of situated learning. The framework suggests grounded designed learning environments that ”provide interactive, complementary activities that enable individuals to address unique learning interests and needs, study multiple levels of complexity, and deepen understanding” [HL97].

A grounded design should incorporate five foundations: psychological, pedagogical, technological, cultural and pragmatic. *Psychological foundations* emphasize theory and research related to how individuals think and learn. *Pedagogical foundations* form the affordances and activities of the environment. *Technological foundations* influence how media can support, constrain, or enhance the learning environment. *Cultural foundations* reflect the prevailing values of a learning community. *Pragmatic foundations* emphasize the reconciling of available resources and constraints with the actual design of a learning environment.

The common values of such learning environments include:

- Centrality of the learner. The learner defines actively how to proceed based on individual needs and questions.
- Situated thinking and authentic contexts. Students learn to solve problems in various authentic contexts, rather than learning a single correct solution.
- Negotiation and interpretation. The socially mediated aspects of learning can be supported by teacher-student or student-student interactions, in order to obtain deepened knowledge through interpretation, explanation and exploration.

- Use of prior knowledge. Individual beliefs and experiences provide uniquely personal framework for building new knowledge.
- Technology scaffolding. Technology enables learners to represent their thinking in concrete ways and to visualize and test the consequences of their reasoning. At this point, the information technologies are considered as useful means supporting learning.

The basic principles of situated learning are considered applicable to information technology-supported learning. This has been also confirmed in different case studies [Her01] [FM02] [Wol01]. In particular, CBN-FM provides a comprehensive framework for the construction of learning environments for situated learning. It is used in the next section to derive the technical requirements for e-learning capable simulations.

2.2 Requirements on E-Learning Capable Simulations

This section is dedicated to the requirement analysis for e-learning capable simulations. Object-oriented analysis and design are applied in this work. The discussion of roles in Section 2.2.1 builds the first step of the analysis. The requirements themselves are classified in two parts. Section 2.2.2 discusses the characteristics of a simulation as a learning context, while Section 2.2.3 considers a simulation with integrated virtual teacher.

2.2.1 Roles

LTS (Learning Technology System) is introduced in Section 1.1 as a general notion for information technology-supported learning, education, and training systems. Two other frequently used terms worth to mention are Learning Management System (LMS) and Virtual Learning Environment (VLE). LMS refers to a system ”that focuses on management and facilitation of student learning activities, along with the provision of content and resources required to help make the activities successful” [Sti00]. VLE is viewed as a designed information space, where educational interactions take place [Dil00]. The scopes covered by the three are considered comparable, although some details differ to each other. To simplify, only LTS is used subsequently as a superset of learning systems that may be declared as LMS, VLE, or similar in the literature.

Roles that abstract persons involved in the development and utilization of an LTS are presented in Figure 2.2.

On the top level are the roles *user* and *developer*. Usually, when constructing a specific LTS, only the user role is considered. As this work is interested particularly in the development methods for LTSs, the developer role needs to be considered, too.

From the user role, two further roles are derived, namely, *student* and *supporter*. Student is the center of an established LTS. The demands of a student are determining for the design of an LTS. Whereby, the supporter role as a special kind of LTS user cannot be ignored. Examples for a supporter are tele-teachers [ILP01] and tele-tutors [FM02], who accompany and guide students when using an LTS. A supporter may use in some cases the same functions of an LTS as a student, but places in other cases different requirements on an LTS than a student. For example, chat is

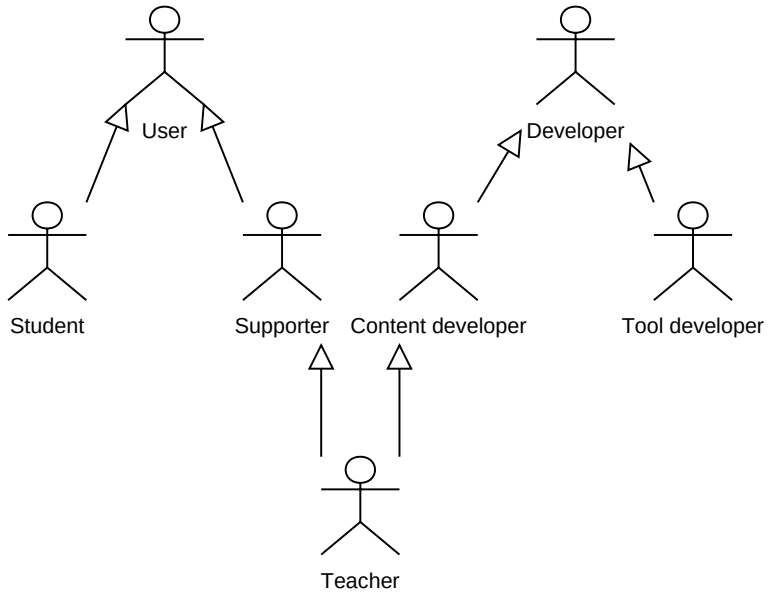

Figure 2.2: Roles related to an LTS

a communication tool of an LTS that can be equally used by a student and by a supporter, while the management facility for student groups is only interesting for a supporter of these groups. For some purposes, a supporter may need a different view on the same material viewed by a student. For example, it may be useful to know how an essay has been produced, rather than its current content, that is, how long it took to finish, how many students have been working on it, how many times it has been modified, etc. Therefore, an LTS that is intended for both students and supporters must cope with their different demands.

The developer role can be further divided into *content developer* and *tool developer*. A tool developer provides artifacts that are necessary to assemble an LTS, including tools for content developers to generate learning content. Content developer is the other role a teacher may take besides the supporter role. The separation of content developer and tool developer has been already implemented in different forms. Traditionally, a teacher puts his/her knowledge into books to pass it to students. The teacher is in this situation a content developer, whereas paper and pens are simple tools used for book-writing. As multimedia techniques are becoming more and more popular, slides, audio records, video clips, etc. are frequently used instead of books. It is the responsibility of tool developers to provide appropriate a means for teachers to create such forms of learning material. This means is referred to as a **content development interface**.

All these roles are incorporated in the following discussion. Particular attention is paid to the content development interface for teachers, because this is considered more challenging in the context of simulation, and has been insufficiently addressed in the literature so far.

In the following two sub-sections, requirements on an e-learning capable simulation are examined. For the summary in Table 2.3, the requirements are associated with identifiers R1 – R6.

2.2.2 Simulation as Learning Context

One of the most important elements in situated learning is the authentic learning context, which resembles situations students will encounter in the future. To support this, the following three requirements of an e-learning capable simulation are important.

R1: Knowledge Representation

Basically, a simulation is an imitation of the operation of a real-world process or system over time [BINN01]. The core of a simulation is its simulation model, which usually takes assumptions about the operation of the system or process to be simulated. The assumptions are expressed in form of mathematical, logical, or symbolic relationships between entities or objects of interest. A simulation model reflects the knowledge about the system to be imitated. This property is crucial for the modeling of the learning context. It supports the desired characteristics *domain knowledge* and *modeling* of CBN-FM.

R2: Interactive Usage

This requirement is important to facilitate explorative work. It corresponds to the characteristics *reflection* and *exploration* of CBN-FM. Interaction refers to a two-way communication, generally at the run-time of a simulation. Delivering information from simulation to the user is only one way. A simulation should also be able to react on requests of the user. For demonstration purpose, appropriate ”look and feel” of a simulation is useful. To which extent ”virtual reality” is necessary is determined by learning objectives. In any case, the interactivity of a simulation must be provided.

R3: Distributed Collaboration Environment

Social ability is essential for many skills. A distributed collaboration environment provides students a virtual space, where they learn to work with others, either in cooperation or in competition. This requirement is to support the desired characteristics *situated learning*, *exploiting cooperation* and *exploiting competition* of CBN-FM.

It is to note, that a virtual space may have, in addition to the conventional **spatial dimension**, a second dimension, which is the **temporal dimension**. Taking telephony and email as examples, telephony spreads over the spatial dimension, as it connects members of a virtual space over different locations. Email has besides a spatial dimension, also a temporal dimension. Email is used as a convenient means to deliver a message to others. The arrival and response of the message occur usually at later points in time.

The spatial distribution of a simulation is a well-studied topic, while few work in the literature addresses the temporal distribution of such an environment. The critical point consists in the fact that a simulation serves as a learning context shared by students meeting in the virtual space. The spatial distribution of the simulation involves allocation of resources across networks, for example to support the distribution of simulation clients or simulation servers (Section 4.1 provides more details on this). The temporal distribution requires, on the other side, the integrity

of the learning context over time. A student may join and leave a virtual space at any point in time. He/she may want to interrupt a simulation, or repeat what others have done before at any point in time. These are basic scenarios the temporal distribution should be able to cope with.

In summary, the three properties elaborated above allow appropriate modeling of the learning context, in which not only students but also teachers as supporters can participate, in order to provide useful aid to students. As discussed in the following, an integrated virtual teacher can relieve a teacher as a supporter by automated mechanisms.

2.2.3 Simulation with Integrated Virtual Teacher

According to CBN-FM, an ideal learning environment should support methods of modeling, coaching, scaffolding and fading, articulation, reflection, as well as exploration. The application of these methods requires not only domain specific knowledge, but also pedagogical skills. It is a matter of ability and training to master both of them.

In the context of an LTS, a third kind of ability is required, which is the ability to use the techniques of the LTS, that serve representation of learning context and realization of pedagogical methods. Prerequisite for building virtual teachers is that a human teacher is able to put his/her knowledge into automated mechanisms as a content developer. Regarding the content development interface, the consideration will concentrate on the methods *coaching*, *scaffolding and fading*. Apart from those which are already covered by the requirements R1, R2 and R3, other desired characteristics, e.g. *increasing complexity*, or *intrinsic motivation*, are interesting for the design of domain specific learning content, but not significant to the design of supporting tools.

A teacher performs coaching, scaffolding and fading in reaction with students and depending on the situation. Usually, a teacher observes when a student carries out a task, and offers suggestion or aid in situations he/she thinks the suggestion or aid would be helpful for the student. The pedagogical skill and experience of the teacher to decide which actions and when to take can be designated as **teaching strategies**. The processing of teaching strategies is the core issue for the realization of virtual teachers. Moreover, a virtual teacher is an integral part of an e-learning capable simulation. To achieve this, three more requirements are considered in the following.

It is noted, that the terms teaching method and teaching strategy are differentiated. Teaching method refers to one of the CBN-FM methods. Teaching strategy is a means of control to carry out certain teaching methods.

R4: Formalization of Teaching Strategies

The formalization of teaching strategies in a machine-processable form is an important first step for the automation.

A crucial point is whether the modeling of teaching strategies is separable from the modeling of learning context. A learning context is mainly determined by the simulation model of the knowledge to be taught. Teaching strategies can be seen as a means of control for the usage of the simulation model. The *separation of the simulation model from its control* allows the

simulation model being used for multiple purposes. In this manner, a simulation model is a general-purpose tool, like a pen. A pen can be used for writing English, Chinese, or something else. How to use the pen needs not to be a feature of the pen, but can be applied in combination with the pen.

The formalization of teaching strategies facilitates also a teacher-friendly content development interface. For teachers who are not familiar with the implementation techniques of an LTS, an abstract description is beneficial for the content development. Furthermore, formalization is necessary to interchange and to standardize virtual teacher mechanisms.

The processing of a teaching strategy consists of two steps. The first one is the assessment about the student who may need help. The second one is the accomplishment of assisting actions to provide help. Details to these two are elaborated in the following.

R5: Assessment about Student

The assessment serves to estimate how a student proceeds with learning objectives. An assessment can be based on simple facts, or statements derived from simple facts. For the latter, diagnosis is a frequently used concept. For example, in [JD00] [DHM+93] diagnosis is used to find mistakes of students.

Either simple or derived, a very basic source of facts for an assessment is the simulation-based learning context, in which the student works exploratively to acquire knowledge. Obviously, this is a dynamic source where facts vary over time. This source is primarily considered in this work. Other sources can be also taken into account for making assessment.

The time aspect of the simulation-based source is incorporated in the consideration of the two options of assessment, namely, **on-line assessment** and **off-line assessment**. On-line assessment refers to the case that the assessment is done at the run-time of a simulation. The facts, as soon as they occur, can be taken into account for the assessment. In comparison, off-line assessment is made after a simulation execution. Both options might be useful. The former allows the accomplishment of assisting actions being aligned to the simulation execution (see below). The latter can be used to summarize results, and to derive criteria for following simulation executions. Different mechanisms are required to support these options. Few of the current proposals support both of them.

R6: Accomplishment of Assisting Actions

Assisting actions can be either performed directly to a simulation in execution or not. Actions performed directly to a simulation execution are **control actions**. They may occur with the intension of students, but in general without their intervention. For example, a control action will terminate the simulation of a chemical experiment automatically, in case some predefined threshold has been reached. The simulation must allocate appropriate control mechanisms to allow such actions.

Other assisting actions do not affect a simulation execution, as long as the student who is concerned by the actions does not respond by intervening the simulation execution. **Notifications** belong to this category. This is the most frequently used instruction form [NGF96]. Explanation,

for example, is a widely-used type of notification. Taking the example above, in case the threshold has been reached, the student doing the experiment will be notified what is going on. Some suggestions may be also sent to the student, upon which he/she may decide to shut down the simulation, doing something else, or just ignoring. Alternatively, documents about the experiment can be loaded, instead of sending suggestions. This involves another kind of actions.

2.2.4 Summary

Table 2.3 summarizes the requirements R1 to R6 in terms of their coverage for the characteristics of an ideal learning environment as defined by the pedagogical framework CBN-FM.

Dimension	Characteristics	Learning Context	Virtual Teacher
Content	Domain knowledge	R1	
	Heuristic strategies		R4, R5, R6
	Control strategies		R4, R5, R6
	Learning strategies		R4, R5, R6
Methods	Modeling	R1	
	Coaching		R4, R5, R6
	Scaffolding and fading		R4, R5, R6
	Articulation	–	–
	Reflection	R2	
	Exploration	R2	
Sequence	Increasing complexity Increasing diversity Global before local skills	–	–
Sociology	Situated learning	R3	
	Culture of expert practice		R4, R5, R6
	Intrinsic motivation	–	–
	Exploiting cooperation	R3	
	Exploiting competition	R3	

Table 2.3: Requirements compared with CBN-FM

Characteristics that are indicated in the table by ”–” are not covered by the previous consideration, among them are the dimension Sequence that is left for the content specific design of a learning environment, as well as the characteristics intrinsic motivation and articulation that are pedagogical skills hardly replaceable by technologies.

Other characteristics are well-addressed by the established requirements. All the requirements, in particular R4, R5 and R6, must be considered in a balanced manner to create optimal properties for a learning environment. Related contributions to this are reviewed in the following section.

2.3 Related Work

Three blocks of concepts related to this work are reviewed in the following. The first block includes reference models for some general issues in LTS. The second block presents approaches for knowledge-based LTSs, which consider in particular automated performance of teaching strategies. The third block focuses on approaches that have simulation as their core technology.

Concepts reviewed in this section cover the established requirements as a whole. Related works addressing particular aspects are introduced later in Section 4.1, 5.1 and 6.1.

2.3.1 Reference Models

Among different initiatives for e-learning, the IMS Global Learning Consortium and the IEEE Learning Technology Standards Committee (LTSC) are two of the most active ones, who have the goal to provide technical recommendations for e-learning developers and vendors. The specifications are exchanged with other organizations, such as the Aviation Industry CBT Committee (AICC), the ARIADNE Foundation, and the Advanced Distributed Learning Initiative (ADL). For example, the ADL Sharable Content Object Reference Model (SCORM) [Adv01] adopts technologies from IMS, AICC and IEEE LTSC.

In the following, three recommendations from IMS, and a reference architecture from IEEE LTSC are briefly introduced.

IMS Learning Design

This is specified by the IMS Learning Design (IMS-LD) working group, whose mission is to develop ”a framework that supports pedagogical diversity and innovation, while promoting the exchange and interoperability of e-learning materials” [IMS03e].

According to the conceptual model of IMS-LD (shown in Figure 2.3), a learning design is aggregated on the highest abstraction level by the learning objectives, a set of components, and a method. This view can be complemented by two lower abstraction levels.

- A component can be of type *role* (learner or staff), *property group*, *property*, *outcome*, *environment*, *activity structure* or *activity*.
- A *component* can be bound to different types of *resources*, which can be *web content*, *person* or *service facility*.
- The *method* captures the dynamic aspects in a learning design. It is composed of *play*, and optionally *condition*s and *notification*s. *Play* represents the flow of activities during the learning process. Multiple *play*s may exit simultaneously, e.g. to represent activities of different roles. *Condition*s, if existent, are evaluated before activities are performed. A *notification* is carried out in association with a certain activity.

This conceptual model is specified using UML notations [Obj03a]. It is further specified in an information model, in which object attributes are used to describe characteristics of each element

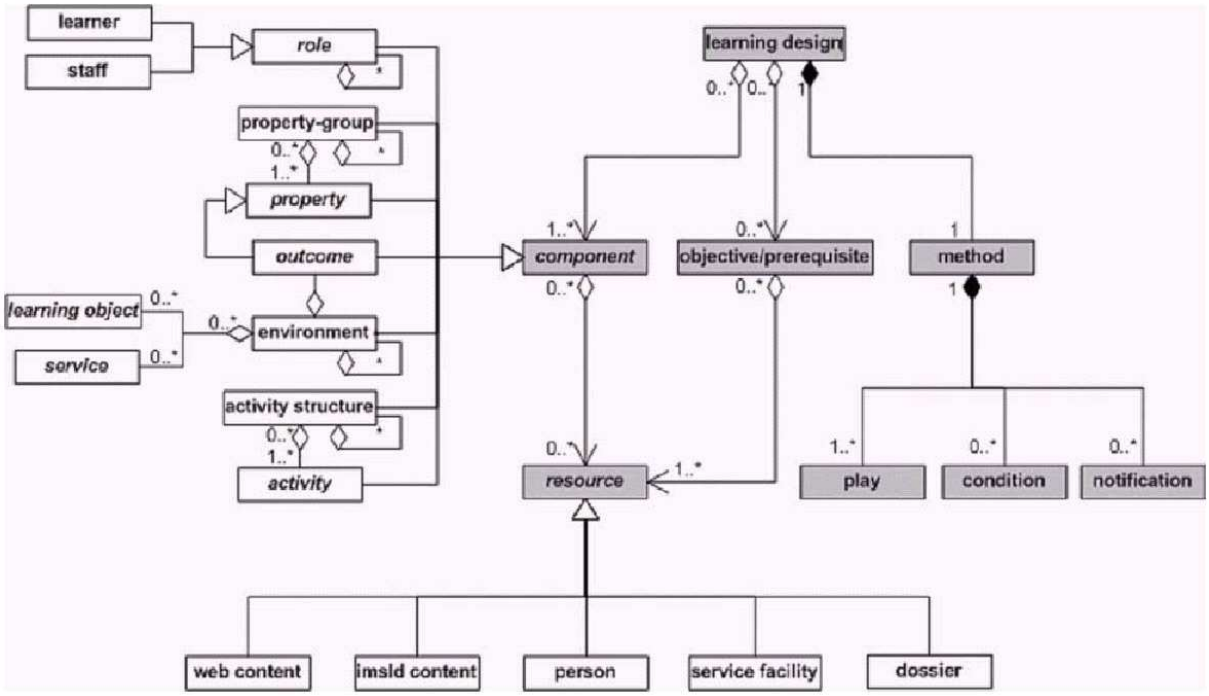

Figure 2.3: Conceptual model of IMS learning design [IMS03e]

in the conceptual model. An XML mapping for the model is provided [IMS03f], together with some best practice examples [IMS03d].

The specification provides guidelines for the high-level design of a learning environment, that may consist of many different types of components. Although only structural concepts are recommended, the method part can be principally extended to describe behavioral details. However, the generation of implementations from such descriptions is left open.

IMS Content Packaging

A learning design can be included in an IMS content package (IMS-CP). The recommendation of IMS-CP supports an approach that significantly forms the evolution of computer-based instruction (CBI) technologies, which is the separation of instructional content from control logic (”evolutionary split” [Adv01]). In contrary to CBI tools in the past, which had monolithic structures, an IMS-CP is composed of reusable resources and adaptable control logic, supported by interchange formats.

Similar to IMS-LD, the IMS-CP specification is also composed of three parts [IMS03b] [IMS03c] [IMS03a]. Figure 2.4 shows the conceptual model of IMS-CP. Each package is a logical directory that consists of a *manifest* and *physical files*. The physical files are actual media elements of different formats, representing instructional content, while the manifest defines the control logic about how these files are used. The manifest is composed of the following elements:

- *Meta-data* that describes the manifest.

- *Organizations* that contain instructions about how to use the package.
- *Resources* that are references to the physical files contained in the package or other external files.
- *(Sub-)Manifest(s)* which are optional, logically nested manifests.

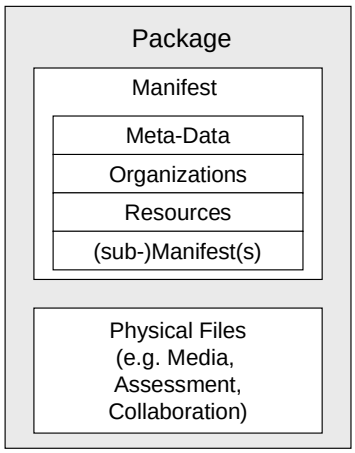

Figure 2.4: IMS content package structure [IMS03b]

”Separating instructional content from control logic” is one of the basic principles followed in this thesis. Content packaging implements this by allocating an Organizations-part in the package, which contains for example instructions of simple sequencing (see below). The instructions must be properly interpreted, in order to control the application of the physical files contained in the package. At this point, the specification remains rather general.

IMS Simple Sequencing

The purpose of the IMS Simple Sequencing Specification (IMS-SEQ) is to define a method to describe learning activities in a consistent way [IMS03g]. A *learning activity* is described by an event or an aggregation of events, whereas an event can be either an instructional event or an event embedded in a content resource. In this manner, a learning content is segmented into pieces, so that a content developer is able to define which piece of content is selected, delivered or skipped for presentation. The content developer is also able to determine the order of the pieces to be presented. Further, learning activities are organized into activity trees. The sequencing behavior navigates through activity trees to fulfil desired learning objectives. A sequencing specification can be included in the Organizations-part of a content package.

This specification is labeled as ”simple” because only a limited number of widely used sequencing behaviors is included. It currently does not address artificial intelligence-based sequencing and sequencing of simulations.

The specification itself is not simple but relative complex. It contains a number of information models and behavior models. The following ones are particularly interesting:

- *Sequencing definition model* is the information model to describe desired sequencing behavior.
- *Tracking model* defines the information model of record information about a learner's interaction with learning activities.
- *Activity state model* is the information model to describe the state or status of a learner.
- *Overall sequencing process* (illustrated in Figure 2.5) describes a logical workflow with involved processes, each of them is described in a separate behavior model. A learner initiates requests on navigation through presented content. These requests are translated by the navigation process into requests to the termination process or the sequencing process. The latter determines what is delivered next to the learner. All these processes rely on the state model describing learning activities.

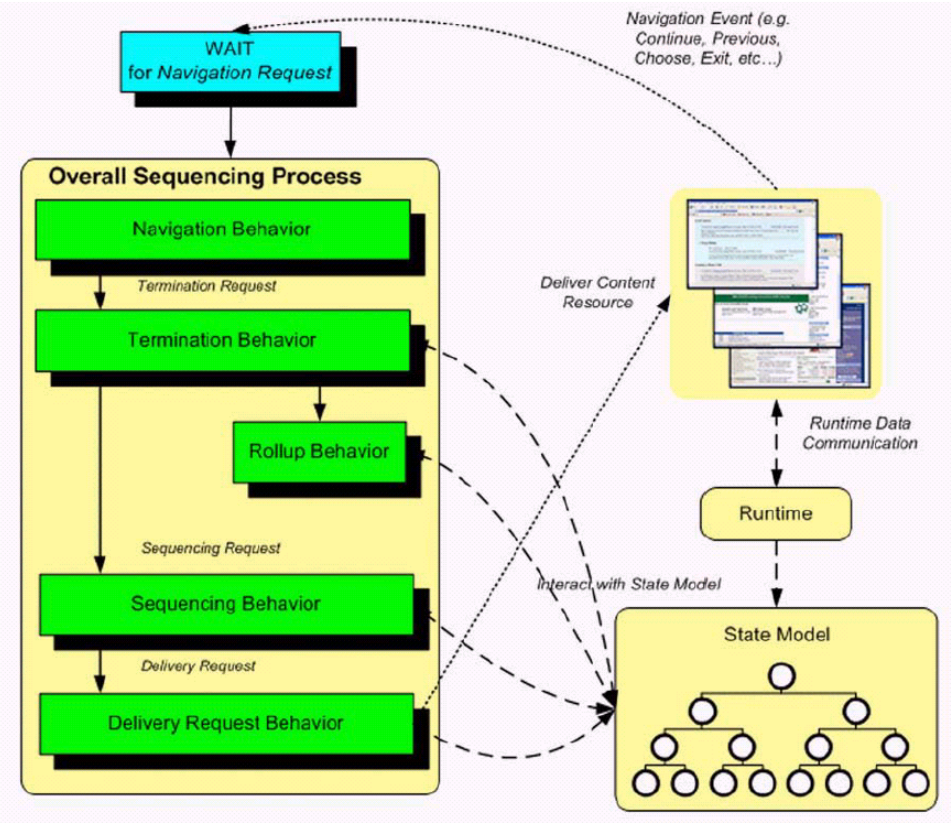

Figure 2.5: IMS overall sequencing process [IMS03g]

This specification provides a detailed modeling of the presentation of learning content consisting of smaller pieces. Interesting is the coupling of the sequencing process with the state model describing learning activities. This approach applies well to static pieces of learning content, which do not vary over time. However, a simulation can be hardly segmented into such pieces.

IEEE Learning Technology Systems Architecture

This architecture (abbr. LTSA) is the basis for other LTSC recommendations. It is a high-level system architecture for LTSs. It was intended as a ”pedagogically neutral, content-neutral, culturally neutral and platform-neutral” architecture [IEE01].

LTSA consists of the following five layers, each representing an abstract view on a learning system:

1. *Learner and environment interactions.* It is the highest layer, and focuses on the interface between a learner or a group of learner and the environment in terms of knowledge discovery, exchange, acquisition, etc.

2. *Learner-related design features.* This layer concerns the nature of a learner as a human, which may affect the lower-level design of a learning system. It is not further considered in LTSA.

3. *System components.* This layer consists of identified components of the architecture. It is the core of the architecture.

4. *Implementation perspective and priorities.* This layer abstracts stakeholders' views on a learning system, each of them concerns a particular aspect that affects the implementation of the system components.

5. *Operational components and interoperability.* It provides codings, APIs (Application Programming Interfaces) and protocols of a learning system.

The system component layer is further described by several processes, data stores and flows, as illustrated in Figure 2.6:

- Processes (4). *Learner entity* is an abstraction of a human learner. *Coach* incorporates information from different sources to select a learning context. *Delivery* transfers learning content to the learning entity. *Evaluation* produces measurements of the learner entity.

- Data stores (2). *Learning resources* include different forms of learning materials, e.g. presentations, tutorials or experiments. *Learner records* store present and past information about learners, e.g. preference and performance.

- Flows (13). Each of them symbolizes the transfer of certain information from one process/store to another. A flow is represented by a labeled, directed arrow. Two types of flows are distinguished: control flow and data flow. A control flow (represented by an arrow with dashed line) starts, stops, or changes processing, while a data flow (represented by an arrow with solid line) refers to inputs and/or outputs of a system or a subsystem. For example, *multimedia* is a data flow representing several types of media delivered from the delivery process to the learner entity process. *Behavior* is a data flow that transfers information about the learner's activities to the evaluation process. *Interaction context* is essential for the evaluation process to understand the learner behavior in case of interactive

media. *Assessment* supplies the coach process with information about the learner's current state. *Locator* points to the learning context to be selected. The locator sent by the delivery process represents a control flow, while the one sent by the coach process contains input data.

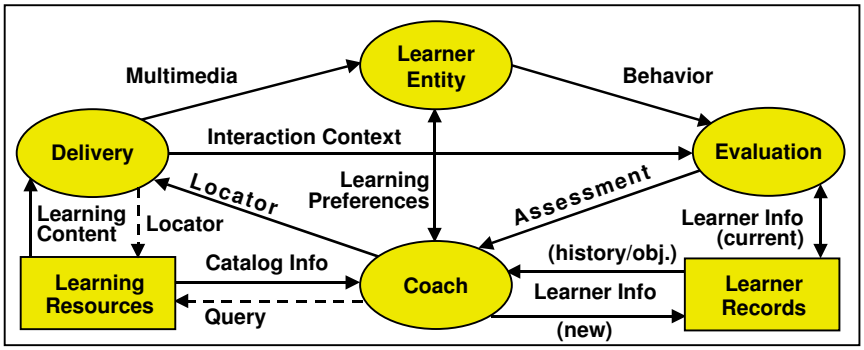

Figure 2.6: System components of LTSA [IEE01]

The processes and flows describe a conceptual view on learning associated activities. Depending on scenarios, the processes can be carried out by different human beings or automated mechanisms, and the flows may have higher or lower priorities. In this sense, a rich variety of stakeholder mappings are provided in the LTSA standard, that covers over forty different known approaches to information technology-supported learning systems. Mappings that are relevant for this work include mentoring, interactive environment, simulation, intelligent tutoring tools and distance learning. The mapping for simulation, for example, places primary the design focus on the flows of multimedia, behavior, interaction context, locator and assessment.

LTSA was intended to cover a broad range of LTSs. The structure of system components addresses the most important processes of an LTS. For e-learning capable simulations, this structure is applicable but not optimal. A slightly different architecture is presented in this thesis, in which for example the delivery process is further refined. Section 3.3 gives an in-depth discussion on this.

2.3.2 Knowledge-Based LTSs

Knowledge-based systems (KBSs), also referred to as expert systems, derive from the discipline of Artificial Intelligent (AI). AI has the aim to emulate human cognitive skills [Jac99]. KBSs focus on the practical application of techniques. They have been applied in education for over 20 years. Due to the lack of powerful hardware and suitable software, earlier computer-aided instruction systems applying KBS-techniques were very limited in supporting individual learner needs [O'S82]. Intelligent systems for education developed later offered greater flexibility at this point [KT92]. Two recent approaches in this area are introduced in the following.

GET-BITS

Generic Tools for Building ITS (GET-BITS) was intended to offer development concepts for Intelligent Tutoring Systems (ITSs) [JD00]. It extends an earlier work named EduSof [JD96]. The motivation of GET-BITS is to provide a highly-usable knowledge acquisition interface to teachers, so that they can create, modify and test teaching concepts without having to be AI specialist themselves.

The problem targeted by GET-BITS is a crucial one for KBSs, namely, ”the transfer and transformation of potential problem-solving expertise from some knowledge source to a program” [Jac99]. This problem can be referred to as a problem of knowledge acquisition. In many knowledge-based ITSs, the technical interface for knowledge acquisition is so complex, that the aid by knowledge engineers is indispensable. Easy-to-use graphical editors are very desirable [Woo92]. A well-founded structure of an ITS is also important to support the representation of broader learning objectives. GET-BITS provides an object-oriented architecture, composed of the following dedicated modules:

- The *domain knowledge module* contains information about the domain knowledge for students to learn. A *knowledge network* connects different frames of learning content semantically together, including lessons, examples, tasks, questions, etc. Text, picture, or simulation can be used as representation forms for these frames.

- The *expert module* contains knowledge about how experts solve problems in the referenced domain. It is used to compare with students' solutions.

- The *interface module* provides mechanisms at the user interface, by which students communicate with the ITS.

- The *tutor module* is responsible to decide how and when the domain knowledge is to be presented to students. It models the teaching process with rules, and uses input from the student model.

- The *diagnostic module* identifies student misunderstanding. In addition, it updates the student model and the teaching rules used by the tutor module.

- The *student model* stores information that is specific to each individual student, in order to determine the state in terms of learning progress.

- The *explanation module* is dedicated to the explanatory knowledge, the third kind of knowledge identified in GET-BITS, additionally to the domain knowledge stored in the domain knowledge module, and the control knowledge represented in the expert module. The explanatory knowledge is not part of the knowledge to be taught to students, but used to control the generation of explanations for the knowledge to be taught.

A key concept of GET-BITS is to use explanation control strategies which are specified in form of if-then rules, e.g.:

IF The explanation type is WHY
THEN Show the current goal

For teachers to create lessons, a set of predefined explanation types is available. They can be bound to content frames.

This approach is interesting, because it addresses in particular the modeling of domain specific expert knowledge and domain general pedagogical knowledge. It explores also the application of explanation control strategies, which correspond to teaching strategies with notification actions. Like IMS-SEQ, knowledge representation frames apply well to static content, but to a simulation only as a whole. The rule-based modeling of strategies is a concept that is also used in the approach introduced below.

ETOILE

Experimental Toolbox for Interactive Learning Environment (ETOILE) is an agent-based approach to building learning environments, in which students are put into problem-solving situations [DHM+93]. The intended learning environments are called "intelligent learning environments", because artificial intelligence techniques are used to implement system components that provide assistance to students. A similar approach is presented in [SSC01]. An instance of such learning environment for experimental psychology, named MEMOLAB, has been developed. The purpose of ETOILE is to support the creative development process for various domains on the same principles of MEMOLAB.

In an ETOILE-based learning environment, students are supported either by human tutors or by computational agents. The latter ones are classified into two types. The one type represents domain experts who have primarily no pedagogical intention but means to optimally solve problems. The other one abstracts domain generic pedagogical knowledge to support the interaction between students and domain experts. Here, a separation between pedagogical knowledge and domain knowledge takes place, in order to allow various teaching strategies being applied to the same learning content. The learner-expert-tutor triangle is the core of the ETOILE architecture.

In respect of the implementation, two points are particularly interesting. One is, that for students to better understand the results, the simulation used in MEMOLAB for psychological experiments produces traces and statistics, whereas, pre-stored comments and keywords are used.

The second point concerns the inference engine of ETOILE. It implements rules that describe expertise of domain-specific and domain-generic agents. It is a general-purpose inference engine using rule representation based on Common LISP, an object-oriented programming language frequently used in AI [Jac99]. A rule is a condition-action expression. The basic rule condition template describes an object-attribute-value vector, that is,

```
(<object>is<class>with
<slot1><operator1><value1>&
..&
<slotn><operatorn><valuen>)
```

where, <slot> is the predicate (attribute of object), and <operator> can be =, <> <, >, <=, or >=. The rule actions include creating/destroying objects, or modifying slot values

of objects. The integration of the inference engine into the learning environment occurs over function calls in rule conditions and actions. In this manner, expert agents are able to diagnose mistakes of a learner based on his/her actions. The inference engine is implemented using a specific interface of the Common LISP tool.

This approach models the interaction between learner, expert and tutor with rule-based agents. Diagnosis and control actions are performed on the basis of learner actions, whereas control actions can be more than explanatory actions, because the rules operate on the object level. Although not clearly stated whether this procedure is integrated with the MEMOLAB simulation, the possibility to open the model of domain knowledge for the instrumentation of control logic is promising. Since function calls used in rule conditions and rule actions are implemented using code-level instrumentations, familiarity with the implementation language is necessary.

2.3.3 Simulation as Core Technology

Simulations are widely used for education. Scientific simulations, such as ns-2 [FV01], DaSSF [LN01] and GTW [DFP+94] provide useful features to be applied to LTSs. However, they cover at most the requirements R1, R2 and R3, as presented in Section 2.2.2. In many other simulation-based software intended as LTSs, only minor run-time teaching support is provided. For example, [FS01] proposes using a benchmarking tool to analyze the results produced by a digital circuit design simulator. In [GZ92], user's actions are compared with predefined mandatory or optional input to a simulation, in order to calculate which tutoring information has to be presented. The following two approaches provide teaching support in a more integrated manner.

HiSAP

Highly interactive Simulation of Algorithms and Protocols (HiSAP) is intended as a framework to generate animation applets for teaching algorithms or protocols from their formal specifications [BR01]. The application of this framework for security protocols is presented in [BP03].

A simulation model relies on an object-oriented concept described by four object classes:

- A *node* represents a communication participant. It is bound to a communication protocol or an algorithm.
- A *connection* links two nodes either in simplex or half duplex mode. Simplex, half duplex and duplex are concepts used for computer networks, describing whether nodes of a network send and receive information at the same time [Tan02a].
- *Message* is used to describe information exchanged over connections.
- *Composed* is used for purposes such as building node hierarchies.

The object-oriented simulation model is based on a formal model that integrates the concept of time. The use of formal methods allows verification of algorithm or protocol specification by automated model checking techniques. It is useful in case students learn to design their own algorithms or protocols.

In examples presented in [BR01], students can be assigned different roles in predefined scenarios to study important cases of an algorithm or a protocol. Intuitive animations are useful for students to learn on their own. The formal specifications allow instrumentation for supplementary tutoring information.

The formal specification-based approach is beneficial for easy creation of simulation models. The simulation object model is well-suited for communication protocols. Tutoring information is directly attached to model descriptions. This is rather a type-level instrumentation. It applies well to study the functionality of algorithms and protocols. Instance-level instrumentation, for example to study a network consisting of many protocol instances of various types, each is configured differently, is not addressed. It is assumed that distributed simulation clients are supported for predefined roles. There is no statement about how they are coordinated over time.

Pedagogical Agents

The work presented in [Joh96] is an agent-based approach. In comparison with ETOILE, it tackles in particular how to structure tutorial interactions within distributed simulation environments. A type of human-like, intelligent agents, called pedagogical agents, is used to provide explanation to learners and to give assessment about their work. Pedagogical agents and the simulation model rely on the Distributed Interactive Simulation (DIS) protocol [Fuj00].

A key concept to support tutoring is plan recognition. It is a student modeling approach. A library of procedural plans is made, with each plan recording the plan steps, together with their pre- and post-conditions. The plans are applied by pedagogical agents to the student behavior, in order to offer assistance in case of impasses, where the student is unable to proceed. This involves monitoring of student actions, and states of simulation objects.

In addition, explanation and debriefing capabilities are provided, which are based on decision analysis, episodic memory and presentation. For decision analysis, hypothetical changes are computed and their outcomes are compared with the outcome of the situation in which the decision was made. Episodic memory records events and the internal changes to recall the situation when each event occurs. Presentation provides explanation to students in appropriate graphical and textual forms.

The functionality of a pedagogical agent is domain-dependent. A pedagogical agent for medical education is presented in [SGJM99] and [SJG99]. There, a rule language is proposed to describe the so-called ”pedagogical opportunities” for situation-based reasoning. The description for a situation that triggers a quiz action is shown below.

(**situation** askUrinalysisQuiz
`:type` quiz
`:condition` (and(order-urinalysis-done == T) (knowsAboutUrinalysis == false))
`:actions` (:quiz urinalysisQuiz)
`:hint` ”Two of these options are appropriate.”
`:rationale` ”The quiz tests your understanding.”)

For the spatial distribution of the collaboration environment, this approach relies on an established protocol for distributed simulation. Issues of the temporal distribution are not addressed.

With plan recognition, assisting actions can be well-embedded into a simulation execution. The rule-based control concept is similar to the approaches GET-BITS and ETOILE.

2.3.4 Summary

Related approaches introduced in the previous two sub-sections are summarized in Table 2.4. They are compared with the requirements on e-learning capable simulations. These approaches are representatives for the very few in the literature that cover more or less the entire range of the established requirements.

In Table 2.4, ”y” denotes that the indicated requirement is supported. ”–” is used if a given feature is not considered by the related work. A feature indicated by ”n” is currently not supported, although it is enclosed in the scope of consideration. Some fields in the summary are described briefly by key words.

Requirements		**GET-BITS**	**ETOILE**	**HiSAP**	**Pedag. Agents**
Knowledge representation		Content frames	Various	Animation applet	Simulation
Interactive usage		y	y	y	y
Distributed collaboration environment	Spatial distribution	–	–	y	y
	Temporal distribution	–	–	–	–
Formalization of teaching strategies	Separable control	y	y	limited	y
	Content development interface	Bind content frames with predefined strategy types	Rule template with function calls	Instrumentation of formal specifications	Rule template
Assessment about students	On-line assessment	y	y	y	y
	Off-line assessment	–	–	–	–
Accomplishment of assisting actions	Control actions	n	y	n	n
	Notifications	y	y	y	y

Table 2.4: Summary of related approaches

Although the approaches GET-BITS and ETOILE do not use simulations as their core technology, all of them have been intended for LTSs. Knowledge representation and interactive usage are well-supported. Various techniques are used for the processing of teaching strategies. The

majority of them use rule-based concepts. All of them support notifications as assisting actions. ETOILE allows also control actions at the expense of programming function calls.

Two features are not addressed at all. One is the temporal distribution of the collaboration environment. The other is the off-line assessment about students. Both are closely related to the time aspect. Not only the facts occurring currently are of interest, but also facts that happened in the past. Incorporating the time concept for the missing features indeed requires consideration of the entire approach. For example, for the temporal distribution, the control of a simulation execution must be able to retain the integrity for the simulation time, in case the simulation is interrupted or repeated. This must be incorporated in the basic simulation control model. Off-line assessment based on facts occurred in the past requires appropriate history storage, which can be also useful for the run-time processing of teaching strategies.

The approach presented in this thesis has the goal to provide the desired features in an integrated manner. For this, it adopts and extends concepts developed in the areas of simulations, databases and knowledge-based systems. An overview of the approach is given in the following chapter.

Chapter 3

Overview of the VTIS Approach

The *Virtual Teaching Interface for e-learning capable Simulations* (VTIS) denotes the approach presented in this thesis. An overview of core concepts of the approach is presented in this chapter. Section 3.1 introduces first the main idea of the approach. Concepts addressing key issues are then summarized in Section 3.2. They are explored in detail in Chapter 4, 5 and 6. The architecture presented in Section 3.3 depicts a technical realization.

3.1 Main Idea

The crucial point to cover the requirements of e-learning capable simulations as a whole is the integration of the processing of teaching strategies with the modeling and the execution of a simulation.

In the reference models of IMS and LTSC, as well as in the related approach GET-BITS, a simulation is treated as a monolithic piece of material to assemble learning content. Comparing with other types of content, such as text, video clips or slides, a simulation with interactive usage may behave differently in different executions. Therefore, a simulation-based learning content involves greater complexity than other types of learning content. On the other side, the behavior of a simulation adheres to the underlying simulation model. Using simulation model as the base of control for teaching strategies is the principle followed by the related approaches ETOILE, HiSAP and Pedagogical Agents.

Generally, two categories of approaches can be distinguished: the anchored approach, and the integration approach. An in-depth discussion of these approaches requires a review of some concepts of formal specification for computer systems. The mathematical notations used below can be found in Appendix A.

Preliminary

To describe computer systems based on grounded mathematical models, a number of formal languages have been developed. Three basic components of a formal language are [Nis99]:

1. *Alphabet*, which is a set of permitted symbols.

2. *Syntax*, which refers to rules defining the right order of symbols in sentences.

3. *Semantics*, which assigns meaning to correctly written sentences.

The alphabet of a specification language usually consists of alphanumeric characters. To formally describe the syntax of such a language, i.e. the grammar, the BNF (Backus-Naur Formalism) notation [ISO96] is frequently used. Defining formal semantics is a crucial task for many specification languages, because only a well-founded semantics allows to unambiguously describe the behavior of a computer system.

A very generic model for semantics specification is a **transition system**. A simple form of transition system is defined as a pair $(\Gamma, \rhd)$ [Plo81] [NN95], where

- Γ is a set of elements γ, called *configurations*, and

- $\rhd \subseteq \Gamma \times \Gamma$ is a binary relation, called *transition relation*.

It states that a transition system steps from one configuration γ to the next configuration γ'.

Widely-used modeling concepts, such as finite automata, context-free grammar, and labeled transition system, can be considered as variants of transition system [Plo81]. Therefore, transition system is an appropriate model for the following discussion concerning the construction of the behavior of an e-learning capable simulation.

Integration Base

The starting point of the discussion is a simulation model, which is free from any teaching-related behavior. The simulation model is described by a transition system, denoted as ts_a, that is,

$$ts_a = (\Gamma_a, \rhd_a)$$

This model is illustrated in Figure 3.1 as a directed graph, likewise for other models in this discussion. A graph can be well applied to illustrate binary relations. Here, nodes of the graph (as unfilled circles) represent the configurations of the transition system, while arrows represent the transition relation.

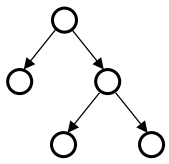

Figure 3.1: Integration base: simulation model as transition system ts_a

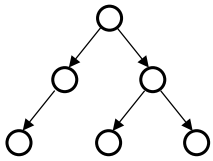

Figure 3.2: Anchored approach as transition system ts_b

Anchored Approach

An anchored approach is not an integration approach, because it involves a completely different model, as illustrated in Figure 3.2.

This approach can be described by a transition system $ts_b = (\Gamma_b, \triangleright_b)$. The coherence to ts_b is difficult to determine. In such a model, the teaching-related part is not separable from the rest of the model. Due to the lack in appropriate supporting software, earlier computer-based instruction tools used to have monolithic structures. They are typical examples for the anchored approach. Obviously, the weakness of this approach is that great effort is likely needed for the adaptation, extension or multi-purpose usage of an existing model.

Behavior-Oriented Integration Approach

Integration approaches based on a simulation model described by ts_a follow the principal of separating learning content from control logic. The modeling of teaching strategies can be seen as an *add-on* to the existing simulation model. They are therefore logically separable from the simulation model. The looser coupling allows flexible adaptation and utilization for multiple teaching purposes.

The integration approaches can be divided into behavior-oriented approaches and data-oriented approaches.

A behavior-oriented approach extends directly the behavior of the original simulation model to implement teaching strategies. This concept is illustrated in Figure 3.3. The extended configuration is represented as a filled circle.

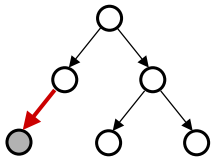

Figure 3.3: Behavior-oriented integration approach as transition system ts_c

A transition system ts_c models this approach as follows, whereas it uses the definition of ts_a to express the coherence to the basic simulation model:

$$ts_c = (\Gamma_c, \triangleright_c)$$

where $\Gamma_a \subseteq \Gamma_c$ and $\triangleright_a \subseteq \triangleright_c$.

ETOILE, HiSAP and Pedagogical Agents are examples for this concept, where an adequate specification interface for teaching strategies is required. ETOILE and Pedagogical Agents use rule templates, while in HiSAP the formal specification of a protocol is extended by instrumentations. Automatism supporting the specification interface is important to relieve content developer from programming.

Data-Oriented Integration Approach

The data-oriented integration approach differs from the behavior-oriented one in the point, that not the behavior of the simulation model but the data reflecting the execution of the simulation model is used to integrate teaching strategies. Figure 3.4 demonstrates this idea.

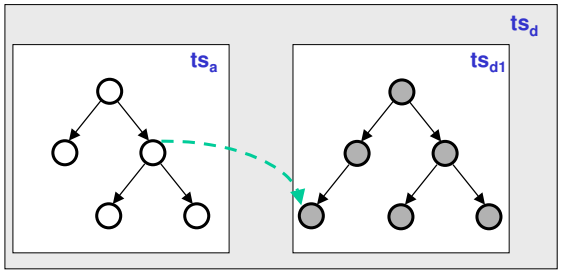

Figure 3.4: Data-oriented integration approach as transition system ts_d

The transition system ts_d is modeled as a combination of the transition systems ts_a and ts_{d1}. This combination can be described formally as the following:

$$ts_d = (ts_a, ts_{d1}, P)$$

where

- ts_a is defined as above,
- $ts_{d1} = (\Gamma_{d1}, \rhd_{d1})$, and
- $P \subseteq \Gamma_a \times \Gamma_{d1}$ describes the relation from the configurations of ts_a to the configurations of ts_{d1}.

This concept describes ts_{d1} that implements the teaching strategies as a separate transition system, and a part that assembles together with ts_a an e-learning capable simulation. The relation P is represented in Figure 3.4 by a single arrow with dashed line from a node of ts_a to a node of ts_{d1}. The dashed line symbolizes a looser coupling between the two transition systems, whose properties are easy to differentiate. By varying the relation P, a new combination can be created with another transition system, say ts_{d2}. In this manner, greater flexibility and extensibility of ts_d can be achieved at lesser expense.

Summary

The main idea of this work follows the data-oriented integration approach. Moreover, it refines the above model of ts_d as ts'_d, by a relation P' that involves the **history** of the configuration of ts_a, that is,

$$ts'_d = (ts_a, ts_{d1}, P')$$

where

- ts_a and ts_{d1} are defined as for ts_d, and
- $P' \subseteq A \times \Gamma_{d1}$ describes the relation from the history of the configurations of ts_a to the configurations of ts_{d1}, where A is a subset of the set of k-fold product of Γ_a, that is, $A \subseteq \{\Gamma_a^k \mid k \in \mathbb{N}\}$.

Concretely, A refers to the current and the past data involved in a simulation execution. The rest of the thesis will elaborate in detail that the data includes not only information about the current and the past states of a simulation, but also information about the user interaction during the simulation execution. The preparation and the processing of the history data to achieve the integration of teaching strategies are the major concerns of the investigation in this thesis.

The key concepts can be sketched using the relation P'. First of all, the relation is built upon a *well-founded simulation model*. With respect to the integration, it is analyzed which properties of the simulation model must be taken into account, and how the properties are described. Two characteristics of an e-learning capable simulation are particularly considered. One is the interactivity of such a simulation to support explorative work. The second is the distributed collaboration environment for students to practice team work.

The second step is to determine how the properties of the simulation model are *collected and conserved as history data*. The time aspect in the history data is reflected in the modeling concept for a temporal simulation database. The structure of the database relies on the analysis in the first step, and can be considered as a representation of A.

The third step is concerned with the right hand side of the relation P', namely the *modeling of teaching strategies* as part of an e-learning capable simulation. Here, the principle of knowledge-based systems is applied. Teaching strategies are formulated as control rules, that operate on the data stored in the temporal simulation database. Using rules that are sensitive to modifications in the database, which are again caused by changes in the simulation execution, teaching strategies can be applied to on-line assessment. Off-line assessment can be achieved in case the rules are used decoupled with a simulation execution.

The following section gives more details to these concepts.

3.2 Key Concepts

The VTIS approach comprises four key concepts, as elaborated in this section.

3.2.1 Interactive Simulation

Interactivity is an essence for e-learning capable simulations. It involves communication between a simulation and its user. A user is able to follow what is going on in a simulation. He/she is also able to have impact on the progress of a simulation. Such simulations are referred to as interactive simulations. In [Fuj00], interactive simulations are also denoted as virtual environments. They are distinguished from analytic simulations, as shown in Table 3.1.

	Analytic simulation	**Interactive simulation/Virtual environment**
Typical objective	Quantitative analysis of complex systems	Creation of a demonstrative representation of a physical system or a process
Execution pacing	As-fast-as-possible	Real-time
Human contact	Generally not included. If included, human is an external observer	Human is integral controller

Table 3.1: Classification of simulations in terms of interactivity

Analytic simulations are typically intended for quantitative analysis of complex systems. They are usually executed as fast as possible. Results are analyzed after the execution. A human is at most an external observer to the simulation. Many known network simulations are analytic simulations, such as DaSSF [LN01], ns [BBE+99] [FV01] and SURGE [BC98].

In comparison to analytic simulations, interactive simulations involve human intervention during the simulation execution. Therefore, this kind of simulations is also referred to as human-in-the-loop simulation. Although real-time execution is conditional for interactive simulations, the timeliness is only as much required as perceptible to human participants. Computer games and aircraft simulators for pilot training are popular examples for interactive simulations. Examples for different other purposes include JANE [PF01], AgentSheets [Rep00], and Ppong! [BS97].

The interactivity of a simulation can be built upon the same core functionality as an analytic simulation. Such core functionality is usually provided by a simulation kernel. In this work, concepts of discrete event-driven simulations and parallel simulations are adapted to form an interactive simulation kernel.

A key issue of the interactive simulation kernel is the co-relation of the **simulation time** and the **wallclock time**. The wallclock time is a reference to the conventional time. The simulation time is an abstraction of the conventional time used in the simulation model. *Scaled real-time execution* is usually used by an interactive simulation kernel, where a **scale factor** that defines the proportion of a second in the simulation time to a second in the wallclock time remains constant for a period of time. The interactivity requires in general the scale factor being set to one. Other values of the scale factor can be also useful where a simulation is comparable to a video player. Fast-forward is for example a frequently used feature of a video player. Section

4.4 gives an in-depth elaboration of the timing mechanisms. Implementation concepts for the interactive simulation kernel are introduced in Section 7.1.

In addition, the modeling concept of the simulation framework SSF (Scalable Simulation Framework) [Cow99] is used for the conceptual simulation model (Section 4.2), from which the temporal data model is derived, as explained in Section 3.2.3. Using the conceptual model, a simulation application can be modeled by *entities*, which may contain sub-entities, and may be connected to each other entities by *ports*. The properties of an entity are described by its *parameters* and *variables*. A parameter can be changed by the user, while a variable is only updated by the simulation model in execution. The conceptual model is intended as an information model describing the structure of a simulation model, because the information model is of primary interest for the database structure. Other behavioral models can be used to describe the behavior of the simulation model. This is however not the scope of this work. The conceptual simulation model is applied to a network simulation as an example (Section 7.2).

3.2.2 Virtual Group Supported by Simulation Session

Students sharing the same learning context over the distributed learning environment build a *virtual group*. A virtual group serves exploration, cooperation and competition. To support virtual groups, two dimensions of distribution must be considered: a spatial dimension and a temporal dimension.

The spatial distribution enables a flexible choice of location for learning. Chat, IP-telephony or shared-white-board are popular but general purpose facilities supporting the spatial distribution. For simulation-based learning, [KT99] presents for example a groupware-based approach. This work is based on a client-server concept. The structure for spatial distributed simulation environments presented in Section 4.3.1 follows the similar principle. A simulation environment consists of a manager and zero or more **simulation environment unit**. Each unit supports one virtual group with one simulation server, and several simulation clients as much as required by members of the virtual group. On the simulation server simulation models representing knowledge to be studied are executed.

The temporal distribution is incorporated in the control mechanism of a simulation server. Temporal distribution allows students in the same virtual group to setup their individual time schedules. Traditionally, for experiments in laboratories, a student group is assigned a fixed time. In order to organize collaborative work between members of a virtual group over time, the concept of **simulation session** is introduced (Section 4.3.2). A simulation session provides a view of control on the simulation server for a continuous period of time. During this time, a member of the virtual group can start, suspend, resume, rewind, or terminate a simulation, change the scale factor to control the speed of the execution, or doing fine-tuning of the simulation. To process user interaction internally, three categories of events are distinguished: **modeled events**, **command events**, and **steering events**. Modeled events are determined by the simulation model. Command events represent requests such as start, suspend, resume, rewind, and terminate. They have no impact on the semantics of modeled events. Opposite to command events, steering events change the future of modeled events. Typical examples of steering events are adding/deleting simulation entities, or changing values of entity parameters. Command events and steering events

together represent the interface for user interaction. However, they are handled differently due to different semantics. Command events are incorporated in the control mechanism of a simulation session, represented by a state machine. Steering events can be derived from the conceptual simulation model as the above examples show.

The control view of a simulation session is built upon the simulation kernel using the concept of **simulation instance**. A simulation instance is an instantiation of a simulation model. As the underlying simulation kernel does not always support backwards execution, the rewind operation requires careful mapping of a simulation session to simulation instances at the time of rewind, because rewind gives the user the impression that the simulation time is set to a time in the past. The control mechanism dealing with this is explained in Section 4.3.2.

3.2.3 Student Assessment Based on Temporal Simulation Database

The basis to apply teaching strategies is the assessment about students. The raw data for the assessment is the history data stored in a temporal simulation database.

This database differs to many other temporal databases in the handling of the simulation time and the wallclock time. Conventional temporal databases support the time dimensions valid time and/or transaction time. Valid time refers to the time of the fact in reality, relative to the modeled world in the database. Transaction time is the time when the fact is presented to the database. For interactive simulations, the reality exists in the simulation time, and in the wallclock time. Therefore, both time dimensions are used by time stamps for the history data.

The temporal data model as presented in Section 5.2 uses the **lifespan** concept to express the relations between elements in the database in one or the other time dimension. The structure of the data model is aligned to the conceptual simulation model. In addition to the entities according to the conceptual simulation model, simulation sessions and simulation instances are also incorporated in this structure.

To support the development of concrete database schemas, a representational, generic data model is specified using the metamodeling language MOF (Meta Object Facility) [Obj02b]. MOF supports different language mappings to generate schemas. XML Schema is used in this work as the concrete schema specification language [Wor01a]. The complete specification of the metamodel and the generic schema are included in Appendix B and C.

3.2.4 Knowledge-Based Processing of Teaching Strategies

The processing of teaching strategies is based on the principle of knowledge-based systems (KBSs). An introduction to KBSs is give in Section 6.1.1. The structure of KBSs can be considered from the view of a user, a knowledge engineer, or a tool developer [GD93]. These views can be mapped to three LTS-related roles, as shown in Table 3.2. The student role corresponds to the user's view. Thereby, a teacher as supporter is not significant in this consideration, because in terms of a KBS a teacher acts desirably as a knowledge engineer.

Knowledge acquisition is a critical issue for KBSs. Many KBSs have complex knowledge modeling interfaces, so that a knowledge owner cannot play the role of a knowledge engineer at the same time. Abstract description has been found useful to support a teacher as a content

LTS role	Views of knowledge-based systems
Student	User
Teacher as content developer	Knowledge engineer
Tool developer	Tool developer

Table 3.2: LTS-related roles and views of knowledge-based systems

developer, as shown in the related approaches GET-BITS, ETOILE and Pedagogical Agents. Expression power, compactness and simplicity are factors to be taken into account for the description technique.

Active databases can be considered as a special type of KBSs, that respond automatically to certain events occurring inside or outside the databases. Due to the fact, that a temporal database deliveries input data for teaching strategies, concepts of active databases apply well to this work. Section 6.2 presents a knowledge model for **active rules**, which are in general event-condition-action rules. According to the temporal data model, insertion and update of database elements are events triggering the evaluation of the condition part of active rules. The condition part consists of queries on the history data stored in the database. This part is supported by a set of typical query use cases (Section 6.2).

The rule template (Section 6.3) is based on the grammar of XQuery [Wor03c]. It is a query language using XML Schema as type system. This rule template is well-adapted to the generic data schema of the temporal simulation database. An architecture for the processing of rules is presented in Section 6.4, as a refinement of the overall architecture introduced in the following section.

3.3 Architecture

The VTIS overall architecture, as illustrated in Figure 3.5, is a component-based architecture. It assembles two dedicated views, each of them is supported by a set of system components.

3.3.1 Dedicated Views

The **student's view** is represented by a virtual teaching interface, at which a student obtains assistance while he/she works interactively with a simulation. Conceptually, the virtual teaching interface comprises a simulation client interface and an assistance interface.

The simulation client interface communicates with the simulation server to obtain information about the simulation progress, and to transfer student requests to the simulation. The construction of the simulation server and simulation clients is elaborated in Chapter 4. Some implementation details, including examples, are presented in Section 7.1 and Section 7.2.

The assistance interface supports the operation of teaching strategies evaluated by the controller. The controller uses teaching strategies stored in the rule repository to process history data provided by the simulation database. It delivers notifications to the assistance interface, and/or

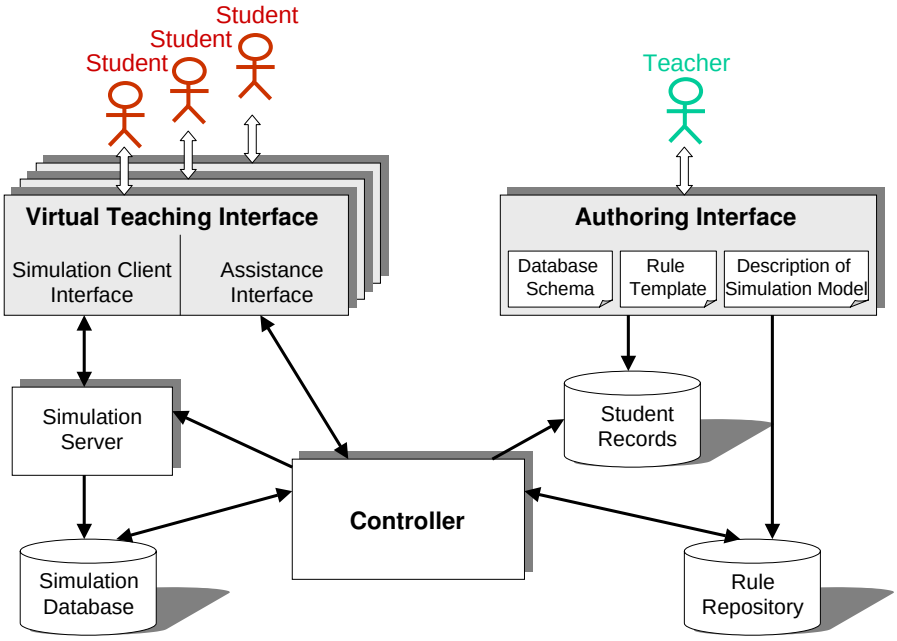

Figure 3.5: VTIS overall architecture

stores information about students in the student records. The realization of the database, aligned to the data model introduced in Chapter 5, is discussed in Section 7.3.

The **teacher's view** is concerned with an authoring interface. Using a database schema that provides structural information about the simulation database, a rule template for the specification of teaching strategies, and a description of the simulation model, a teacher supplies his/her knowledge to the rule repository that is used by the controller to guide the student's work. Components for the rule processing are explored in Section 6.4 and Section 7.4.

3.3.2 Comparison with LTSA

The LTSA architecture is very general to various types of LTSs, while the VTIS architecture is intended for e-learning capable simulations based on the data-oriented integration concept. The correlation between the two are sketched in the following (see also Table 3.3):

- The LTSA learner entity process represents VTIS student.
- The LTSA delivery process corresponds to the VTIS simulation client interface, the simulation server, and the assistance interface.
- The LTSA evaluation process and coach process aggregate the VTIS controller that computes assessment about student and performs notification or control actions. Whereby, the functionality of the VTIS controller is determined by the VTIS rule repository.
- The LTSA learning resources are represented by the VTIS simulation server.

- The LTSA learner records are comparable to the VTIS student records.

Obviously, despite of conceptual correspondence, an exact componentwise mapping is difficult. The main reason consists in the data-oriented concept of the VTIS architecture. At on point, besides the delivery of learning content, data is prepared for the evaluation about the learner. In LTSA, learner information is stored in learner records. It is updated by the evaluation process on observation of the learner behavior, and utilized by the coach process to locate content to deliver. According to VTIS, the information about the learner's behavior is reflected in the history data stored in the simulation database. On examination of the data, the controller decides whether the student records should be updated, control actions on the simulation server should be performed, or notifications to the assistance interface should be delivered. The second point is that the VTIS architecture includes also an authoring concept supporting the development of learning content. LTSA does not have correspondence to this part.

LTSA Components	**VTIS Components**
Learner entity process	Student
Delivery process	Simulation server, simulation client, assistance interfacde
Evaluation process, coach process	Controller
Learning resources	Simulation server
Learner records	Student records
–	Authoring interface
–	Teacher
–	Rule repository
–	Simulation database

Table 3.3: Comparing LTSA and VTIS components

3.4 Summary

This chapter introduces the main idea and key concepts of the approach named VTIS. The basic principle of VTIS is a data-oriented integration concept that is described using the formalism of transition systems. The greater flexibility of an e-learning capable simulation is achieved by a looser coupling of teaching strategies with simulation models over the manipulation of simulation history data. This is supported by four key concepts of the VTIS approach.

Table 3.4 gives a summary of these concepts as a guidance for the following chapters. Like Table 2.4, it uses the established requirements as criteria. Section numbers indicate where a particular feature is discussed in the following chapters.

Requirements		**Interactive simulation**	**Virtual group**	**Temporal database**	**Active rules**
Knowledge representation		Sec. 4.2			
Interactive usage		Sec. 4.3, 4.4			
Distributed collaboration environment	Spatial distribution		Sec. 4.3.1		
	Temporal distribution		Sec. 4.3.2		
Formalization of teaching strategies	Separable control			Sec. 5.2	Sec. 6.2
	Content development interface			Sec. 5.3, 5.4	Sec. 6.3
Assessment about students	On-line assessment			Sec. 5.2, 5.3, 5.4	Sec. 6.2, 6.3, 6.4
	Off-line assessment			Sec. 5.2, 5.3, 5.4	Sec. 6.2, 6.3, 6.4
Accomplishment of assisting actions	Control actions				Sec. 6.2, 6.3, 6.4
	Notifications				Sec. 6.2, 6.3, 6.4

Table 3.4: Summary of VTIS concepts

Chapter 4

Interactive Simulation for Virtual User Groups

This section focuses on the key concepts of VTIS is concerned with interactive simulation and the support for virtual user groups. It is noted, that in this context, ”user group” instead of ”student group” is used, because at this point no difference is made between a student as a group member, and a teacher as a group member.

Basic simulation concepts are briefly introduced in Section 4.1. In Section 4.2, a conceptual model for the construction of simulation applications is presented. The architectural and structural concepts for the management of the spatial and temporal distribution of virtual user groups are elaborated in Section 4.3. The timing and control issues of an interactive simulation kernel are discussed in Section 4.4.

The concepts are based on a joined work with RWTH-Aachen and Fraunhofer FOKUS. The following presentation focuses on issues related to the VTIS approach. For other details please refer to [LLPM+03b] and [PRS+04].

4.1 Basic Concepts and Related Work

Terms such as simulation, simulation model or simulation state have been already used in the previous discussion. Their formal definitions are now provided in the following, whereas concepts from [BINN01], [CL99], and [Fuj00] are adopted.

System

System is one of those primitive concepts, for which there exists a common understanding, and numerous definitions. The following one is used in this work:

> A system is a group of objects that are joined together in some regular interaction or interdependence toward to accomplishment of some purpose [BINN01].

According to this, an **entity** is an object of interest in the system, and an **attribute** is a property of an entity.

Applying the input-output-modeling [CL99], a system can be described by input variables, represented by the vector $\mathbf{u}(t)$ ($[...]^T$ denotes the transpose of a vector):

$$\mathbf{u}(t) = [u_1(t), ..., u_p(t)]^T, t_0 \leq t \leq t_f$$

and output variables, represented by the vector $\mathbf{y}(t)$:

$$\mathbf{y}(t) = [y_1(t), ..., y_m(t)]^T, t_0 \leq t \leq t_f$$

where each variable is represented as a time function over a period of time, denoted by $[t_0, t_f]$.

A **static system** is one where the output $\mathbf{y}(t)$ is independent of past values of the input $\mathbf{u}(\tau)$, $\tau < t$ for all *t*. Otherwise, it is a **dynamic system**. A **time-invariant dynamic system**, sometimes called **stationary system**, has the following property: if an input $\mathbf{u}(t)$ results in an output $\mathbf{y}(t)$, then the input $\mathbf{u}(t-\tau)$ results in the output $\mathbf{y}(t-\tau)$, for any $\tau < t$. A dynamic system not obeying this rule is called a **time-varying dynamic system**.

Roughly speaking, the **state** of a system describes its behavior at a time instant in some measurable way. Like input and output, the state of a system is described by state variables, represented by the vector $\mathbf{x}(t)$:

$$\mathbf{x}(t) = [x_1(t), ..., x_n(t)]^T, t_0 \leq t \leq t_f$$

The state ***x(t$_0$)*** is the information required at time t_0 such that the output $\mathbf{y}(t)$, for all $t \geq t_0$, is uniquely determined from this information and from $\mathbf{u}(t)$, $t \geq t_0$. The **state space**, denoted by X, is the set of all possible values that the state may take. In a **continuous-state system**, as illustrated in Figure 4.1, the state variable $\mathbf{x}(t)$ changes continuously over time.

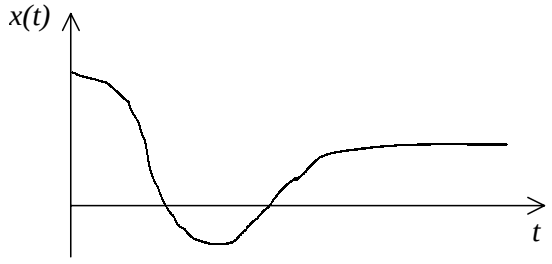

Figure 4.1: Continuous state space and continuous time

In a **deterministic system**, for a given set of input variables $\mathbf{u}(t)$, $t \geq t_0$, there is a unique set of output variables $\mathbf{y}(t)$, $t \geq t_0$. A system is a **stochastic system** if at least one of its output variables is a random variable, which is determined probabilistically.

In the strict sense, two notions of discretion are distinguished: discrete state and discrete time. In a **discrete-state system**, opposite to a continuous-state system, the state space is a set of discrete state values. In a **discrete-time system**, the time consists of a sequence of single points $t_0 < t_1 < ... < t_k < ...$, for $k \in \mathbb{N}_0$, with equal-length intervals between them, where $\mathbb{N}_0$ represents the set of natural numbers including zero. Discrete time does not imply discrete space, neither vice versa.

A **discrete-event system** is a discrete-space, **event-driven system**. **Event** is a concept, like system, that can be well-understood intuitively. It can be thought of something that occurs instantaneously. In an event-driven system, state transitions are triggered by events. It has therefore a discrete state space. An alternative to an event-driven system is a **time-driven system**, in which state may change at any point in time. A continuous-state system is by its nature a time-driven system. The discrete-state property of a discrete-event system is illustrated in Figure 4.2. The state space X is a set of states, which is in this example $X = \{s_1, s_2, s_3, s_4\}$. The state changes when an event occurs, e.g. event e_1 at time t_1.

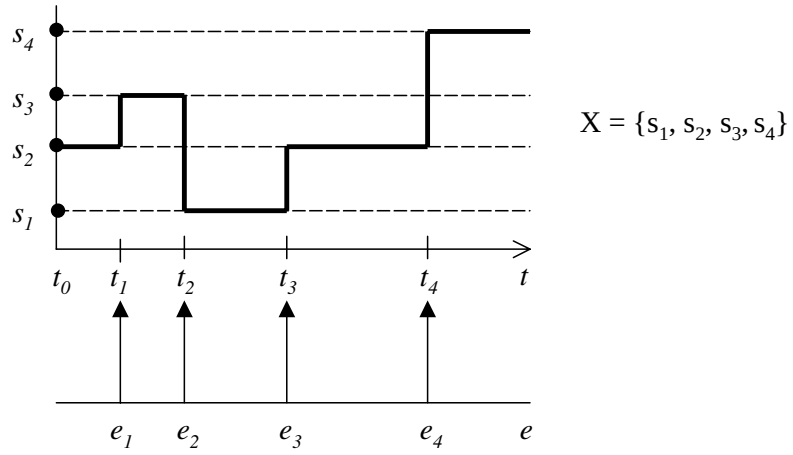

Figure 4.2: Discrete-event system with discrete state space ([CL99])

Simulation

A **simulation** is the imitation of the operation of a real-world process or system over time [BINN01]. The heart of a simulation is its simulation model. A **simulation model** is built upon assumptions about the operation of the process or system to be simulated, which are expressed in mathematical, logical, and/or symbolic relationships.

The following three notions of time are frequently used in the study of simulation [Fuj00]:

- **Physical time** refers to time in the physical system. It is the time in the conventional sense.
- **Simulation time** is an abstraction used by the simulation to model the physical time.
- **Wallclock time** is a reference to physical time obtained by reading a hardware clock.

Two further time-related concepts are activity and delay. An **activity** is defined as a duration of time of specified length, which is known when it begins. A **delay** is a duration of time of unspecified indefinite length, which is not known until it ends.

The research on simulation spreads over a broad spectrum. Some related concepts for this work are introduced in the following.

4.1.1 Discrete-Event Simulation

Depending on how the state changes as the simulation time advances, simulations can be classified into continuous or discrete simulations. A **continuous simulation** is one, in which the state changes continuously over time. In a **discrete simulation**, the state changes only at a discrete set of points in time. By its nature as a discrete system, a discrete simulation can be either time-driven or event-driven. A **discrete-event simulation** is a discrete-event system, in which state changes are determined by the occurrence of events and activities. **Time stamps** associated with events and activities set the points in time of their occurrence.

Three prevalent modeling concepts of discrete simulation, called **world views** (or schemes [CL99]), are the event-scheduling world view, the process-interaction world view, and the activity-scanning world view. The **event-scheduling world view** involves processing of the Future Event List (FEL), which contains events, together with the time when the events should occur in the future. The FEL is ordered with the smallest event time first. The order is retained when an event is removed from the list, or when a new event is added to the list. The scheduling algorithm makes sure that the first event is fetched and the simulation time is advanced to the time associated with this event.

According to the **process-interaction world view**, a simulation is modeled with entities and processes. It is in particular intuitive for a high-level modeling. Entities are consumers of system resources. The life-cycle of entities is described by processes, each of them is described by time-sequenced events, activities and delays. Processes interact in terms of resource demanding. The implementation of the process-interaction approach, hidden from a modeler's view, often makes use of the event-scheduling algorithm.

In comparison to the event-scheduling world view and the process-interaction world view, both use variable time advance (i.e. time advances to the next imminent event), the **activity-scanning world view** uses fixed time increment and is therefore a time-driven concept. At each time increment, conditions for activities are checked to determine which activities should begin.

4.1.2 Parallel Simulation

Scalability and performance are crucial aspects for simulations. With the increasing complexity in the simulation model, sequential simulations encounter more difficulties in execution time and memory space. **Parallel discrete-event simulation** (PDES) addresses these issues by modeling a simulation into potential parallel *logical process*es (LPs) . Each LP executes its local events. In case that an LP generates new events which affect other LPs, it notifies those LPs by passing *message*s to them. By exploiting the parallelism in the simulation model, a PDES can be executed on shared-memory or distributed memory computers. Therefore, PDES is in particular interesting for large-scale simulation models. [Fuj00] provides a comprehensive survey of the field of PDES. An overview on PDES tools is given in [LLC+99].

A main issue in achieving parallelism is the *causality constraint*. In a discrete-event simulation, an event with the smallest time stamp is processed first. All events should be processed in the time stamp order to ensure the causality. This can be easily achieved in a sequential simulation, where all events are centrally managed in an event list. In PDES, each LP processes only

local events on its own event list. Due to the parallelism of LPs and the lack of a global clock, the synchronization issue arises. This is a well-studied problem in distributed systems, to which causal ordering generally applies. Causal ordering is defined by the Lamport's *happened-before* relation [CDK01] [Tan02b], denoted by "$\rightarrow$". For any message *m* exchanged between LPs, *send(m)* $\rightarrow$ *receive(m)*, taking *send(m)* as the event of sending of the message, and *receive(m)* as the event of receiving the message.

In terms of algorithms to ensure causal ordering, two categories of PDES are distinguished, namely the conservative approach and the optimistic approach.

The **conservative** approach is one to prevent causality error from ever occurring. The local event processing is blocked until it is ensured that the next event is safe from future event arrivals with smaller time stamps caused by other LPs. The *lookahead* problem is a crucial issue for the performance of this approach, which refers to the ability to predict what will happen, or will not happen in the simulated future. Model characteristics can be used to achieve good lookahead. For example, LPs can be modeled by FIFO (first-in-first-out) job queues. They communicate the potential points in simulation time when a message may be sent to others. It is safe to process local events until those indicated points in simulation time [Liu03].

In contrary to the conservative approach, the **optimistic** approach allows events to be processed out of order. Once a causality error is detected, *roll-back* to the state before the occurrence of the error is performed. State saving and recovery mechanisms are essential for this approach. A well-known optimistic algorithm is Jefferson's Time Warp [Jef85], where incoming and outgoing messages are held in separate queues. At the time when a *straggler event* arrives, which denotes an event with a smaller time stamp than the current local simulation clock, *anti-messages* are generated and stored in the outgoing queue, one for each of the messages already sent after the time of the straggler event. Anti-messages are delivered to their destinations to force *roll-back* of earlier states. The global control mechanism to ensure roll-back is based on the concept of Global Virtual Time (GVT). Each LP holds a local virtual clock. GVT at real time *r* (time in the conventional sense, or physical time) is defined as the minimum of the virtual times included in two sets: one contains all virtual times at local virtual clocks, the other consists of the virtual send times of all messages that have been sent but not yet been processed at time *r*. GVT never decreases, while individual local virtual clocks may roll back. Any event with virtual time less than GVT can not be rolled back.

Different algorithms have been developed for both approaches. Some combine the concepts of both [NJY97]. Whether a conservative approach or an optimistic approach is more appropriate depends very much on the characteristics of the application being simulated. Generally, a conservative simulation is easier to implement than an optimistic simulation, because the state saving and recovery mechanisms for an optimistic simulation are more complex for programming and debugging. Regarding the performance, a conservative simulation may be sometimes over pessimistic about the parallelism in a model due to the lack of good lookahead, and overhead for roll-back mechanisms is in general required for an optimistic simulation. For simulations of computer networks, the conservative approach as well as the optimistic approach have been used [LPN$^+$01] [PF01].

4.1.3 Real-Time Simulation and Interactive Simulation

Real-time simulation execution is the prerequisite for interactive simulations (more see below) and emulations. Interactive simulations are sometimes called human-in-the-loop simulations, because human intervention is allowed during simulation execution. Emulations, on the other side, interact with real systems, rather than with human beings. For example, network emulators [SBU00] [BBE+99] are embedded in the communication with other network nodes.

According to [Jen], the definition of real-time involves two fundamental factors: time constraints on individual activities, and criteria for using those time constraints to achieve system timeliness. The most familiar example of time constraint is a *deadline*, which is defined as "a time until which the activity's completion has more utility to the system, and after which the activity's completion has less utility" [Jen]. For a *hard deadline*, the utility can be considered as a binary set $\{1,0\}$: utility 1 for activity's completion before deadline, otherwise utility 0. *Soft deadline* is a general case of deadline, where utility is measured in terms of lateness (completion time minus deadline). Larger positive values of lateness represent lower utility. *Hard real-time systems* require that all hard deadlines must be absolutely fulfilled, while *soft real-time systems* handle soft deadlines.

A **real-time simulation** demands the execution of a simulation being paced by the wallclock time. For time-driven discrete simulations, the selection of appropriate clock rate is crucial [KLIK99]. An option to achieve real-time execution in an event-driven simulation is, that the simulation time remains unchanged until the wallclock time reaches the time when the next event should be fired. The simulation time advances directly to the time indicated in the time stamp of the event. This occurs under the constraint of the computation time, because it is easier to speed-up a simulation than to slow it down. Setting the proportion between the simulation time and the wallclock time appropriately, the event-driven approach can be used for simulation execution with variable speed, including real-time execution.

The most PDESs are intended to achieve rapid simulation execution. They are however not excluded for real-time execution. Proper adaptation must in particular take care of synchronization mechanisms. A few optimistic simulation-based approaches can be found in the literature, for example [FGUC97] and [PF01], which are basically intended for interactive simulations.

Comparing to real-time simulation, **interactive simulation** requires additional handling of interactions with external entities, including delivery of information about the simulation progress, and implementation of user requests that probably affect the simulation progress.

The approach presented in [FGUC97] provides interactions with output primitives and input primitives. The output primitives are intended for sampling, notification and queries, all concerned with state information either at a point of, or over a range of the virtual time. Whereas, only valid state information is delivered. For this, the notion of World Virtual Time (WVT) is introduced, which is the virtual time of the last valid simulation state presented to external entities. Only state prior to GVT is considered valid. Therefore, it follows that WVT $\leq$ GVT. [PF01] is based on a similar concept. Input to a simulation is modeled by injection of steering events or re-execution events. Steering events guide the future simulation progress. Re-execution events cause a simulation be rewound. In the latter case, Time Warp state saving and roll-back mechanisms are utilized.

4.1.4 Client-Server Simulation Architecture

For interactive simulations involving multiple users, the client-server paradigm is considered appropriate [KT98] [PF01]. The distribution of simulation clients is necessary to support interactions initiated by users individually and concurrently. This however does not require necessarily the distribution of a simulation server. Two categories of server architectures can be identified:

- **Centralized server architecture** is one, where a single server maintains the shared state of a simulation. Clients, distributed at users' site, may perform computations pertaining only to local entities. The clients are primarily to display the shared simulation state to the user, and to deliver user requests to the server.
- **Distributed server architecture** exploits multiple servers, or more precisely, server parts, to compute the shared simulation state. Clients are aware of the distribution of the server, and handle user interactions accordingly.

This classification is different to [Fuj00], in which a third category of architecture is added to the above two, namely the serverless architecture. Here, the serverless architecture is considered to be an extreme of the distributed server architecture, where the server parts are distributed along with clients. Neither the number of processors utilized by the server, nor the type of network connecting the processors, but the awareness of the server distribution from the client point of view, is significant for the classification. A centralized server may use shared memory or distributed memory processors, as long as it is perceived by clients as a single server. In other words, a client-server architecture may be mapped to different operational models.

In the literature, the term *distributed simulation* has been assigned different meanings. In this document, the context of distribution is explicitly stated to avoid misunderstanding, e.g. simulation execution on distributed memory processors, or distribution of a simulation user environment.

The High Level Architecture (HLA) [Def96] is an approach to distributed server architecture. The development of HLA has been initiated and organized by the Department of Defense (DoD) of the US government. It extends an earlier work named Distributed Interactive Simulation (DIS) [DIS94], also supported by DoD. A HLA-based simulation is a federation of autonomous simulation nodes, called federates, that exchange messages to notify state changes on individual nodes. To support the communication between federates of different types, HLA provides a Run-Time Interface (RTI) with an object-oriented API (Application Programming Interface). This API has been adopted by the OMG Distributed Simulation Systems (DSS) Specification [Obj00]. It comprises federation management, time management, ownership management, data distribution management etc. It addresses also complex issues concerning the synchronization between federates.

4.1.5 Simulation Application Modeling

The separation of simulation application from simulation kernel, as illustrated in Figure 4.3, is helpful for application development to be abstract from basic simulation mechanisms, such

as event handling and synchronization, and to concentrate on application specific properties. Existing simulation application modeling concepts differ from each other due to various factors. Domain dependency and coupling to simulation kernel are two of the most significant factors. For example, the modeling concepts for network simulations provided by GTW [FDP+97], SimKit [Pro95], OMNeT++ [Var02], and ns-2 [FV01] are closely coupled with the kernel.

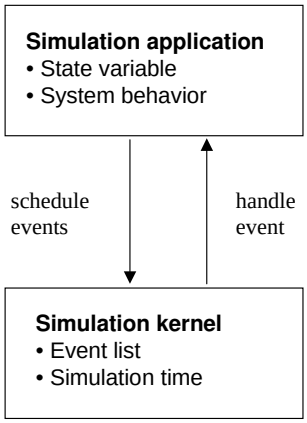

Figure 4.3: Simulation kernel and simulation application

The developers of the Scalable Simulation Framework (SSF) [Cow99] put their effort in a modeling concept that is applicable for different kernel implementation approaches of discrete-event simulation. SSF was intended as a generic framework for various domains as well, although the majority of the SSF-based applications are so far network simulations. The design goals of SSF include high performance, scalability and simplicity. The core of SSF is an object-oriented API consisting of five base classes: *entity*, *process*, *event*, *in-channel* and *out-channel* (see also Figure 4.4).

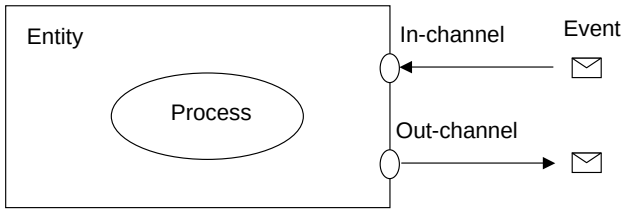

Figure 4.4: Scalable Simulation Framework (SSF)

Entity is the basic building block of a simulation model. An entity serves as a container for processes and state variables, and owns channel endpoints to interact with other entities. Entities may act independently, or build alignment groups. The co-alignment of entities is significant for sharing the same timeline in terms of virtual time.

Process describes the state evolution of an entity. Here, SSF adopts the process-oriented world view. An entity may have multiple processes. A process is typically triggered by the

arrival of events from associated in-channels, or by the expiration of a certain amount of simulation time.

In-channel represents the in-coming communication endpoints of an entity. An in-channel is connected with an out-channel.

Out-channel is the counterpart of in-channel. Delay can be specified for an out-channel to model the duration of an event delivery. The separation of in-channel from out-channel serves the modeling of uni-directional communication.

Event is the base class for messages exchanged over links between entities modeled by in-channel and out-channel.

The broad range of simulation concepts introduced in this section are adopted and refined for the VTIS approach. One point concerns a conceptual model for simulation applications, as presented in Section 4.2. It is an extension of the simulation framework SSF. A second point revolves a centralized simulation server architecture to support the management of virtual groups. It is discussed in Section 4.3. Techniques of discrete-event simulations and parallel simulations are applied to the implementation of the interactive simulation kernel. Details to this are presented in Section 7.1. In addition, particular issues addressing the timing and control for the temporal distribution of virtual groups are extensively elaborated in Section 4.4.

4.2 Conceptual Model for Simulation Applications

The focus of this section is a conceptual model for simulation applications. It is based on the application modeling framework SSF. This conceptual model is primarily intended as a basis for the VTIS database model presented in Section 5.2. Hence, it is rather an information model according to [Dis99], or a static model after the terminology in [DW98], in the sense that it describes the generic structure of a simulation application, including the core elements, their basic properties, and the relationships between them.

A behavior model or an interaction model can be used to complement the structural specification in the static model. Their consideration is out of scope of this work, because the goal of the conceptual model is towards a database specification, for which the structural information is of primary interest. The conceptual model is inspired by the entity-based framework of SSF, but differs to it due to the focus on structural information. As an abstract model, it can be mapped to SSF and to other similar modeling concepts. The conceptual model is an object-oriented model, specified using UML [Obj03a] [HK03]. Figure 4.5 shows the graphical representation of this model. The major part of this model has been presented in [LLP02] and [LLPM+03b].

The model consists of the following classes, defined in the namespace `vip`:

DataType is included for the completeness of the model. It represents predefined primitive and structured data types.

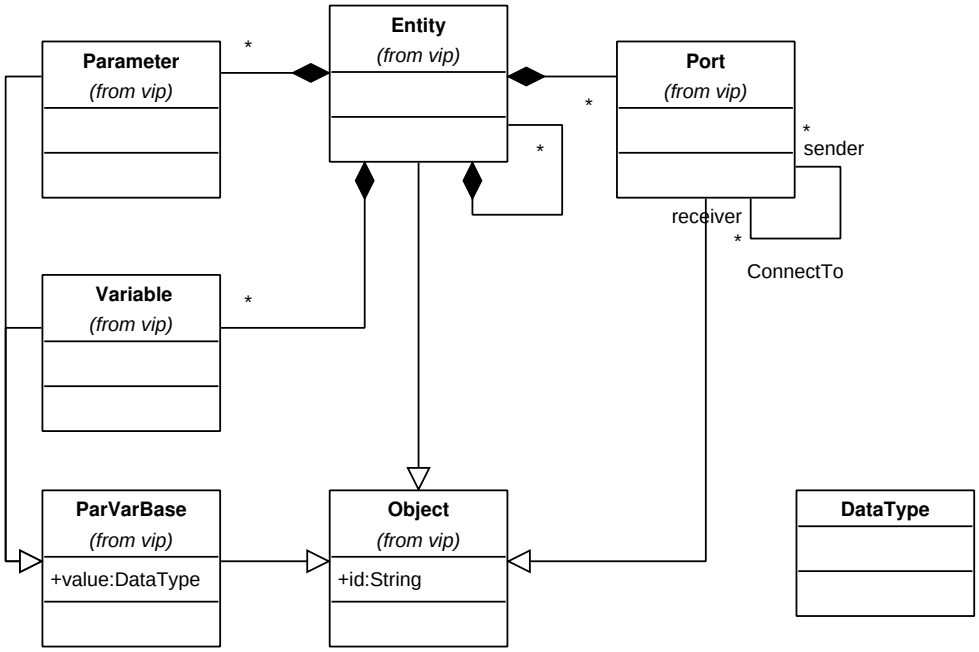

Figure 4.5: Conceptual model for simulation application

Object is the base class for all objects that have an identification (id).

Entity describes a system entity. It is the elementary construct in a simulation model. The properties of an Entity is not modeled by processes, as e.g. in SSF (see Section 4.1.4), but by sub-Entities, Parameters, Variables and Ports.

Port is used to specify communication ends of Entity. A Port is contained in an Entity. A Port may be connected to other Ports as sender, receiver or both. This relationship is specified by the ConnectTo association (see below). A Port is by default bi-directional, and can be therefore mapped to a pair of SSF in-channel and out-channel. A uni-directional Port can be derived from Port with additional constraining attributes. The naming and background of Port rely on computer networks [Tan02a], for which numerous simulation concepts and techniques have been developed, for example GTW [FDP+97], SimKit [Pro95] and OMNeT++ [Var02]. However, Port in this model is not limited to a physical connecting point of a network equipment such as a switch, but allows also modeling of abstract communication endpoints, e.g. a service access point (SAP) of a communication protocol. In summary, Port is an abstraction of communication ends of Entities, and is therefore applicable to any systems that can be described by interacting entities.

ParVarBase serves as the base class for Parameter and Variable. It has a value attribute, which can be of any type defined by DataType.

Parameter is used to configure a simulation model, either in the initialization phase, or at runtime. Particularly in the latter case, a Parameter represents the interaction interface of an

Entity to the user. A Parameter is bound to an Entity. The modification of a Parameter value is only done by user, which makes Parameter semantically different from Variable.

Variable is used to specify state variables of a simulation model reflecting the simulation progress. A Variable is contained in an Entity. The modification of the Variable value is done by the simulation model, not by user. Parameter and Variable are semantically different, although they may share commonality in the technical implementation.

ConnectTo is an association relationship between Ports. It has two association ends, a sender and a receiver. The multiplicity for both association ends is zero or more. That is, a Port as sender can be associated with zero or more other Port as receiver, and vice versa.

The conceptual model is recommended but not prescribed for simulation applications using the VTIS approach. It is applied to the examples in this work (more in Section 7.2). For a simulation model conforming to this, the VTIS database model is immediately applicable to derive the database structure. For other modeling concepts, the conceptual model is useful to derive appropriate mappings to the VTIS database model.

4.3 Management of Virtual User Groups

The concept of virtual user groups involves the distribution of a simulation over the spatial dimension and the temporal dimension (see also Section 3.2.2). The spatial distribution is incorporated in a client-server architecture with a centralized simulation server. For the temporal distribution, the concept of simulation session is introduced.

4.3.1 Spatial Distribution

In principle, distributed simulation clients can be supported either by a centralized simulation server or multiple distributed simulation server parts (Section 4.1.4). The arguments to use the centralized approach in this work are the following:

- *Unique interface for simulation clients.* From the client point of view, a centralized server provides a unique and concentrated interaction interface. For the consideration of functional aspects, the centralized approach is a good starting point. It does not exclude distributed computation for the simulation server. Other issues regarding the performance of the server, such as reliability, fault tolerance, or redundancy, can be also considered, although they are not the primary focus of this work.

- *Synchronization of simulation clients.* This can be a complex issue for a distributed server (see e.g. HLA [Def96]). In the centralized server approach, simulation clients synchronize with a single coordination point. The latter matches the scenario for e-learning capable simulation well, in which members of a virtual group share the same learning context. Thus, the user-simulation-interaction is more significant for the construction of a simulation. The user-user-communication can be provided by other facilities of a learning environment, e.g. groupware, in which an e-learning capable simulation is embedded.

- *Management of simulation data.* With regard to the simulation database, a centralized approach simplifies the management of simulation data. Distributed data management, for example in case of a serverless architecture, is left for future work.

The relationship between a simulation server and simulation clients is incorporated in a three-site architecture. The composition of this architecture is described in the following.

Three-Site Architecture

The architecture as shown in Figure 4.6 is a refinement of the centralized server architecture. It takes a further factor into account, which is the separation of content-neutral operations of a virtual group from the content-specific ones. Content-neutral operations are mainly organizational operations, such as joining a group, leaving a group, and obtaining information about a group. These operations are generic to a simulation model, while content-specific operations are directly performed to a simulation model. The architecture presented here is concerned only with content-neutral operations. For content-specific operations the temporal distribution must be taken into account. They are examined in Section 4.3.2.

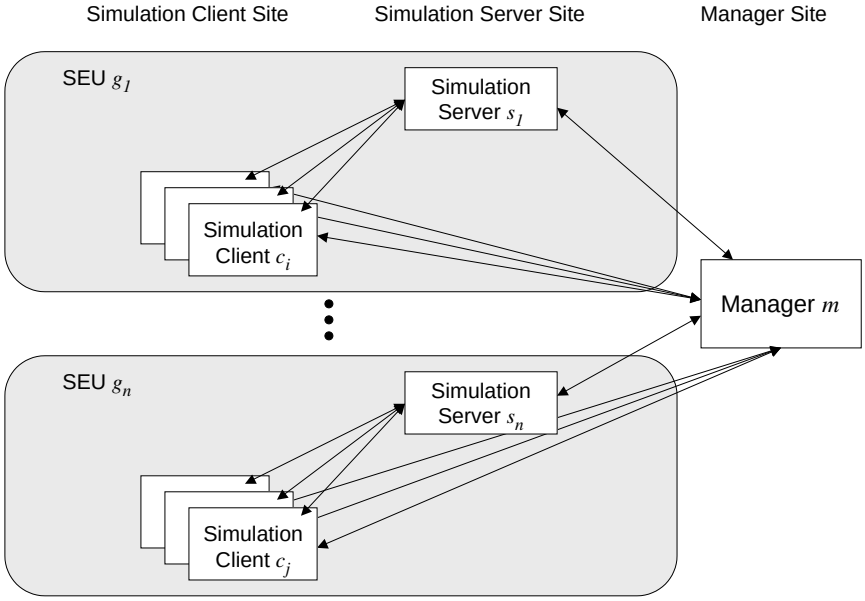

Figure 4.6: Three-site architecture of simulation environment

In terms of potential distributed locations, three principle sites are introduced: the *manager site*, the *simulation server site*, and the *simulation client site*. Obviously, the prerequisite for this is the functional partitioning, which again facilitates the scalability of the entire environment, in which a simulation model is executed. This environment is denoted as *simulation environment*. Prior to the formal definition of a simulation environment, an intuitive description of the major concepts is given below.

From the organizational point of view, a simulation environment is composed by a *manager*, and zero or more *simulation environment unit* (SEU). The computational elements of a unit are

one simulation server, and zero or more simulation clients. A simulation server maintains the execution of a simulation model, which is shared by members of a virtual user group as learning context. A simulation client supports the interaction between users and the simulation server. Each simulation client may reside on a different location, i.e. simulation client site.

Association of Simulation Clients

As simulation clients are created for users of a simulation environment, further consideration on the relationship between a user and a simulation client is necessary. Typically, each group member is assigned a simulation client, which supports his/her interaction with the simulation server. There are however also other scenarios to consider. For example, a user may play different roles in a virtual group (see also the related approach HiSAP in Section 2.3.3), or, two users may wish to act as one.

Therefore, instead of binding a simulation client directly with a user, binding is made between a simulation client and a representative of a user, called *user identity* (ID). Leaving security issues (including authentication and authorization mechanisms) for further study, the consideration can focus on static bindings. They exist for the desired duration and are independent on the semantics of the interaction performed over simulation clients.

As illustrated in Figure 4.7, users and IDs are added to the SEUs, in order to study the relationships between them. The major characteristics are outline in the following:

1. There is exactly one simulation server in a SEU.
2. Zero or more simulation clients may exist in a SEU.
3. A user may have zero or more IDs.
4. An ID must be associated with at least one user.
5. IDs used in the same SEU, independently from which user, must be unique.
6. The same ID, independently from which user, can be used in different SEUs.
7. A simulation client, if existent, is associated exactly with one user ID.
8. In case more than one user share the same ID, they are not distinguishable from the simulation client perspective.

Simulation Environment

The formal definition of a simulation environment uses the notations of set, tuple and function. A review of the notations can be found in Appendix A.

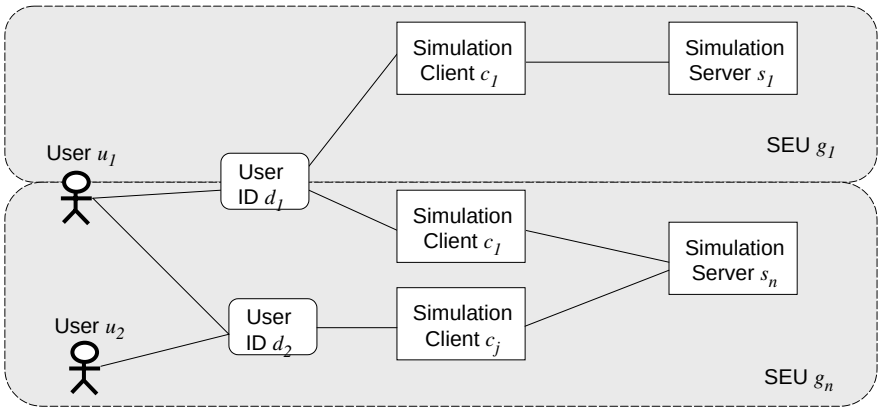

Figure 4.7: Simulation environment unit

Definition 4.1: A *simulation environment E* is defined as a tuple (m, G), where *m* represents the *manager* of the simulation environment, and *G* is a set of SEUs $g_n, n \in \mathbb{N}_0$ organized by the manager. That is,

$$G = \emptyset \text{ or } G = \{g_n \mid n \in \mathbb{N}_0\}$$

where, and also for the subsequent presentation, $\emptyset$ represents the empty set, and $\mathbb{N}_0$ is the set of natural numbers including zero.

Further, an SEU g_n is a tuple (s_n, U, D, C). s_n represents the simulation server created for g_n. *U* is a non-empty set of users utilizing facilities in g_n. It represents the virtual user group supported by the SEU. *D* is the set of IDs used in g_n. *C* is a set that includes all simulation clients allocated for g_n. That is,

$$U = \{u_i \mid i \in \mathbb{N}_0\}$$
$$D = \emptyset \text{ or } D = \{d_j \mid j \in \mathbb{N}_0\}$$
$$C = \emptyset \text{ or } C = \{c_k \mid k \in \mathbb{N}_0\}$$

The function $f_{dc} : D \rightarrow C$ is qualified as a partial surjective and partial injective function. The relation $R_{ud} : U \rightarrow D$ is not a function, because user IDs are not assigned uniquely.

Figure 4.8 presents an example SEU in an abstract form according to this definition.

4.3.2 Temporal Distribution

The temporal distribution of a simulation system allows set up of individual time schedules for members of a virtual user group.

Many processes that last time in the nature from millisecond to millions of years, can be represented in simulation in a period of time that is reasonable for human observation. In the simulated world, a process can be accelerated, decelerated, or interrupted, which is not always possible in the real world. A further advantage of simulation is, that users of a simulation are

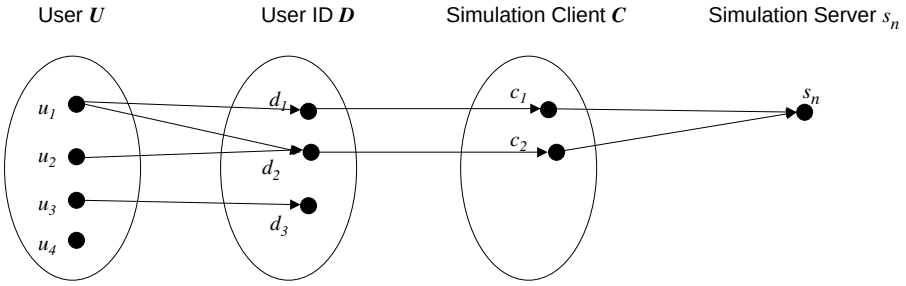

Figure 4.8: Abstraction of a simulation environment unit

able to segment a complex experiment into pieces, and to study them over hours, days, or longer. This happens also with experiments in laboratories. There, however, members of a group must be present at the same time, in order not to miss anything that happens during the experiment. An experiment carried out in laboratories lacks often of time or feasibility to repeat certain steps for absent members.

Reproducibility is one of the major characteristics of a well-founded simulation. For group members, who share the same working context with others, but prefer individual working time, the information produced by a simulation must be appropriately managed for the group. To handle this, the concept of simulation session is introduced.

A simulation session will be further refined by sub-elements in hierarchical structures. To describe the relationships between all these elements in terms of time (either simulation time, or wallclock time) the notion of **lifespan** is used. At this point, a lifespan of an element of interest is informally defined by the time between the point when the element is started, and the point when it is stopped. A formal definition is given in Section 5.1.3 in the context of temporal database.

Simulation Session

A simulation session represents the control view of the piece of learning context, that is provided by a simulation server for a continuous period of wallclock time, in which the learning context is shared by all members of a virtual group.

A simulation session is explicitly started and stopped by group members. During the lifespan of a simulation server, multiple simulation sessions may be used. To retain the shared learning context, the lifespans of different simulation sessions for a group do not overlap. This concept is depicted in Figure 4.9.

Simulation session is a concept of control, which is generic from the semantics of the underlying simulation model. Different simulation sessions may rely on the same simulation model, or on different simulation models. The background is, that students may wish to stay in the same group, but with a different learning context. It is comparable to a course at university, in which student groups are set up at the beginning of a semester, and are retained for different exercises during the entire semester.

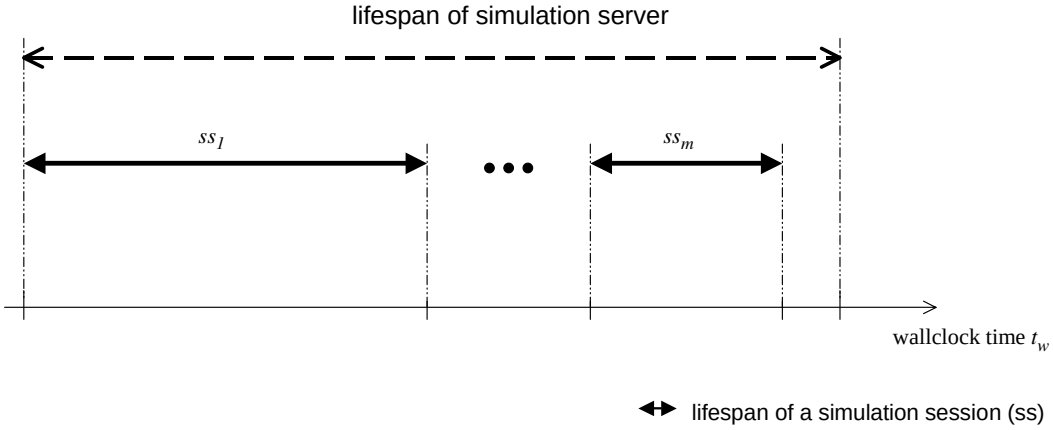

Figure 4.9: Simulation session

Modeled Events, Command Events and Steering Events

The control of a simulation session is triggered by user interaction. In terms of event-driven simulation, user interaction can be interpreted as external events. The handling of external events is a key issue for interactive simulations and emulations (see also Section 4.1.3).

In comparison with external events, internal events can be in general considered as events, whose generation and consumption occur without external intervention, but are solely dependent on the simulation model. However, the classification of internal and external events is not sufficient for the intended simulation session control, as the rest of this chapter shows. Instead, three categories of events are distinguished, namely, modeled events, command events and steering events. The idea of steering events is borrowed from [PF01], although the modeling concept of such events presented here allows more fine-grained control as stated in related work.

- **Modeled events** match the previous description of internal events. Their processing is directly determined by the simulation model. Typical examples for modeled events are sending and receiving messages, which are done between logical processes of simulation entities to trigger the state change of the entities.

- **Command events** are control events, such as start, suspend, resume, and stop, that do not change the semantics of modeled events. They may cause the processing of modeled events being speeded up, slowed down, or stopped, but neither the content nor the order of any modeled event will be changed.

- **Steering events**, opposite to command events, change the future modeled events. According to the conceptual model for simulation application (Section 4.2), requests to add or to remove an entity, to connect or to disconnect entity ports, as well as to modify the parameter of an entity, are all steering events. In comparison, change of variable values reflect the occurrence of modeled events.

The following discussion is devoted particularly to the handling of command events and steering events. Command events are considered below in a state machine for simulation sessions. A

discussion on steering events is given later in the context of timing and control.

Simulation Session States

The semantics-neutral property of command events is comparable to the principle of a video recorder. A video recorder has recording and playing features. It copies scenes on a tap (or other recording media), or plays scenes from the tap. A video recorder has usually also forwarding and rewinding features. Some of them can play in slow-motion mode.

The control of a simulation session incorporating command events approximates the operation of a video recorder. This is described by a UML state diagram [HK03] (see Figure 4.10).

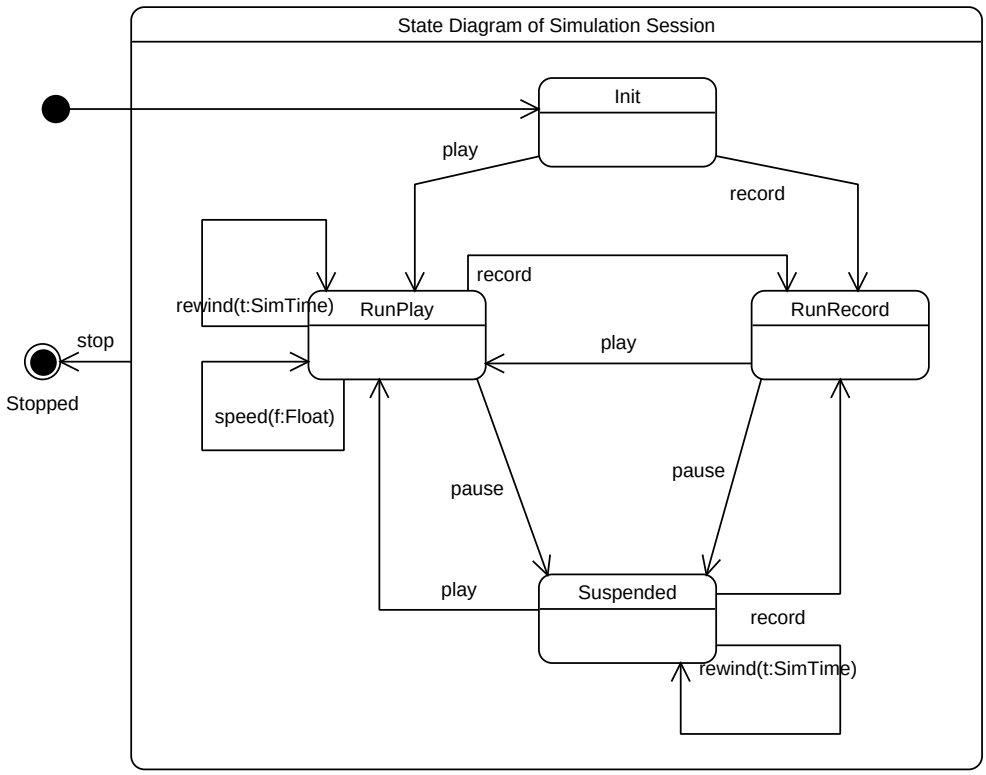

Figure 4.10: State diagram of simulation session

Apart from the predefined initial state and final state (`Stopped`) in UML, there are four explicit states: *Init*, *RunRecord*, *RunPlay*, and *Suspended*. The state transitions are triggered by six types of command event, namely `play`, `record`, `pause`, `speed`, `rewind`, and `stop`.

- *Init* is the initialization state of a simulation session. It is entered immediately after the start of the simulation session. From here, the state *RunRecord* is reached on event `record`. Alternatively, the state *RunPlay* can be triggered by the event `play`.

- *RunRecord* corresponds to the recording phase of a video recorder. This is the only state, in which steering events are accepted and processed. To allow interactivity, the simulation

time advances in synchronous with the wallclock time (more in Section 4.4.1). The event `pause` interrupts the recording phase, on which the state *Suspended* is entered.

- *Suspended* is the state, in which the simulation time is ”frozen”, while the wallclock time advances. The event `record` causes return to the *RunRecord* state. The state *RunPlay* can be reached either from here, or from *RunRecord*, both on the event `play`.

- *RunPlay* is comparable to playing a video. To set the speed of playing, the event `speed` is used. The parameter of `speed` indicates the scale factor for scaled real-time execution (explained in Section 4.4.1). Setting appropriate value for the scale factor, fast-forwarding or slow-motion features of a video recorder can be approximated. Playing can be paused by the event `pause`. To change to the *RunRecord* state, the event `record` is used.

- From any of these four states, the event `stop` causes the termination of the simulation session, i.e. entering into the *Stopped* state.

- The `rewind` event needs to be considered carefully. Applied to a video recorder, `rewind` causes a video tap being rewound back to a point, from which either a scene on the tap is played, or it is overwritten by a new one. The situation with a simulation is a bit more complex. The concerns become clearer after a discussion on timing in Section 4.4.1. At this point, it is only stated, that this event is accepted either in the *RunPlay* state, or in the *Suspended* state. The effect perceived by a user is, that the simulation returns to an earlier point in simulation time (≥ 0) as specified by the parameter of the operation. If occurred in the *RunPlay* state, it continues playing from that point in simulation time.

This section presents concepts addressing the spatial distribution and the temporal distribution of virtual user groups in a simulation environment.

Concerning the *spatial distribution*, a simulation environment relies on a centralized server architecture, and is formally described by a collection of units, called SEU, each supports a virtual group, and all are coordinated by a manager.

To control the *temporal distribution*, the concept of simulation session is introduced. It represents the control view of a piece of shared learning context for a virtual group over time. The lifecycle of a simulation session is determined by command events – one of the three categories of events that affect the progress of a simulation execution. This section ends with an analysis of the command events in terms of a state diagram for simulation session. The other two event categories, which are modeled event and steering event, in particular the latter one, are discussed in the following section.

4.4 Timing and Control Mechanisms

To realize the control of a simulation session according to the state machine described previously, timing and control mechanisms are carefully studied. The focus is put on issues that have impact on the storage of history data to reconstruct a simulation execution, which are mainly concerned

with the scale factor for real-time execution, as well as with the handling of command events and steering events.

4.4.1 Scale Factor in Real-Time Execution

Timing is a basic but important aspect for an interactive simulation kernel. To model processes of the real world, which may last very different periods of time, the simulation time may be modeled at any appropriate proportion to the physical time. Opposite to as-fast-as-possible simulations, in which the simulation time is decoupled with the wallclock time, in an interactive simulation, real-time execution is required. That is, the advance of the simulation time must be paced by the wallclock time. For the following consideration, the simulation time is assumed to use the same dimension (or unit) as the wallclock time, e.g. second.

To accept user interaction, the simulation time advances desirably in synchronous with the wallclock time. This is the recording phase, or the *RunRecord* state, according to the simulation session control. In the playing phase (state *RunPlay*), the so-called *scaled real-time execution* [Fuj00] can be used. It is a variation of the real-time execution, in which the proportion of a second in the simulation time to a second in the wallclock time is set by a constant factor, denoted as **scale factor**. As shown in Figure 4.11, the value of the scale factor may change on command events.

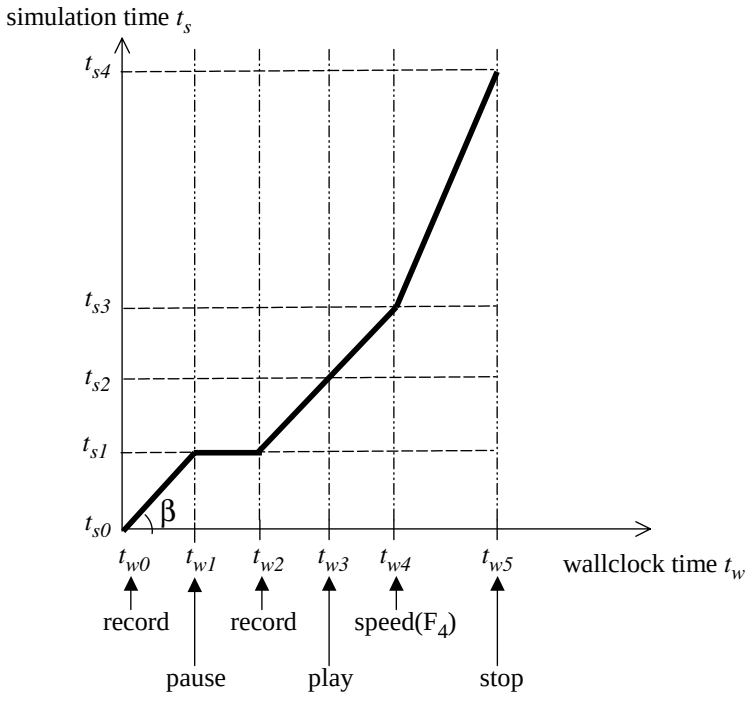

Figure 4.11: Scale factor for timing in simulation kernel

In this representation, the wallclock time is the horizontal axis, and the simulation time is the vertical axis. They are segmented into corresponding intervals, with the begin and the end of

each interval being marked by the occurrence of command events. The scale factor for a certain interval equals the tangent of the angle β between the curve in this interval and the horizontal axis. For example, in the area defined by the intervals $[t_{s0}, t_{s1}]$ and $[t_{w0}, t_{w1}]$, the scale factor should be 1 for real-time execution, because the simulation session is in the *RunRecord* state. That is,

$$tan(\beta) = 1, \text{ therefore } \beta = 45^\circ$$

Two approximations are made in this representation. First, the intervals of the simulation time are continuous time spaces. In discrete-event simulation, such an interval is in fact a sequence of discrete points in time. Second, it is assumed that the command events occur at instants of time. That is, their occurrence does not consume time.

The curve between the axes is a function from the wallclock time intervals to the simulation time intervals. This can be formalized using the notation for set and sequence [Nis99]. The lifespan of a simulation session in the wallclock time is described by a sequence $seqT_w$, which is based on the set T_w. Elements of T_w are time intervals over the wallclock time. The intervals do not overlap. Each interval appears only once in $seqT_w$. As the wallclock time does not flow backwards, the intervals in $seqT_w$ are consecutive and in the increasing order. t_{w0} is the start point of the simulation session, and $t_{w(k+1)}$ is its end point.

$$seqT_w = \langle w_0, w_1, ..., w_k \rangle,\ k \in \mathbb{N}_0$$

where,

$$T_w = \{w_n = [t_{wn}, t_{w(n+1)}] \mid n = 0, 1, ..., k;\ k \in \mathbb{N}_0;\ t_{wn} < t_{w(n+1)}\}$$

Intervals over the simulation time are described by the set T_s.

$$T_s = \{s_m = [t_{sm}, t_{s(m+1)}] \mid m \in \mathbb{N}_0;\ t_{sm} \leq t_{s(m+1)}\}$$

The simulation time is assumed to step only forwards, if not stopped. The case of backwards execution is not considered in this work. However, due to `rewind`, the simulation time intervals may overlap (see also Section 4.4.2). Therefore, the simulation time is not represented by consecutive intervals.

For T_w and T_s as sets of intervals, not as sets of discrete time points, there is a one-to-one relationship that can be described by a bijective function $T_w \rightarrow T_s$. Such a function, as defined below, determines the scale factor f_{ws}, whose value is under the assumption of scaled real-time a constant for every pair of intervals, denoted as F_n. In the example shown in Figure 4.11, the value of f_{ws} is the constant $F_4 > 1$ for the interval-pair $([t_{w4}, t_{w5}], [t_{s3}, t_{s4}])$.

Definition 4.2: The function of scale factor is defined as follows, based on the above definition for the wallclock time intervals and the simulation time intervals:

$$f_{ws} = \{(w_n, s_m) \mid w_n \in T_w;\ s_m \in T_s\}$$
$$= \{F_n = \frac{t_{s(m+1)} - t_{sm}}{t_{w(n+1)} - t_{wn}} \mid n = 0, 1, ..., k;\ m, k \in \mathbb{N}_0\}$$

In the practice, the factor is selected to compute the simulation time based on the wallclock time. For example, a certain time t_{sx} in the simulation time interval $[t_{sm}, t_{s(m+1)}]$ at the wallclock time t_{wx} in the wallclock time interval $[t_{wn}, t_{w(n+1)}]$ is:

$$t_{sx} = t_{sm} + F_n * (t_{wx} - t_{wn})$$

The range and the value of the scale factor depend on the command event applied to the interval, and the simulation session state triggered by the command event:

- In the *RunRecord* state, $F_n = 1$ is desirable, because the interactivity requires real-time execution. The event `record`, applied either to *Init*, *RunPlay* or *Suspended*, sets $F_n = 1$.

- In the *RunPlay* state, F_n can vary between 0, and the maximum which is allowed by the computing power, say F_{max}. If $F_n > 1$, the simulation time advances faster than the wallclock time, otherwise slower. By default, the event `play`, also applied to *RunRecord* or *Suspended*, sets $F_n = 1$. Other values are set explicitly by the `speed` event. In case that as-fast-as-possible execution is used instead of scaled real-time, a constant scale factor is not necessary, because the wallclock time is not significant for the execution. The scale factor is then undefined, denoted by F_{undef}.

- In the *Suspended* state, $F_n = 0$, because the simulation time does not proceed.

4.4.2 Control by Command Events

As introduced in Section 4.3.2, command events are control events, which do not change the semantics of the processing of modeled events. Six types of command events trigger the states of a simulation session (see also Figure 4.10). It represents the user's view on the control of a simulation session. To build this concept upon a simulation kernel, which does not directly support the states and events of the simulation session, mappings must be considered.

The state machine of a simple interactive simulation kernel is shown in Figure 4.12. It has three explicit states: *Init*, *Running* and *Suspended*. Only forward execution is allowed. That is, the simulation time increases in the *Running* state, and stops in the *Suspended* state. This state machine is used as an example to discuss the handling of rewind requests. Prior to the discussion, the notion of simulation instance needs to be introduced.

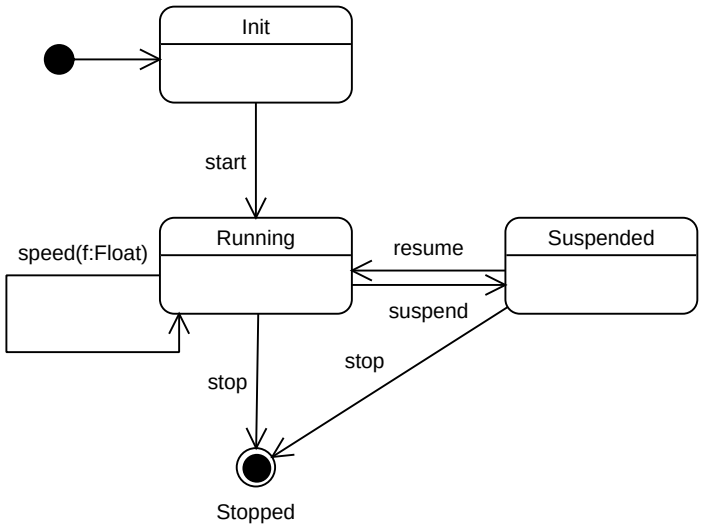

Figure 4.12: State diagram for simulation kernel

Simulation Instance

The simulation kernel state machine shown by Figure 4.12 represents only a control view on the kernel for its users, precisely, for the users of the simulation instance based on the kernel. A simulation kernel is considered to be used only together with a simulation application (see also Section 4.1.4).

A simulation instance is an executing instance of this unit, which has the following characteristics:

- A simulation instance is controlled by a simulation session.
- Over the wallclock time, simulation instances of the same simulation session do not overlap.
- At any one point in the wallclock time, if exists, there is only one simulation instance active.
- The active time of a simulation instance is the wallclock time between its creation and its termination, which is also defined by the lifespan of the simulation instance.

The relations between the lifespans of a simulation server, its simulation sessions, and the simulation instances of the simulation sessions, are illustrated in Figure 4.13.

Principally, the states of a simulation session *Init*, *Suspended* and *Stopped* can be directly mapped to the states of the simulation kernel with the same naming. The states *RunPlay* and *RunRecord* of a simulation session are mapped to the *Running* state of the simulation kernel. That is, the distinction between the playing phase and the recording phase is only aware to the session-level control. According to Figure 4.10, the difference of these two states are whether the events `speed` and `rewind` are accepted, and whether steering events are interpreted. Rewind is the critical case that is considered in the following in detail.

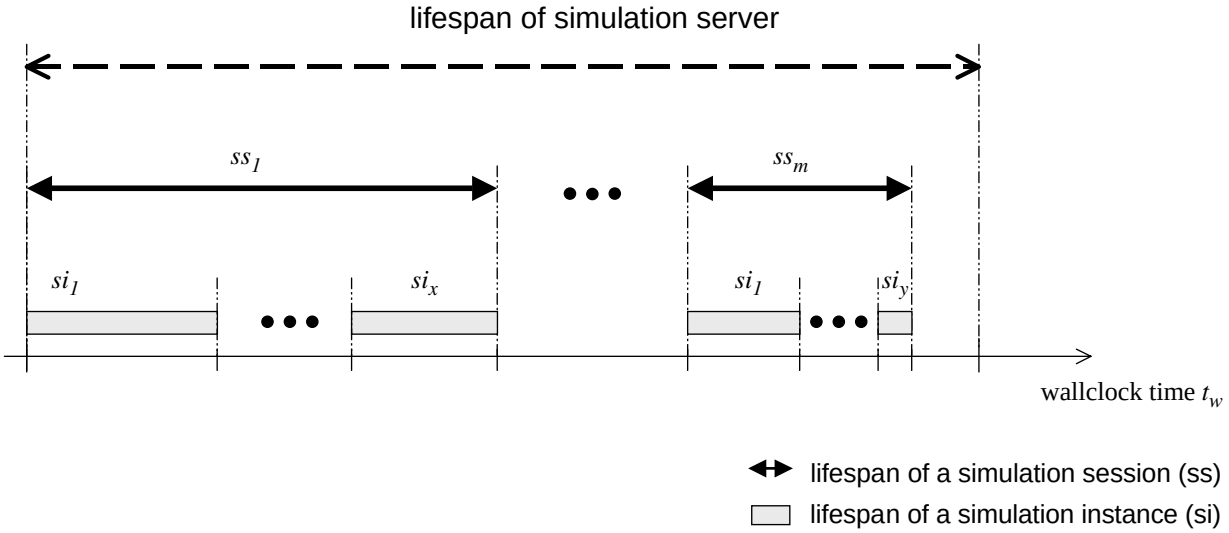

Figure 4.13: Simulation session and simulation instance

Rewind

Figure 4.14 shows an extension of Figure 4.11 with a `rewind` event. The parameter value of `rewind` is t'_{s3}. As Figure 4.11 shows, this is a simplified representation, perceived by a user of the simulation session. The user has the impression, that the simulation time at the wallclock time t_{w5}, as `rewind` occurs, is set back from t_{s4} to an earlier point t'_{s3}.

As in the simulation kernel the simulation time never decreases, the simulation session must terminate the first simulation instance at `rewind`, and start a second one immediately after that. As illustrated in Figure 4.15, the second simulation instance proceeds fast (probably at F_{max}) until t'_{s3}, and continues from there at $F_5 = 1$, because the state before `rewind` is *RunPlay*.

It is noted, that at t'_{s3}, the second simulation instance is a duplicate of the first one. Moreover, in this example, both instances are semantically identical in the interval $[t_{s0}, t'_{s3}]$, because for the semantics only the projections of the events on the axis of simulation time are significant. How command events and steering events initiated by users are ordered in an interactive simulation kernel is out of the scope of this thesis, but treated in a related joined work. Please refer to [PRS+04] for further details on this issue. In this thesis, it is assumed that all events are well ordered for event handling.

A further point is concerned with the semantics after t'_{s3}, if the state before and after `rewind` is `RunPlay`. Two options are possible:

1. Repeat the previous simulation instance from t'_{s3} until occurrence of the next `record` event, say t'_{sx}. In this case, not only the modeled events, but also the steering events occur in the interval $[t'_{s3}, t'_{sx}]$ are incorporated.

2. Taking t'_{s3} as starting point and continue only with modeled events.

This distinction would make no sense for a video recorder. The differentiation between modeled events and steering events makes a simulation fundamentally different to a video recorder.

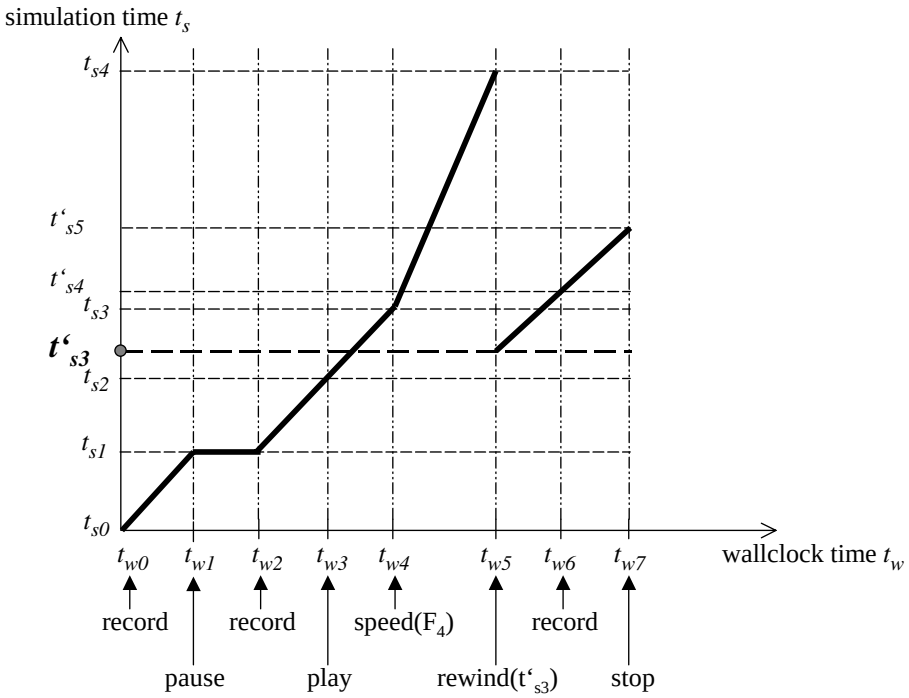

Figure 4.14: Rewind in simulation session control

A video tape is either empty or contains scenes that are always the same every time they are shown. The scenes are comparable to modeled events in a simulation. Due to steering events, every simulation execution could be a different one.

Which option is selected is treated as a question of design. Either of them, or both of them, can be meaningful for an application. With regard to the effective storage of data, it is interesting to consider, in terms of the simulation time, until which point the second simulation instance is a duplicate of the first one. For the first option, this could be the interval $[t_{s0}, t'_{sx}]$. But in any case, the equality in the interval $[t_{s0}, t'_{s3}]$ can be utilized. This conclusion is incorporated in the temporal data model presented in Section 5.2.

4.4.3 Control by Steering Events

Steering events are events, that change the future processing of modeled events. Steering events are only accepted in the *RunRecord* state. According to the conceptual model for simulation applications (Section 4.2), steering events include:

- Adding or deleting an entity.
- Changing a port connection of an entity.
- Changing a parameter value of an entity.

According to the previous consideration on `rewind`, steering events can be either initiated by users or generated automatically. Also command events can be programmed. This confirms

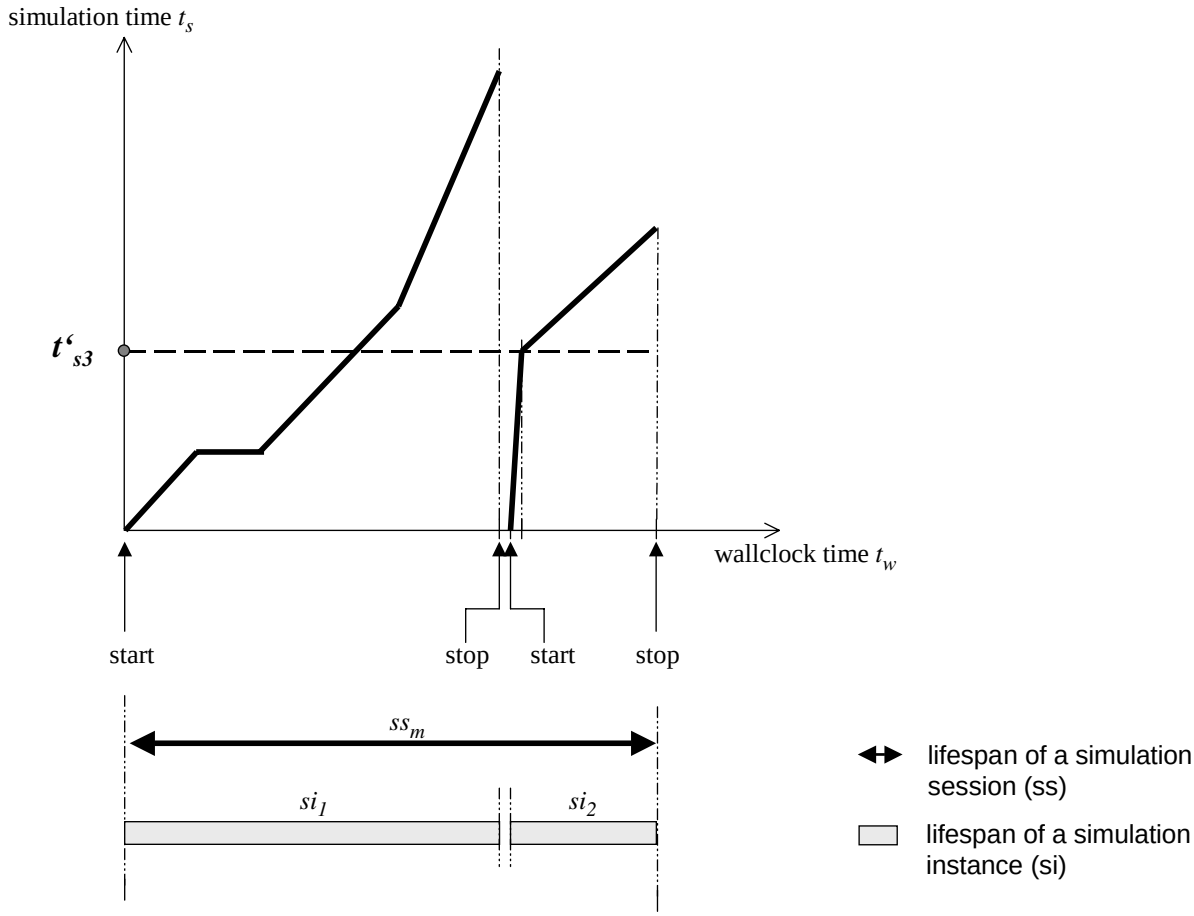

Figure 4.15: Rewind mapped to simulation kernel

that the classification of modeled, command and steering events is more appropriate than the distinction between internal and external events only.

This section discusses issues concerning the timing and control of a simulation execution, which are significant for the general structure of the simulation database that is elaborated in the following chapter. The first major issue addresses the *scale factor* for the real-time simulation execution. It is formally described, that the range and value of the scale factor, as well as its time of setting, are characterized by corresponding intervals in the simulation time and in the wallclock time, whose boundaries are determined by command events. The second crucial point is concerned with the command event *rewind*, which gives user the impression that the simulation time has been set to a time point in the past. As the simulation time conventionally never decreases, the concept of simulation instance is introduced. It divides the lifecycle of a simulation session at the time points of rewind into smaller pieces, whereas in each of the pieces the simulation time steps only forward.

4.5 Summary

This chapter introduces several concepts that build a basis for the modeling of simulation history data. The conceptual model for simulation applications is used to describe the structure of simulation models. In addition, the concepts of simulation session and simulation instance are introduced to model the control of interactive simulations in a spatial and temporal distributed environment. In particular, timing issues are examined for the time concept of the temporal data model discussed in the following chapter.

Chapter 5

Temporal Simulation Database

In the VTIS approach, the temporal simulation database provides raw data for the knowledge-based processing of teaching strategies. The core concept of the database is a temporal data model. It incorporates time into the conceptual model for simulation application introduced in Section 4.2, in order to record the history of simulation executions. It is based on some known database approaches. It includes also new concepts that concern particular issues of interactive simulations.

In the first section of this chapter, related concepts and approaches in the area of databases are reviewed. Section 5.2 presents then the temporal data model, whereas the model is described in a representation-neutral manner. The representation-specific issues are addressed by Section 5.3 and 5.4 to support the development of concrete database structures based on the temporal data model.

5.1 Basic Concepts and Related Work

Some frequently used database terminologies are introduced in the following. They originate mainly from [EN97].

A **database** is a collection of related data. A **database system**, or denoted frequently as database management system (DBMS), is a collection of programs that enable users to create and to maintain a database.

A **data model** provides concepts for the description of the structure of a database, including data types, relationships, and constraints. Basic operations for retrievals and updates on the database may be also included. Some known data models are introduced in Section 5.1.1. A **conceptual data model**, also called a high-level data model, is close to the way users perceive the data. The entity-relationship model (Section 5.1.1) is a popular conceptual data model. A *physical data model* defines how data is stored on a computer. It is also called a low-level data model. A **representational data model**, or an implementation data model, is conceptually between the two. It describes the structure of a database so, that is understood by the user, and not far from the way the data are stored. Relational data models (Section 5.1.1) are models of this category. **Object-oriented data model** is a new family of data models that emerged several

years ago. They are generally representational data models.

Data schema is the structure of a concrete database. Concepts provided by the data model are used for the description of the data schema.

A **transaction** is a logical unit of database processing that includes one or more database access operations, which can be insertion, deletion, modification, or retrieval operations.

Persistent storage is one of the essential advantages of using databases. The data of a program stored in a database survives the termination of the program. The data is then called *persistent*. The data describing variables of an object can be retrieved later by other programs.

Different **integrity constraints** may be used to ensure the correctness of the data in a database [Ger97]. A database provides capabilities to define and to enforce integrity constraints. An example is the *referential integrity constraint* which is e.g. used by relational databases (Section 5.1.1) to maintain the consistency between two relations.

5.1.1 Data Models

Some frequently used data model are introduced in the following.

Relational Data Model

A large number of the current databases is based on the relational data model. A relational database is a collection of *relation*s. Each relation is represented by a table. Each row of the table is a *tuple*. A column header is called an *attribute*. The data type applied to each column is called a *domain*. Mathematically, this model is defined by sets and relations.

To manipulate the data in a relational database, a set of operations, called relational algebra, is specified. Two important operations are `SELECT` and `JOIN`. The `SELECT` operation is used to select a subset of the tuples from a relation that satisfies the condition specified for the operation. `JOIN` is used to combine related tuples from two relations into single tuples.

A standard specification for relational databases is SQL (Structured Query Language). It provides a formalism for schema definition and data query. For an extensive introduction of the relational data model and SQL please refer to [EN97].

Entity-Relationship Model

The *entity-relationship* (ER) model is a popular high-level data model. It describes data as entities, relationships, and attributes. The ER model can be mapped to the relational data model.

The concepts of entity and attribute are close to those defined for systems (Section 4.1). That is, an *entity* represents an object of interest, either physical or conceptual. The properties of an entity are described by its *attribute*s. An *entity type* describes the commonality of a group of entities. A particular entity is characterized by *values* of its attributes. The attribute values construct the major part of a database.

*Relationship*s define how entities are related to each other. They are described by *relationship type*s. Entities are said to *participate* in relationships. The number of participating entities define the degree of a relationship, e.g. a binary relationship of two participants, or a ternary relationship

of three participants. Each entity participating in a relationship plays a particular *role* in the relationship.

There are different variations of the ER model. The *enhanced entity-relationship* (EER) model extends the ER model with concepts for generalization and specialization, as well as inheritance. These are in fact close to the object-oriented concepts.

Object-Oriented Data Model

Object-oriented databases (OODBs) emerged in the late 1980s, as a result of the integration of database technologies and object-oriented software engineering.

An OODB is a collection of objects. An object is described by a unique *object identifier* (OID), a *state* and a *behavior*. The *state* is determined by values of the object *properties*, which can be either *attributes* or *relationships*. The behavior of an object is specified by *operations* that can be performed on the object. This concept integrates the structural description of data with the manipulation operations on the data, which are for example separately considered in the relational data model. Object-oriented development concepts and languages can be applied to OODBs very well. However, some database specific issues must be taken into account, e.g. persistent data storage.

Since the beginning of the 1990s, the ODMG (Object Data Management Group) has been working on a standard for OODBs. The specification ODMG 3.0 [Cea00] was released in 2000. It includes an Object Definition Language (ODL) and an Object Query Language (OQL). ODL is based on the Interface Definition Language (IDL) defined by the OMG [Obj02a]. OQL is a declarative query language strongly inspired by SQL.

The following is an example for an ODL-based specification taken from [CZ00]. The type of object is defined by a class named `Person`. Its properties are described by the attributes `age`, `name` and `gender`, as well as by the relationship `is_married_to`. In ODMG, only binary relationships are defined. Each relationship is represented by a pair of inverse references. This example is a special case, in which both references point to the same property of the same type. In addition, each `Person` object can be manipulated by the operation `get_married`. In any case, the referential integrity for a relationship must be ensured.

```
class Person {
  attribute short age;
  attribute string name;
  attribute enum gender {male, female};

  relationship Person is_married_to
    inverse Person::is_married_to;

  void get_married(in Person p);
};
```

The syntax of OQL is aligned to SQL. For example, to find out the age of a person named ”John” from the database `persons`, the following expression is used:

```
select p.age
from p in persons
where p.name = 'John';
```

The *dot notation* is used for *path expression*s, for example `p.age`. Typically, a path expression starts with an object name, which is followed by zero or more relationship name or attribute name, separated by dots. This concept is adopted for XML Path expressions (more in Section 5.1.2).

In spite of interesting concepts, pure OODBs are not widely used today. Object-relational databases are one category of derived approaches. An alternative approach to object-oriented databases are XML-based databases.

5.1.2 XML-Based Database

The Extensible Markup Language (XML) is a data description language defined by the World Wide Web Consortium (W3C). XML is in fact a metalanguage, consisting of a few rules to define new languages. Such a language is characterized by tags, as shown below by an example. Each element of content is marked by a start tag, e.g. <Person>, and by an end tag, e.g. </Person>.

```
<Person>
  <age>35</age>
  <name>John</name>
  <gender>male</gender>
</Person>
```

XML documents are text-based and human-readable. They are essential factors for the wide use of XML for Web services [Coy02]. XML is often used to refer to the XML family, which covers a broad range of data processing technologies. Interesting for this work are in particular XML Schema and XQuery, which provide core concepts for XML-based databases.

XML Schema

An XML document is in general self-describing, and does not require an explicit grammar specification. However, a grammar specification based on XML Schema has some benefits [BPS+03]. XML Schema provides a rich set of mechanisms for specifying more extensive grammar constraints. A grammar specification is in particular worth for XML documents that share the same structure, or for XML documents that are derivations of the same structure. It is useful as a means for validation, too.

The first complete XML Schema specification was released in 2001. The specification is composed of several parts, including Primer [Wor01a], Structures [Wor01b] and Datatypes [Wor01c]. Until the publication of XML Schema, DTD (Document Type Definition) had been the only mechanism to describe grammar for XML documents. XML Schema was not intended to be a superset of DTD. DTD and XML Schema exist today in parallel. DTD is less complex

and easier to use than XML Schema. In comparison with DTD, XML Schema introduces more advanced concepts, including:

1. *Broader range of built-in data types.* Besides the string types of DTD, different integer types, floating types and time/date types are defined.

2. *Simple and complex types.* A `simple type` derives from built-in types, which may be used as restriction bases (by constraining value ranges), or as member types to construct lists or unions. A `complex type` may contain substructure, e.g. sequence of elements and attributes, while a `simple type` does not. The content of a `complex type` may be simple or complex. A `simple content` may extend (using `extension`) or restrict (using `restriction`) simple types. A `complex content`, similar to a `complex type`, may have substructure, and may use `extension` or `restriction` of other types. Simple and complex types can be used to construct element types.

3. *Namespace.* A namespace defines a scope for uniqueness of names. This construct is not specific for XML Schema. In XML Schema, it is used here to reference names in different schema scopes.

4. *Object-oriented concept.* As stated in point 2, `extension` and `restriction` are used to extend or to restrict property when defining new types based on existing types. This is an inheritance concept. In addition, the `abstract` attribute can be used to declare element or complex types as non-instantiable types, which are used as extension or restriction bases. A complete mapping for object-oriented schema is given in [BPS+03].

Using XML Schema, the grammar for the above Person-example can be specified as the following:

```
<xsd:element name="Person">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="age" type="xsd:integer"/>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="gender">
        <xsd:simpleType>
          <xsd:restriction base="xsd:token">
            <xsd:enumeration value="male"/>
            <xsd:enumeration value="female"/>
          </xsd:restriction>
        </xsd:simpleType>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

XQuery

XQuery is a new language to query XML documents. XQuery 1.0 was published just recently. It borrows features from several other query languages, among them, SQL and OQL. The type system of XQuery is based on XML Schema. XQuery 1.0 [Wor03c] is an extension of XPath 2.0 [Wor03b]. Both are based on the same data model [Wor03d].

In the **data model**, the structure of an XML document is represented as an ordered, labeled tree. Seven kinds of *node*s can construct a tree: document, element, attribute, text, namespace, processing instruction, and comment. In addition to nodes, *atomic value*s, which are single values of the simple types according to XML Schema, can be also used. A node or an atomic value is called an *item*. A series of items is known as a *sequence*. Every value in XQuery is represented by a sequence, which may contain zero or more items. This data model is applied to input and output of queries, which are XML documents. It applies also to query expressions.

Expression is the basic building block of XQuery. The major expression concepts are introduced in the following.

Path expressions are used to locate nodes. These expressions rely on XPath [Wor03b]. A path expression consists of a series of one or more *step*s, separated by "/" or "//". It can be constrained by predicates. For example, to select the node `Person` with `name` equals "John" from the document `persons.xml`, the following expression is used:

```
doc("persons.xml")/Person[name="John"]
```

A set of the so-called *axe*s are defined to navigate through nodes, for example, `child`, `descendant`, `parent` and `attribute`.

FLWOR expressions provide a means to combine and to restructure information. They are similar to the SQL `SELECT` statements. A FLWOR expression starts with one or more **`for`** or **`let`** clauses in any order, followed by an optional **`where`** clause, an optional **`order by`** clause, and a required **`return`** clause [CDF+03]. This is where the name FLWOR comes from. In the example below, the names of the male persons are returned in alphabetical order.

```
for $p in doc("persons.xml")/Person
where $p/gender = "male"
order by $p/name
return $p/name
```

The `for` and `let` clauses can bind multiple variables and tuples of variables. In this manner, information can be easily combined, for example to achieve results similarly as using the SQL `JOIN` statement.

Constructors create new elements for the output. Element, attribute, and document constructors can be used. They can be applied in combination with queries. In the following example, an element `example` will be produced, whose content is the age of the person named "John".

```
<example>
  {doc("persons.xml")/Person[name="John"]/age}
</example>
```

A set of built-in **operators** (arithmetic, comparison and sequence) and **functions** are supplied. In addition, *external variables* and *functions* can be included, either directly or over *library modules*.

Different mappings enable the storage of XML data in legacy databases that have internally non-XML data format, e.g. relational databases [CDF+03] or object-oriented databases [CZ00]. An alternative approach is a "native" XML database. According to [CDF+03], a "native" XML database is characterized by the pervasive use of XQuery and its data model at all levels of the database system. This is reflected in various aspects, including the data definition interface, the data query interface, the interoperability, and the database configuration interface.

5.1.3 Temporal Database

A *temporal database* is a database that supports some aspects of time [OS95b]. In another definition, a temporal database is one that maintains past, present, and future data [TCG+93]. A *database state*, or a **snapshot**, is the data in the database at an instant of time. Therefore, non-temporal databases are also called snapshot databases.

Time Dimensions

Roughly speaking, data in a temporal database is a function of time. Three *dimensions of time* are distinguished regarding temporal databases [JeB+98].

- The **valid time** of a fact is the time when the fact is true in the modeled reality. A *valid-time database* contains the entire history of the modeled reality.
- The **transaction time** is the time when the fact is present in the database as stored data. Hence, a *transaction-time database* allows rolling back the database to a previous state. A *bitemporal database* supports both valid time and transaction time.
- While the previous two may be either or both interpreted by a database, the third one, the **user-defined time** is interpreted directly by the user. According to [JeB+98], a database that supports only the user-defined time is not a temporal database.

Continuous time in the nature is represented in temporal data models by discrete time points. The following three concepts are used to describe discrete time points:

- Each of the time points is called an **instant**.
- An **interval** is a consecutive set of time points and is designated by its boundary points.
- Instants or intervals are used by **time stamps**.

Temporal database approaches differ mainly in the questions, how to attach time stamps to data, and how to use time stamps to retrieve data. Some representative approaches are reviewed in the following.

History-Oriented Approach

A special class of temporal databases is called **history-oriented**, whereas the history is a temporal representation of an ”object” of the real world or of a database [JeB+98]. Depending on the type of the ”object”, which can be entity, attribute, or schema, there can be entity histories, attribute histories, or schema histories.

The characteristics of history-oriented databases include:

1. History unique identifications, e.g. via time-invariant keys or object identifiers (OIDs).

2. History-related integrity constraints, e.g. history uniqueness, entity history integrity, etc., which can be expressed and enforced. The histories can be directly referenced by the queries.

Two frequently used approaches to history-oriented databases are: the *snapshot-collection* approach, and the *snapshot-delta* approach. The former views the history of a database as a collection of snapshots of the entire database. In the latter case, a history consists of a single database snapshot and its deltas. For frequent changes, each of them affects only a small part of the data, the snapshot-delta approach is more flexible. However, the flexibility is at an expense of more complexity in the data model and query processing.

A change-oriented snapshot-delta approach is DOEM (Delta OEM) [CAW99]. It is intended for semi-structured data like HTML, that does not necessarily adhere to a certain schema. DOEM is an extension of the Object Exchange Model (OEM), that depicts object-based data using labeled graphs, i.e. nodes for objects and labeled arcs for object relationships [CGMH+94] [AQM+96]. In DOEM, changes are described by node annotations and arc annotations. They contain time stamps to the represented change operations: create/update node and add/remove arc. This concept is also reflected in the query language, which is based on an extension of OQL. The graphical notation tends to have difficulties for more complex data structure [OQT01].

A number of *temporal data model*s for history-oriented databases are built as extensions of conventional non-temporal data models, such as relational, entity-relation, object-oriented or XML-based models.

Temporal-Relational Model

To extend relational data models with time, either tuples or attributes can be associated with time stamps. The latter case is more complex to model, because elements of a tuple may be associated with different time stamps. This must be incorporated in the query logic. Different variations for SQL have been developed [TCG+93].

Some concepts developed for the temporal-relational models are also adopted for other models. Two important concepts are temporal element and temporal assignment [GY88]. The concepts can be applied to the valid time, and to the transaction time.

- A **temporal element** is defined as a finite union of intervals, each of them is a subset of $[0, now]$, where 0 represents the relative beginning, and now represents the current time.

- A **temporal assignment** to an attribute is a function from a temporal element to the domain of the attribute.

Temporal EER Model

In a temporal EER model ([TCG+93], Chapter 9), every entity e is designated by a **lifespan**, which is associated with a temporal element, denoted by $T(e)$. The temporal assigment $A(e)$ is described by the function $A(e) : T(e) \rightarrow dom(A)$, where $dom(A)$ represents the domain of the attribute.

Analogously, a relationship is also associated with a lifespan, denoted by $T(r)$, which is constrained by the temporal elements of the participating entities $e_1, e_2, ..., e_n$, precisely, $T(r) \subseteq (T(e_1) \cap T(e_2) \cap ... \cap T(e_n))$. Attributes of a relationship are treated similarly to entity attributes.

A special kind of attributes is called a **surrogate**. It is a time-invariant key to identify an entity uniquely in a database.

Temporal ODMG

T_ODMG is a temporal extension of ODMG [BFGM98]. For the properties of an object, that are described in ODMG by attributes and relationships, three property "natures" are distinguished: temporal, immutable, and static. A *temporal property* is one, whose value may change over time, while an *immutable property* can not be modified during the lifespan of the object. A *static property* may vary over time, but only its current value, not its history, is stored in the database.

To describe the history data, some time-related data types are introduced, e.g. `TimeStamp` and `TimeInterval`. Both are interfaces providing methods for time manipulation. For example, `TimeStamp` allows to read a time stamp in different formats, such as `Date` (in year, month, day, etc.). `TimeInterval` provides the methods `union` and `intersect` to operate with time intervals.

Temporal XML

Histories of documents are also referred to as versions. [Nør02a] and [Nør02b] present a work on the management of versioned XML documents with temporal databases. The data model assumes that every element in an XML document is associated with a time stamp. A document is stored by a complete current version and a chain of previous versions as deltas. It applies therefore the snapshot-delta approach. The `PatternScan` operator supports temporal operations, for example to access the document history, the element history, the create time, or the delete time.

A similar concept is presented in [WZ03], where valid time is attached to attributes of elements in an XML document. In contrary, [ZD02] proposes to add an additional valid time axis to store the history of attributes and elements. Moreover, the history is stored separately from the non-temporal part, for which the consistency must be carefully considered.

An extension to the newly released XQuery is τXQuery [GS03]. Three kinds of queries are differentiated: current, sequenced, and representational. A *current query* reflects the current state of the database. A *sequence query* is applied to the history of the database, without concerning the representation of the temporal data. For these two kinds, except the keywords `current` and `validtime`, standard XQuery grammar is used. In comparison, a *representational query* refers explicitly to time stamps in the data representation, which are specified using the keywords `timeVaryingAttribute` for attributes, and `timestamp` for elements.

5.1.4 Simulation Database

Storage of simulation data is mostly used for state saving and restoration. For example, the roll-back mechanism in optimistical simulations (see also Section 4.1.2) requires the storage of previous states [Jef85] [FGUC97]. Since the data is consumed internally, proprietary methods are usually used there.

The work presented in [OS95a] is one of the few approaches that can be found in the literature, which concern the export of simulation data. It exploits a relational database to store simulation states of an analytic simulation with abstract time stamps, in order to analyze them after a simulation execution. The state of a simulation is considered as a sequence, which is globally ordered for the entire simulation. Each of them is stored in the database with a time stamp, whereas a state can be described by multiple relations. The time stamps are associated with tuples, rather than with attributes. With a graphical query language based on high-level Petri Nets, a single state, or a sequence of states, can be investigated by describing the start and the goal state, and possibly some intermediate states.

The simulation database in the VTIS approach is intended to record the history of simulation executions, including the current and past state changes. In a straight-forward manner, the structure of the database is aligned to the structure of the simulation model. The latter is described in Section 4.2 as an object-oriented conceptual model for simulation applications.

Although most of the known concepts to temporal data models as reviewed previously, are basically applicable, they are rather general to application domains. Specific issues concerning the storage of interactive simulation data have not been treated so far, in particular the co-relation between the simulation time and the wallclock time. Although a number of work for bitemporal databases can be found in the literature, e.g. [Gan99], [TJS98] and [NEE95], they are mainly concerned with the valid time and the transaction time. The semantics of the simulation time and the wallclock time is however very different. Concepts addressing this problem are presented in the following section.

5.2 Temporal Data Model

As core of the temporal database, the temporal data model is concerned with issues on the time dimensions and the integrity constraints. They are incorporated finally in a set of database entity types.

5.2.1 Time Dimensions

Time dimensions is the first issue to be addressed in a temporal data model. According to the purpose of the database as a history database, one might assume to focus on the valid time only. Usually, the valid time of a simulation database represents solely the simulation time. However, the history of an interactive simulation involves also the wallclock time. For events that reflect the user interaction, the wallclock time is particularly interesting. Although both time dimensions have significance for the history data, dual time-stamping for every event is not meaningful and also not necessary. The question is how to utilize the co-relation between the two time concepts to construct appropriate database structures.

The consideration starts with some conclusions drawn in Chapter 4. According to the event classification introduced in Section 4.3.2, events initiated by users are either command events or steering events. To have impact on a simulation execution, these events must be ordered internally for event processing, where each of them is definitively designated by a time stamp in the simulation time. It is shown in Section 4.4.2, that such a time stamp can be used to reconstruct a steering event in case of rewind. Since in the lifespan of a simulation instance the simulation time is non-decreasing, and steering events and modeled events are only processed in the running state of a simulation instance, the time stamps in the simulation time for steering events and modeled events are unambiguous on the simulation time axis (see also Figure 4.15). This property is crucial to achieve the semantical reproducibility of simulation executions.

Theoretically, the occurrence of steering events and modeled events in the wallclock time can be calculated, according to Definition 4.2 in Section 4.4.1. Prerequisite is, that the intervals segmented by command events, and the scale factor in every interval, are known. Command events serve the control of a simulation session. They trigger the states of the simulation session and set the scale factor. Command events have no impact on the semantics of other events, but on the perception of them. Therefore, dual time-stamps are used to incorporate command events.

Command events are however not directly stored in the database, but are reflected in the history of the state and the scale factor of a simulation session. This is in-line with the concept of a history-oriented database, which is intended to record the evolution of simulation entity states. According to the simulation session state specification presented in Figure 4.10, from any state, command events `play`, `record`, `pause`, and `stop` trigger always the states *RunPlay*, *RunRecord*, *Suspended*, and *Stopped*, respectively. The command event `speed` changes only the scale factor in the *RunPlay* state. All these five command events can be easily reconstructed from the state and scale factor history, which is stored in a `SimSession` entity in the database. The reconstruction of `rewind` relies on the concept of simulation instance. A simulation instance is represented as a `SimInstance` entity in the database. A `SimInstance` serves as a container for `Entity` that represents entities in the simulation model.

As a summary of this discussion:

- Two time dimensions are significant for the history data stored in the database, namely, the wallclock time and the simulation time. Dual time stamping is however not always necessary.

- Modeled events and steering events can be unambiguously identified by time stamps in the

simulation time only.

- Command events have dual time stamps. They are not directly stored in the database, but represented by the history data of the state, the scale factor, and the simulation instances of a simulation session.

Before the database entities can be described, the two time dimensions need to be formally defined. The concepts of time interval and temporal element [TCG+93] apply generally to both time dimensions. To recall, a time interval is a set of consecutive time points and is designated by its boundary points. A time element is a finite set of time intervals. A wallclock time element is denoted by T'_w, and a simulation time element is denoted by T'_s. They are distinct to T_w and T_s introduced in Definition 4.2 in Section 4.4.1, which are a particular set of wallclock time intervals, and a particular set of simulation time intervals. The boundaries of T_w and T_s intervals are determined by the points of occurrence of command events. This constraint is not set for T'_w and T'_s. T'_w and T'_s are constrained by the maximal time for interval boundaries.

Definition 5.1: A *wallclock time element* T'_w is a set of intervals, each of them is denoted as I_{wn} and is defined by a pair of boundary points in the wallclock time, denoted as t_{wx} and t_{wy}. Every interval is a subset of the interval $[0, now_w]$, where 0 represents the relative beginning of the wallclock time, and now_w represents the current wallclock time. That is, $T'_w = \{I_{wn} = [t_{wx}, t_{wy}] \mid n, x, y \in \mathbb{N}_0; 0 \leq t_{wx} \leq t_{wy} \leq now_w\}$.

Definition 5.2: A *simulation time element* T'_s is a set of intervals, each of them is denoted as I_{sm} and is defined by a pair of boundary points in simulation time, denoted as t_{si} and t_{sj}. Every interval is a subset of the interval $[0, T_{smax}]$, where 0 represents the relative beginning of the simulation time, and T_{smax} is the maximum of the simulation time that has been reached until the point of consideration. T_{smax} is the current simulation time now_s, if no `rewind` has been applied, because in case of `rewind`, a user would have the impression that the simulation time has been set to a point in the past. That is, $T'_s = \{I_{sm} = [t_{si}, t_{sj}] \mid m, i, j \in \mathbb{N}_0; 0 \leq t_{si} \leq t_{sy} \leq T_{smax}; T_{smax} \geq now_s\}$.

5.2.2 Integrity Constraints

The integrity constraints specified in this section are used as the baseline for the management of temporal data. Since the data structure is aligned to the object-oriented conceptual simulation model, principles of the temporal EER model and T_ODMG apply well, in which time stamps are associated with entities (or objects), relationships, and their attributes. Comparing to T_ODMG, only immutable and temporal object properties are considered. In terms of the temporal data model, static properties are handled the same as immutable properties. That is, this data model is also applicable for snapshot database. In addition, the reference relationship adopts the ODMG relationship. The containment relationship is a new concept to allow more complex data structure.

The integrity constraints address various aspects of the database, as described in the following.

The basic building block in the database is an **entity**. This data model defines a set of entity types, including `SimSession`, `SimInstance` and `Entity`. The former two are special purpose entities to model simulation sessions and simulation instances. The third one corresponds to `Entity` in the simulation model. An entity may contain other entities.

Every entity e in the database has an **identifier (ID)**, denoted by $D(e)$. $D(e)$ is a time-invariant attribute of e. $D(e)$ uniquely designates e in the immediate scope it is contained. An entity that contains other entities builds a naming scope for the contained entities. The global identifier of e in the entire database is a concatenation of the IDs over its containment hierarchy.

The **lifespan of an entity** e can be defined in either time dimension:

- The lifespan $L_w(e)$ defined for the wallclock time is an interval of T'_w, that is, $L_w(e) \in T'_w$.
- The lifespan $L_s(e)$ defined for the simulation time $L_s(e)$ is an interval of T'_s, but only in case T_{smax} is equal to now_s. That is, $L_s(e) \in T_s \wedge T_{smax} = now_s$.

It is noted, that this definition differs from the general definition of lifespan which is associated with a temporal element [TCG+93]. Since neither the wallclock time, nor the simulation time decreases, the lifespan of an entity is described by a single interval with consecutive time points, whose upper boundary never goes beyond the current time. This concept is incorporated in the definitions of `SimSession` and `SimInstance`.

A **time-varying attribute** of an entity e, denoted by $A(e)$, has by default the same lifespan as the entity, which is therefore not explicitly specified. The temporal value of $A(e)$ is represented as a mathematical relation. Depending on the semantics of the entity and the attribute, the relation is binary or ternary.

- A binary relation is between the lifespan of the entity in simulation time, and the domain of the attribute, that is, $(L_s(e), dom(A))$.
- A ternary relation is determined by the lifespan of the entity in wallclock time, a simulation time element of the entity, and the domain of the attribute, that is, $(L_w(e), T'_s(e), dom(A))$.

In case of a **containment relationship**, for example an entity, say e_x, is contained in another entity, say e_y, the ID $D(e_x)$ obeys the constraint on IDs stated previously. In addition, there are following constraints on the lifespan of e_x:

- If not explicitly specified, the lifespan of the contained element equals the lifespan of the container element.
- Otherwise, if $L_s(e_y)$ and $L_s(e_x)$ are specified, $L_s(e_x) \subseteq L_s(e_y)$, likewise for the wallclock time. This constraint does not apply if a time dimension is used by e_x but not used by e_y, neither vice versa.

Like ODMG, only **binary reference relationships** are considered. Such a relationship associates two entities by two inverse references. For example, entity e_m and entity e_n are associated by a reference relationship. This is described by $R(e_n)$ and $R(e_m)$, which are relationship

properties of e_n and e_m, respectively. They point to each other as inverse. The lifespan of a reference relationship is constrained by the lifespan of associated entities, in case the entities use the same time dimension. It follows for the example, in case the simulation time is used, $L_s(R(e_n)) = L_s(R(e_m)) \subseteq (L_s(e_n) \cap L_s(e_m))$. The variability of a reference relationship is expressed by its lifespan. Such a relationship does not have attributes.

A set of database entity types according to these integrity constraints are defined as a basis to develop the database structure. They are presented in the following section.

5.2.3 Database Entity Types

Four entity types are defined for the data model. They are denoted as `DBRoot`, `SimSession`, `SimInstance` and `Entity`. The classification of the four is derived directly from the simulation concepts introduced in Chapter 4.

DBRoot

`DBRoot` is the root of the simulation server database. It contains one attribute describing the lifespan of the simulation server, and may be composed of zero or more `SimSession`.

Definition 5.3 A `DBRoot` entity in the database is defined as the following:

1. The lifespan of `DBRoot` DB is defined by $L_w(DB)$.
2. DB is composed of zero or more `SimSession`, each of them denoted by x_n. That is, $DB = \emptyset \; or \; DB = \{x_n \mid n \in \mathbb{N}_0\}$.

SimSession

A `SimSession` is a special type of entity that represents a simulation session. As described in Section 4.3.2, each virtual user group is assigned a simulation server. A simulation server may use multiple simulation sessions in sequence. Therefore the lifespan of `SimSession` is defined for the wallclock time. As stated in Section 5.2.1, the history of a simulation session is described by its state and scale factor.

Definition 5.4 A `SimSession` entity in the database is defined as the following:

1. A `SimSession` x is uniquely identified by an identifier $D(x)$.
2. The lifespan of x is defined by $L_w(x)$. $L_w(x) \subseteq L_w(DB)$, where $L_w(DB)$ is the lifespan of the container `DBRoot`. There is a corresponding simulation time element $T'_s(x)$.
3. Two time-varying attributes describe x. One describes the state changes of the associated simulation session, denoted by $St(x)$.
   - $St(x) = (L_w(x), T'_s(x), dom(St))$.

- $dom(St)$ define the state space of the attribute values.
  $dom(St) = \{Init, RunRecord, RunReplay, Suspended, Stopped\}$.
  The implicit start state is ignored.
- The point of a state change, which is always triggered by a command event, is defined by an instant $t_w \in L_w(x)$, and an instant $t_s \in T'_s(x)$.

4. The other attribute $Sc(x)$ describes the changes of the scale factor.
   - $Sc(x) = (L_w(x), T'_s(x), dom(Sc))$.
   - In case only scaled real-time is used, $dom(Sc) = \{a \mid a \in \mathbb{R} \wedge 0 \leq a \leq F_{max}\}$, where $\mathbb{R}$ is the set of real numbers, and F_{max} is the maximum of the scale factor that can be achieved. If non-real-time is used additionally, for example for fast-forwarding, $dom(Sc)$ includes also F_{undef}.
   - Similarly to $St(x)$, the point of every change is defined by an instant $t_w \in L_w(x)$, and an instant $t_s \in I_{sm}(x)$, where $I_{sm}(x) \in T'_s(x)$.
5. x is composed of zero or more `SimInstance`, each of them denoted by z_n. That is, $x = \emptyset$ *or* $x = \{z_n \mid n \in \mathbb{N}_0\}$.

SimInstance

`SimInstance` is the representation of a simulation instance. Therefore, it is always contained in a `SimSession` entity. The specification of the `SimInstance` lifespan in the wallclock time is essential to reconstruct a `rewind` event. In parallel, the lifespan for the simulation time is also defined, because for a simulation instance, different to a simulation session, T_{smax} is equal to now_s.

`SimInstance` does not have a state attribute, because its state history is implicitly managed by the state attribute of the container `SimSession`.

Definition 5.5 A `SimInstance` entity in the database is defined as the following:

1. A `SimInstance` z is uniquely identified in the scope of its container `SimSession` by $D(z)$.
2. The lifespan of z is defined by a relation $(L_w(z), L_s(z))$. $L_w(z) \subseteq L_w(x)$, where $L_w(x)$ is the lifespan of the container `SimSession`.
3. z is composed of zero or more `Entity`, each of them denoted by e_n.
   That is, $z = \emptyset$ *or* $z = \{e_n \mid n \in \mathbb{N}_0\}$.

Entity

An `Entity` in the database reflects an `Entity` in the conceptual simulation model. In this chapter, if not explicitly stated, `Entity` refers to a database `Entity`.

An `Entity` is contained either in a `SimInstance` or in another `Entity`. Therefore the lifespan of an `Entity` is a subset of the lifespan of the `SimInstance`, in which it is contained. Moreover, because of the association between a `SimInstance` and its container

SimSession, for Entity, only the simulation time lifespan needs to be specified. For every time point in the simulation time lifespan, the corresponding time point in the wallclock time lifespan can be calculated.

In the world of Entity, there is only a single time dimension. This property can be utilized for the data management in several ways. Primarily, all time-varying properties of an Entity and of its sub-entities, can be described solely in the simulation time. It is therefore easier to ensure the integrity constraints. Further, in case a segment of a previous simulation instance is to be repeated, e.g. for rewind, the associated Entity segment can be easily duplicated and integrated in the new simulation instance without side effect.

Aligned to the conceptual simulation model, an Entity is described by zero or more Parameter, Variable, and Port, which are themselves also database entities. In addition, an Entity may contain zero or more other Entity.

Definition 5.6 A database Entity is defined as the following:

1. An Entity e is uniquely identified in the scope (SimInstance or Entity), in which it is contained, by an identifier $D(e)$.
2. The lifespan of e, denoted by $L_s(e)$ is defined by a subset of the simulation time lifespan of its container SimInstance or its container Entity.
3. e may contain zero or more Parameter, Variable, and Port database entities, which describe its time-varying properties, as well as zero or more other Entity. That is, $e = (P(e), V(e), B(e), E(e))$, where,
   $P(e) = \emptyset$ *or* $P(e) = \{p_m(e) \mid m \in \mathbb{N}_0\}$ for contained Parameter entities,
   $V(e) = \emptyset$ *or* $V(e) = \{v_n(e) \mid n \in \mathbb{N}_0\}$ for contained Variable entities,
   $B(e) = \emptyset$ *or* $B(e) = \{b_k(e) \mid k \in \mathbb{N}_0\}$ for contained Port entities, and
   $E(e) = \emptyset$ *or* $E(e) = \{e_l(e) \mid l \in \mathbb{N}_0; L_s(e_l) \subseteq L_s(e)\}$ for contained sub-Enity.

Definition 5.7 A database Parameter entity is defined as the following:

1. A Parameter entity p must be contained in an Entity e.
2. p is uniquely identified in the container Entity by $D(p)$.
3. The lifespan of p is identical with the lifespan of e, denoted by $L_s(e)$, and is therefore not explicitly specified.
4. p has one attribute $U(p)$ describing the value of the represented parameter, i.e. $U(p) = (L_s(e), dom(U))$. Whereby, $dom(U)$ represents the value domain. Each value change is triggered by a steering event.

Definition 5.8 A database Variable entity is defined as the following:

1. A Variable entity v must be contained in an Entity e.
2. v is uniquely identified in the container Entity by $D(v)$.

3. The lifespan of v is identical with the lifespan of e, denoted by $L_s(e)$, and is therefore not explicitly specified.
4. v has one attribute $U(v)$ describing the value changes, i.e. $U(v) = (L_s(v), dom(U))$. Whereby, $dom(U)$ represents the value domain. The value change is determined only by modeled events.

Definition 5.9 A database `Port` entity is defined as the following:

1. A `Port` entity b must be contained in an `Entity` e.
2. b is uniquely identified in the container `Entity` by $D(b)$.
3. The lifespan of b is identical with the lifespan of e, denoted by $L_s(e)$, and is therefore not explicitly specified.
4. The time-varying property of a port is described by its connection relationships, which are reference relationships. The connection between the port represented by b, and the port represented by b' (contained in `Entity` e'), is specified by the relationships $c(b)$ and $c(b')$, respectively. Whereby, $c(b)$ is a relationship of b that has the inverse relationship $c(b')$ of b', and vice versa. The variability of a port connection is defined by the lifespan of the connection relationship.
   That is, $L_s(c(b)) = L_s(c(b')) \subseteq (L_s(e) \cap L_s(e'))$.

The temporal data model described above is general to data representation format. Although it is rather an object-oriented model, relying on some concepts of the temporal EER model and T_ODMG, it can be also mapped to relational data representation.

To support the development of application-specific database structures based on the temporal data model, issues on data schema specification need to be considered. The approach used here consists in two main points. One is to define a *generic* data schema that supports the entity types of the temporal data model. This data schema is stated generic, because application-specific refinement can be derived from it. The second point is to define a *representational* data schema that is immediately supported by standard implementation mechanisms. The representation of time stamps is among others an important issue to be addressed. This approach is reflected in a MOF-based metamodel for representational data schema presented in Section 5.3, and a mapping of this metamodel in XML Schema discussed in Section 5.4.

5.3 MOF-Based Metamodel for Representational Data Schema

The Meta Object Facility (MOF) is defined by the Object Management Group (OMG) as ”a framework for specifying, constructing and managing technology neutral metamodels” [Obj02b]. Models based on MOF are interchangeable between different notations. Thanks to notation mappings, data schemas can be also easily derived. An introduction to MOF, and the MOF-based specification of the generic data schema, are the content of this section.

5.3.1 Meta Object Facility

MOF is a meta-modeling language, based on a four-layer metadata architecture. The layers are denoted by M0, M1, M2 and M3, with increasing abstraction level.

1. *Information layer* (M0) contains the data to be described.
2. *Model layer* (M1) contains the metadata that describes the data in the information layer. UML models are, for example, M1 data.
3. *Metamodel layer* (M2) contains the meta-metadata that defines the structure and semantics of the metadata in M1 layer. The metamodel for UML is M2 data.
4. *Meta-metamodel layer* (M3) contains the description of structure and semantics of meta-metadata in M2 layer. This is the layer of the MOF model.

MOF is self-describing. The MOF model itself is described by constructs of MOF. MOF is used to specify the metamodels for UML [Obj03a]. In fact, a number of UML concepts are adopted by MOF. The UML graphical representation is directly used in MOF and MOF-based specifications. As shown in Table 5.1, the metamodel presented in Section 5.3.2 is intended as a M2-model, and the data schema derived from it is a M1-model.

Meta-level	**MOF terms**	**Examples**
M3	Meta-metamodel	The ”MOF Model”
M2	Meta-metadata, metamodel	UML meta-model, VTIS MOF-based metamodel
M1	Metadata, model	UML models, VTIS XML schema-based data schema
M0	Data	Modeled systems, VTIS database data

Table 5.1: MOF layers

Major MOF modeling concepts to define information models for metadata are the following:

Classes are type descriptions of metaobjects. A metaobject refers to an abstract or technology specific object that represents metadata. The properties of a class may be described by three kinds of features, namely attributes, operations and references.

- Attributes can be mapped to get and set operations. *Attribute*s and *operation*s are used to describe the behavior of a class.
- *Reference* is a construct to describe relationships between classes. The kind of a relationship is defined by an *association* (see below). A reference of a class binds an ”exposed” end of an association with this class or a super-class of this class, and points to the class that is bound to the other association end (”referenced” end). The

concept of reference is comparable with ”call by reference” used in some programming languages such as C++, or object referencing as for example used in CORBA [Obj02a]. Over a reference, the object pointed by the referenced association end can be easily accessed.

Reference has no direct equivalence in UML, while attributes and operations can be immediately mapped to UML constructs.

Associations are constructs for expressing relationships between metaobjects. An association has exactly two *association end*s, whereas each association end has a multiplicity specification to define the possible number of objects bound to the other end.

DataTypes include constructs to define data types. MOF provides a package of *primitive data types* and constructors for *complex data types*, such as enumeration types, structure types, collection types, and alias types.

Packages are used to group model elements. Package generalization, nesting, importing, and clustering are four mechanisms for metamodel composition and reuse. Package generalization is analog to class generalization, where one package inherits model elements from another package. Package nesting refers to a containment mechanism, where a package is a component of its enclosing package. Package importing allows inclusion of only those desired elements. Package clustering binds the importing and imported package into a cluster, from which an independent instance can be created.

Constraints are used to express additional consistency rules for modeling elements. A constraint consists typically of a rule expression, an evaluation policy and a set of constrained elements. The *Object Constraint Language* (OCL) [Obj03a] is recommended as a formal rule specification language. Informal description encapsulated in a predefined format is also allowed.

Direct mappings defined for MOF include currently a mapping to the OMG Interface Definition Language (IDL) [Obj02a], and a mapping to XMI (XML Metadata Interchange) [Obj02d]. XMI defines production rules for different modeling languages. The production rules for XML Schema have been newly developed [Obj03b]. They are however currently supported by very few modeling tools.

5.3.2 Metamodel Specification

MOF is considered appropriate to specify the representational data model according to the following arguments:

- The object-oriented property of MOF allows to express generalization and specialization for the intended data structure.
- MOF includes a package for primary types, and provides mechanisms to define new data types.

- MOF supports different association concepts.

The overall structure of the metamodel is shown in Figure 5.1, using UML graphical notations. A complete specification can be found in Appendix B. The main concepts of the metamodel are outlined in the following:

1. The data types `SimTime` and `WclTime` are used by time stamps to record history data in the simulation time and the wallclock time, respectively.

2. `STInterval` and `WTInterval` are used to specify lifespans of entities, `STInterval` for simulation time lifespans, and `WTInterval` for wallclock time lifespans. Each of them consists of two attributes: `start` and `end`, describing respectively the start and end point of a lifespan. For a lifespan that has a defined lower boundary, and a upper boundary pointing to the current time, a half-open representation is used, i.e. the value of `end` is undefined. It is assumed that the current simulation time now_s and the current wallclock time now_w can be obtained from the simulation environment. The simulation kernel provides usually such features.

3. `IndexedEntry` is the base to record the history of time-varying object properties. A history is stored as an indexed list of entries. Each entry contains one or two time stamps and a value. The derived type `STIndexedEntry` is used for time-stamping in the simulation time. For dual time-stamping the type `BTIndexedEntry` is used. For type-specific values, further derived types are used.

4. `STElement`, `WTElement` and `BTElement` derive from `GlobalElement`, which has an attribute `id`. Every class, that represents an entity type of the temporal data model, is a `GlobalElement`. `STElement` is used by entities that have a simulation time lifespan, and `WTElement` is used by entities that have a wallclock time lifespan. Entities that have lifespan in dual time dimensions derive from `BTElement`.

5. `DBRoot` is introduced to represent the root of a database, which may contain zero or more `SimSession`.

6. `SimSession`, `SimInstance` and `Entity` correspond directly to entity types of the temporal data model. `SimSession` is a `WTElement` with attributes `state` and `scale`, which describe the history of the state and the scale factor.

7. `SimInstance` inherits from `BTElement` to describe lifespan in both time dimensions.

8. `Entity` is an `STElement`. It is a container for zero or more `Parameter`, `Variable`, `Port` and `Entity`.

9. `Parameter` and `Variable` are generalized by `ParVarBase`, which inherits from `GlobalElement` and contains an additional attribute `type`. `Parameter` and `Variable` contains zero or more `TimedValue` to record the value change history.

10. `Port` is a `GlobalElement`, and is represented as a container of zero or more `PortConn`, each stores the information of one port connection. This representation is a mapping to the reference relationship concept in the temporal data model. The MOF reference concept can not be applied, because it does not provide the specification of lifespan, nor other kinds of attributes. Therefore the data structure `PortConn` is defined. `PortConn` has besides an attribute for lifespan, also the attribute `inverse`, which specifies the ID of the inverse `Port`.

The metamodel relies on concepts that are supported by the majority of known object-oriented modeling languages. The MOF reference concept is for example not supported by UML, and has no direct correspondence in XML Schema, but can be used effectively for IDL-based models. The argument not to use MOF reference to describe `Port` connection property is given above. Also for time-stamping simple structures are used, that can be easily transformed to other representation formats. For this work, the format of XML Schema is selected. The mapping of the metamodel into this format is discussed in Section 5.4.

5.4 Mapping to XML Schema

XML technologies emerged only several years ago. Enormous effort has been and is still being put into the development of new concepts and tools. The specifications of XML Schema and XQuery are just part of the results of this effort. Both provide essential concepts for XML-based databases. As introduced in Section 5.1.2, XML Schema provides rich features to describe complex data structure. It is used as the type system for XQuery, which is based on various well-proven data query concepts. The strength of XQuery has been shown in some recent publications [BCS01] [BCS01] [CDF+03] [Wie02].

On the other side, the review of related work in Section 5.1.3 shows, that the temporal extension of XML is an on-going work. Different extended notations attempt to move the complexity of time-stamping and temporal expressions to elsewhere that is transparent for users. Therefore, the approach to use a representational data schema is important to be open for a future standard. The MOF-based metamodel follows already this approach, so that the mapping of the metamodel to XML Schema can be built easily upon the current standard grammar of XML Schema.

It is noted that the mapping presented here is only one possible option. As stated earlier, the MOF to XMI mapping, together with the XML Schema production rules for XMI, could be used to generate a XML Schema document from a MOF-based specification automatically. Since the technologies are rather new, there is currently rare tool-support in the practice, that is conform to the specifications.

Mapping concepts summarized in the following are used in this work to obtain data schemas manually. They are kept as simple as possible to retain the readability of data schemas. The main goal is to facilitate the implementation of the VTIS approach. Thanks to the metamodel, adaptation of the mapping concepts can be made easily.

- The schema is identified by the target namespace `"http://www.lmu.de/vtis"`, abbreviated by `vtis`.

- The data types `SimTime`, `WclTime` and `SimSessionStateKind` are mapped to `xs:simpleType`, that is, `SimTime` to `xs:double`, `WclTime` to `xs:string`, and `SimSessionStateKind` to a restriction of `xs:token`.

- A MOF-class is either mapped to an `xs:complexType` or an `xs:element`.

- The inheritance relationship between MOF-classes is described by `xs:extension` elements. For example, `STElement` is an `xs:complexType` using the `xs:complexType` `GlobalElement` as its extension base.

- As the type of a `xs:attribute` is always a simple type, an attribute of a MOF-class is either mapped to an `xs:attribute` or an `xs:element`, contained in the corresponding `xs:complexType` or `xs:element` for the MOF-class. For example, the attribute `id` is an `xs:attribute` of `GlobalElement`, and the attribute `slife` is an `xs:element` of `STElement`.

- The containment relationship between MOF-classes is mapped to the containment of `xs:element`. For example, `SimSession` is an `xs:element` with occurrence from zero to unbounded contained in `DBRoot`.

- To allow various types to describe `TimedValue`, the choice group `ValueNodeContent`, and the types `ComplexValue` and `ComplexValueItem` are introduced. `ValueNodeContent` contains the elements of `simpleValue` and `complexValue`, and can be referenced by `TimedValue` or `ComplexValueItem`. In this manner, values of complex hierarchies can be constructed. For example, a value of a structured type X containing i elements is a `complexValue` with i `item`s, each described by a `ComplexValueItem`. In case the type of an `item` is a complex one, it contains a `ComplexValue` element with sub-`item`s.

- The choice group `EntityContent` is introduced for the `xs:complexType` `Entity` to allow arbitrarily ordered containment of `Parameter`, `Variable`, `Port` and `Entity`.

- Although the element type `Entity` can be immediately applied, extensions of `Entity` may be desired for certain applications. For this, the choice group `EntityList` is useful. It is referenced by the element type `SimInstance`. The containment of `EntityList` can be adapted on demand. As shown in C.2, the generic schema can be extended easily by specific schemas. The integration point for such specific schemas is represented by `EntityList`. It contains by default the element `Entity`.

The generic data schema based on the MOF-based metamodel, applying these mapping concepts, is included in Appendix C.

5.5 Summary

This chapter presents core concepts for the temporal simulation database. The temporal data model differs to conventional ones particularly in the consideration of the two time dimensions for history data, namely the simulation time and the wallclock time. An analysis shows that dual time stamps are not necessary for every data set. The result of the analysis is incorporated in a generic database structure, which is formally described using MOF-notations. A mapping of the database structure to XML Schema is also provided.

The temporal data model does not include concepts to query the history data, because this topic is a core issue of the knowledge-based processing of teaching strategies, and is left for the discussion in Chapter 6.

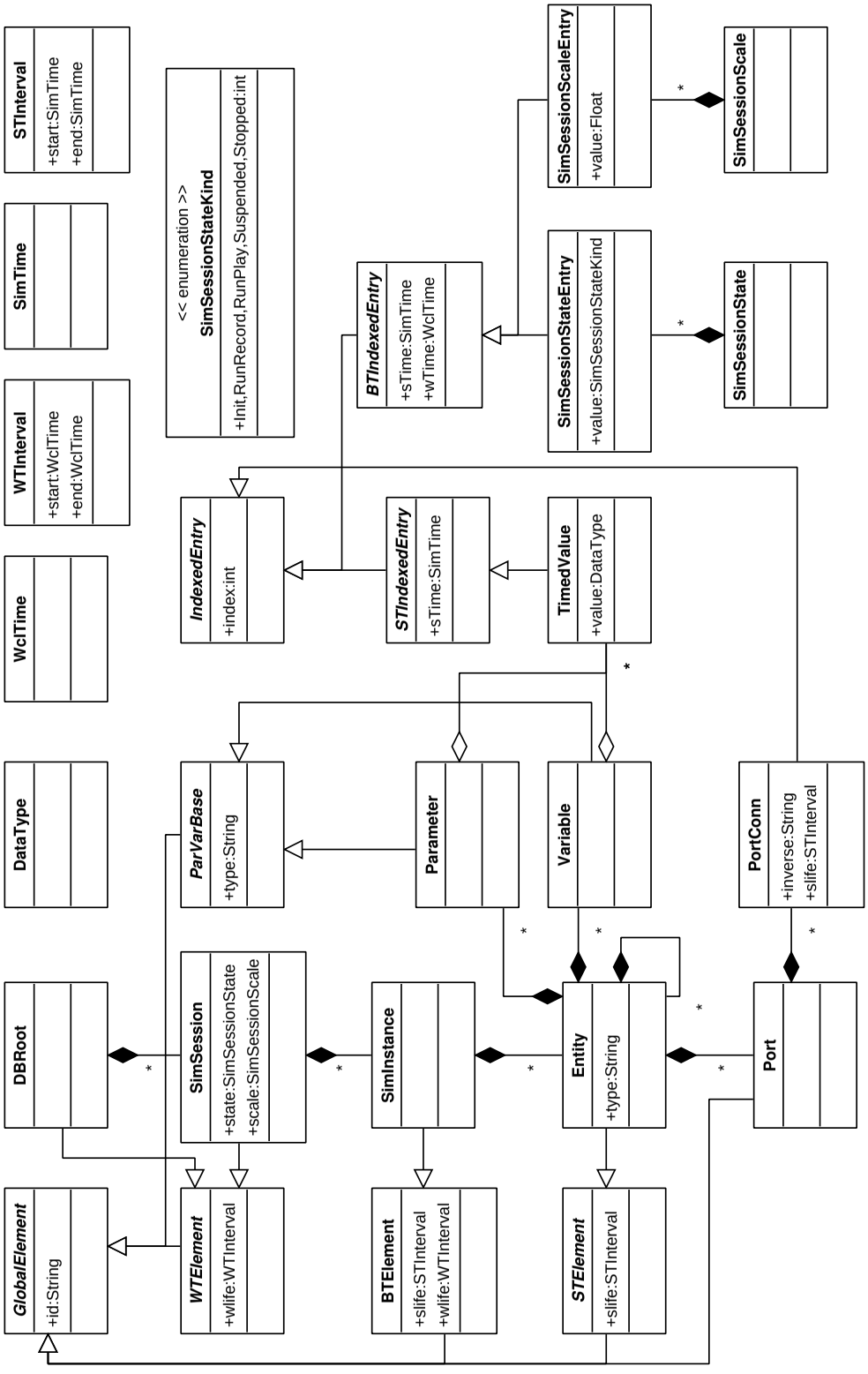

Figure 5.1: Database metamodel structure

Chapter 6

Teaching Strategies as Active Rules

This chapter focuses on the specification and the processing of teaching strategies based on the temporal database introduced in the previous chapter. Section 6.1 reviews first some related concepts from the areas of knowledge-based systems and active databases. Active rules used by active databases are considered appropriate for this part of the VTIS approach. A knowledge model for the description of teaching strategies as active rules is presented in Section 6.2. An XQuery-based rule template, as well as a rule processing architecture are introduced in Section 6.3 and Section 6.4, respectively.

6.1 Basic Concepts and Related Work

This section gives an introduction to basic concepts of knowledge-based systems and active databases. It also includes a review of related work.

6.1.1 Knowledge-Based Systems

A knowledge-based systems (KBS), sometimes referred to as an expert system [Jac99], is ”a computerized system that uses knowledge about some domain to arrive at a solution to a problem from that domain. The solution is essentially the same as that concluded by a person knowledgeable about the domain of the problem when confronted with the same problem” [GD93]. Knowledge may consist of simple facts or relations between facts. It can be either algorithmic or heuristic. To be used by KBSs, knowledge must be represented in machine-readable forms.

Core concepts of KBSs are the separation of knowledge from its use, and the separation of generic problem-solving knowledge from problem-specific data. These are reflected in the structure of KBSs, which can be considered from three different views: the user’s view, the knowledge engineer’s view, and the tool developer’s view [GD93].

From the **user’s view**, a KBS consists of an intelligent program, a problem-specific database, and a user interface (see Figure 6.1). The *intelligent program* contains all of the problem-solving intelligence of the system. The *problem-specific database* contains information about the current problem to be solved by the intelligent program, including conclusions (also intermediate

conclusions) the intelligent program has been able to derive. The *user interface* allows a user to query the intelligent program and to provide problem-specific data to the database.

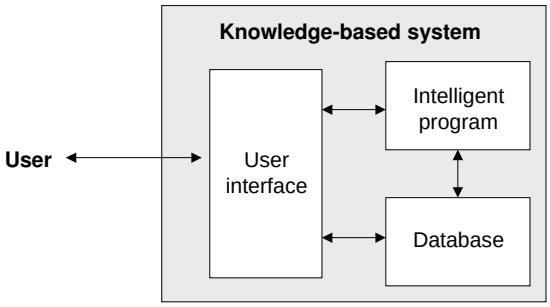

Figure 6.1: User's view of a knowledge-based system [GD93]

From the **knowledge engineer's view**, the intelligent program is composed of a *knowledge base* and an *inference engine*. The knowledge base contains all of the relevant, domain-specific, problem-solving knowledge, represented in a certain format. The inference engine interprets knowledge stored in the knowledge base to provide solutions for the current problem. The **tool developer's view** refines the knowledge engineer's view with additional development tools, such as knowledge acquisition tool or test case database.

The correspondence between these different views and the LTS-related roles is discussed in Section 3.2.4. Most significant for the VTIS approach is to provide a teacher as a content developer the knowledge engineer's view. At this point, the formalization of teaching strategies for knowledge acquisition which is the primary responsibility of a knowledge engineer, needs to be carefully considered.

Rule-based systems are the most commonly used KBSs. The **if-then pattern** is a general rule pattern, which is described by assertions. A rule may use multiple if-patterns and then-patterns. An *assertion* is a statement that something is true, for example "Elephant has long nose". The if-pattern matches one or more of the assertions in a collection of assertions, which is sometimes referred to as *working memory*. A rule is *triggered* if all the if-patterns of the rule are *satisfied*, that is, each of the if-patterns matches an assertion. A rule-based system, which has then-patterns that put new assertions into the working memory, is a **deduction system**. In comparison, a **reaction system** has then-patterns that specify actions, rather than assertions. A triggered rule is *fired*, if the then-patterns of the rule are fulfilled. It means that in case of a deduction system new assertions are established, and in case of a reaction system actions are performed [DDLS00].

A teaching strategy may involve both deduction and reaction. The modeling concept provided in this thesis focuses on the reaction property. It can be extended in future work to support deduction. Due to the fact, that a simulation database delivers raw data for the processing of teaching strategies, concepts of active databases are considered particularly interesting for this work.

6.1.2 Active Databases

Active databases are a special type of databases that respond automatically to events that are taking place either within or outside the databases [PD99]. In contrast, ”passive” databases react only to explicit commands by the user or the application program. The reactive behavior of an active database is defined by a knowledge model and an execution model.

Knowledge Model

A knowledge model supports the description of active functionality in form of **active rules**. An active rule is composed of up to three kinds of elements: event, condition, and action. Such a rule is also referred to as an ECA-rule. A typical ECA-rule has for example the following syntax:

```
on event
if condition
do action
```

An **event** is something that occurs instantaneously (see also Section 4.1). According to [PD99], the specification of an event may cover different aspects, including:

- Source of the event. It can be *structure operation* (e.g. insert, update or delete), *behavior invocation* (e.g. the event is raised by a user-defined operation), *transaction* (e.g. abort, commit, begin-transaction), etc.
- Granularity of the event. An event can be defined for *every* or *specific* members in a *set*, or in certain *subsets*.
- Type of the event. A *primitive* event is raised by a single, low-level occurrence, while a *composite* event is raised by some correlation of primitive or composite events.
- Event operators, in case multiple events are specified for a rule. The most common operators include disjunction (*or*), conjunction (*and*), sequence (*seq*), and non-occurrence (*not*).
- Role of the event. It can be *mandatory*, *optional* or *none*, indicating whether events must be given for active rules. In case of *none*, the active rules are in fact condition-action rules.

Condition is generally optional for an ECA-rule. If absent, an active rule is an event-action rule. In case both event and condition are optional, at least one of them has to be specified. If a condition is specified, it can be associated with *context*. Since the evaluation of the condition consumes time, the database state may change during the evaluation. Context can be a statement about the database states, in which the condition should be evaluated.

Similarly, context can be also used for the **action** specification. Actions can be structure operations, behavior invocations, external calls, notifications, abortion of transaction, or others.

ECA-like rules are also frequently used in other areas, such as policy-based management [DDLS00] [Rad03].

Execution Model

An execution model determines the runtime mechanism of active rule processing. The process may involve different phases, including signaling, triggering, evaluation, scheduling and execution, as illustrated in Figure 6.2. The output of a certain phase, that is used as input for the next phase, such as event occurrence or triggered rules, is also represented in the figure.

The *signaling* phase is sensitive to an event source, and is characterized by event occurrence. According to the event occurred, a set of triggered rules is selected in the *triggering* phase. The triggered rules are evaluated in the *evaluation* phase on their conditions, from which a subset called evaluated rules is chosen for the scheduling phase. Rule conflicts, if present, are resolved in the *scheduling* phase. The output of the scheduling phase are selected rules, whose actions are finally performed in the *execution* phase.

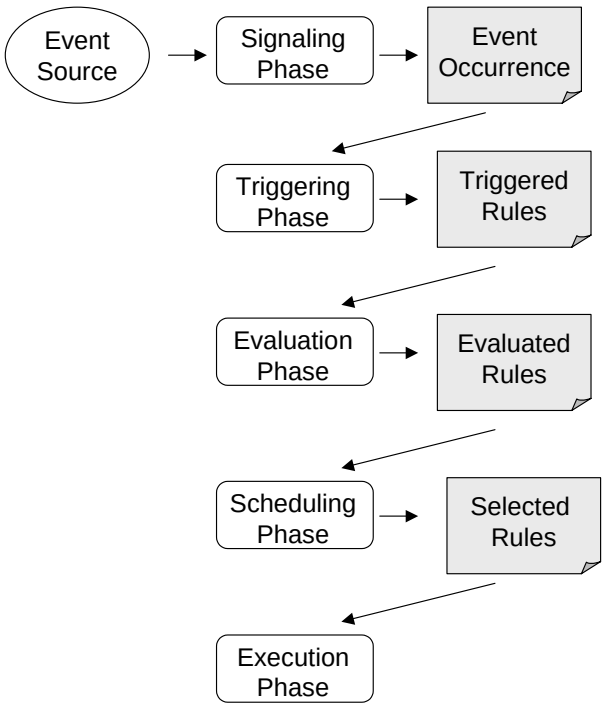

Figure 6.2: Processing phases of active rules

Different policies can be used in the execution model to control the processing phases. For example, in the scheduling phase, in case multiple rules are triggered at the same time, priorities can be used to determine the order which one is to be fired first.

Numerous proposals for active databases based on relational or object-oriented databases are introduced in [PD99]. Some new approaches for XML-based databases can be also found in the literature.

Active Rules for XML

An ECA language for XML, called AXML, is presented in [BPW02] and [AAC+99]. An AXML-event is either an insertion or a deletion of nodes in an XML document. XPath [Wor03b] expressions are used for the condition part. The action part may consist of one or more insertion and deletion operations, which are executed again on XML documents. The implementation of AXML, in particular for the events, relies on XML DOM (Document Object Model) [Wor00].

A similar proposal is presented in [BCS01]. In addition to insertion and deletion, update of nodes is the third kind of events. The condition specification is based on a pre-release of XQuery [Wor03c]. The action part of the rules describes notifications. The XML protocol SOAP (Simple Object Access Protocol) [Wor03a] is used to distribute notifications.

Recalling the discussion in Section 2.2.3, teaching strategies are based on the assessment about students, and are carried out by actions. The behavior of students, that directly affects a simulation execution, is reflected in the history data stored in the simulation database. Although not the single source, the history data provides an essential input for the assessment which again determines the actions to be performed.

Active rules apply well to the basic idea as depicted above. It is not goal of this work to develop brand new concepts for active databases or rule-based expert systems, but to exploit existing ones for a case study to demonstrate their applicability to the implementation of teaching strategies. There are still some crucial issues to address for the knowledge model and the execution model of the active rules. They are discussed in the remainder of this chapter.

6.2 Knowledge Model

In this section, the knowledge model used to construct active rules is discussed. The foundation of this model is the temporal data model introduced in Section 5.2.

6.2.1 Event

Events are directly associated with database manipulation operations. Two types of events are assumed, namely *insert* and *update*. The event type *delete*, as for example included in the proposals [BPW02] and [BCS01], is not considered, because the simulation database is a history-oriented database. Principally, any change in a simulation execution, as far as modeled, is documented in the database as history data. Deletion of a simulation entity, for example, causes an update of the lifespan entry for the entity.

The classification of insert and update events can be directly derived from the temporal data model. In the following description, denotations introduced in the MOF-based metamodel (Section 5.3) are used.

Insert

Insert events can be classified into insertion of new database elements, new relationships, or new history entries, as presented in Table 6.1.

	Caused by command event	Caused by steering event
new entity	`SimSession`, `SimInstance`	`Entity`
new relationship		`PortConn`
new history entry	`SimSessionStateEntry`, `SimSessionScaleEntry`	`Parameter::TimedValue`, `Variable::TimedValue`

Table 6.1: Insertion of database elements

According to the temporal data model, attributes of `SimSession` or `Entity` do not have their own lifespans, but exist as long as their parent entities. Thus, attributes are created together with their parent entities, and their creation is not explicitly represented as insert events. Analogously, `Parameter`, `Variable` and `Port` are also not explicitly inserted, because they are always contained in an `Entity`.

For insert events, either insertion of new entities, new relationships, or new history entries, it can be further differentiated, whether they are caused by command events or steering events in the simulation, as described in the following:

1. The command event `start` causes the creation of a new `SimSession` entity.

2. The command event `rewind` causes the creation of a new `SimInstance` entity.

3. The command events `play`, `record`, and `pause` add new entries to the state attribute of the `SimSession` entity (`SimSessionStateEntry`), while the command event `speed` adds new entries to the scale attribute (`SimSessionScaleEntry`).

4. Steering events in a simulation can be associated with the creation of new simulation entities, the connection/disconnection of ports, as well as the change of parameter or variable values. This is reflected in the database by the insertion of new `Entity`, new `PortConn` relationship of a `Port`, and new value history entries (`TimedValue`) for `Parameter` or `Variable`, respectively.

Update

Update usually refers to overwriting an old value by a new one. Either the old value is undefined, or it is not of interest for the database and thus can be overwritten by the new one. For the latter case, *static* attributes are defined in T_ODMG [BFGM98]. In the base temporal data model, only the *end* attribute of entity lifespans can be updated. It is set undefined when the entity is first time created. Static attributes in a derived data model can be treated analogously.

The wallclock time lifespan of a simulation session or a simulation instance is represented by a *wlife* attribute in the associated `SimSession` or `SimInstance`, respectively. The simulation time lifespan of a simulation instance, a simulation entity, and a port connection is represented by the *slife* attribute of `SimInstance`, `Entity`, and the relationship `PortConn`. The *start* attribute of a lifespan, either a wallclock time lifespan, or a simulation lifespan, is set immediately after the according entity is created. In comparison, the *end* attribute remains undefined until the entity is terminated. In other words, as long as the lifespan is half-open (i.e. undefined *end* attribute), the entity is active in the simulation. The *end* attribute documents the time of its termination.

Caused by command event	Caused by steering event
`SimSession::wlife::end`, `SimInstance::wlife::end`, `SimInstance::slife::end`	`Entity::slife::end`, `PortConn::slife::end`

Table 6.2: Update of lifespan end

6.2.2 Condition

The condition part of active rules is primarily based on queries on the data stored in the simulation database. The generic data schema of the database is based on a MOF-based metamodel as presented in Appendix B, and is specified in Appendix C using XML Schema. Principally, any query language supporting the generic data schema can be used. Care must be taken for time stamp and time interval related queries which are denoted as temporal queries [TCG+93]. In the following, some frequently used operations and use cases are summarized. It serves the specification of the rule template, as further considered in Section 6.3.

Time Stamp and Time Interval Operations

Operations on time stamps and time intervals are essential to support temporal queries. A time stamp represents an instant of time, either in the simulation time or in the wallclock time. Every entry in the history of a time-varying attribute contains a time stamp. The simulation time and the wallclock time may use different units, and may be represented in different forms. Since the representations can be usually converted into a number, e.g. long integer or double, comparison operators such as $>$, $\geq$, $<$, $\leq$, $=$ can be used. In addition, for the wallclock time, extraction operators such as `seconds-from-time` are also useful.

An interval is considered in the context of temporal database as a set of consecutive instants. Therefore, conventional set operators such as $\supset$, $\supseteq$, $\subset$, $\subseteq$, $=$, $\cap$, $\cup$ apply well to intervals. In addition, it can be useful to determine whether an instant is an element of an interval.

Query Use Cases

Applying the operations on time stamps and time intervals, queries of the temporal simulation database can be classified into the following use cases. The use cases are not intended as an exhaustive but an initial set of primitives to construct complex conditional expressions.

Use case 1: Wallclock time lifespan of `WTElement` or `BTElement`.

Can be applied to `DBRoot`, `SimSession`, and `SimInstance`.
Using $L_w(d) = [t_{ws}(d), t_{we}(d)]$ to refer to the wallclock time lifespan of the corresponding entity in the simulation environment, two cases can be distinguished: (a) $t_{we}(d) = now_w$, where now_w is the current wallclock time. That is, the entity is still running. (b) $t_{we}(d) < now_w$. That is, the entity is already terminated. In any case, the duration can be calculated by $t_{we}(d) - t_{ws}(d)$.

Use case 2: Simulation time lifespan of `STElement` or `BTElement`.

Can be applied to `SimInstance`, `Entity`, and `PortConn`. The interpretation of the simulation lifespan is similar to the previous one, except that `PortConn` has no direct correspondence in the simulation environment, but is an abstraction of a port connection.

Use case 3: List of immediately contained elements, whose wallclock time lifespan $L_w(e)$ intersect with the interval I_w, that is, $L_w(e) \cap I_w \neq \emptyset$.

Can be applied to `DBRoot` and `SimSession`, to obtain contained `SimSession` and `SimInstance`, respectively.

Use case 4: List of immediately contained elements, whose simulation time lifespan $L_s(e)$ intersect with the interval I_s, that is, $L_s(e) \cap I_s \neq \emptyset$.

Similar to the previous case, this can be applied to `SimInstance`, `Entity`, or `Port`, to obtain contained `Entity`, sub-`Entity`, and `PortConn`, respectively. `Parameter`, `Variable`, and `Port` that do not have explicit lifespan description, are not evaluated.

Use case 5: Given a time point t_{wx}, find an immediate contained element, that $t_{wx} \in L_w(e)$, where $L_w(e)$ is the lifespan of the element.

Applicability as use case 3. Find the corresponding entity which is used at time t_{wx}. In case $t_{wx} = now_w$, it is the currently used one.

Use case 6: Given a time point t_{sx}, find an immediate contained element that $t_{sx} \in L_s(e)$, where $L_s(e)$ is the lifespan of the element.

Applicability as use case 4. Find the corresponding entity which is used at time t_{sx}. In case $t_{sx} = now_s$, it is the currently used one.

6.2.3 Action

Actions to carry out teaching strategies are classified in Section 2.2.3 into control actions that have direct impact on the simulation execution, and actions that do not. The former category of actions causes feed-back to the source of rule events. Most kinds of rule conflict are due to such feed-back actions [BPW02] [AHW95]. For example, two rules are triggered at the same time. Executing the action of one rule first, rather than otherwise, might cause the simulation to progress differently. In this case, the rule scheduling must define which rule is fired first.

Actions in the first category considered currently are those that cause **command events** to control a simulation execution. As shown in Section 4.3.2 and Section 4.4.2, command events, opposite to steering events, do not change the semantics of how a simulation model is executed. They controls only when a simulation execution is started or terminated, as well as the speed of the simulation execution. The processing of command events is determined by the simulation session state model. Therefore, even if command events are issued in the wrong order, they will not cause any unexpected behavior. An extensive study concerning steering events is left for future work.

Actions in the second category supported at the time being are the following two: notification, and creation of student records. As shown in Section 2.3, **notifications** are frequently used for teaching. They are informative actions that do not directly affect a simulation execution. Therefore, the scheduling of rules can be kept simple, because rule conflict might occur only due to the content or the order of notifications. As notifications are delivered to the user, no further side effect for the simulation execution can occur. **Creation of student records** is useful for the off-line assessment. While notifications are distributed to the users, student records can be collected anywhere desired as data that can be reviewed or used later.

The informal description of the knowledge model in this section is completed in the following section by a rule template with formalized grammar. The XML query language XQuery [Wor03c] is considered appropriate as a foundation for the rule template.

As introduced in Section 5.1.2, XQuery is a query language that provides rich mechanisms for data manipulation. Path expressions and FLWOR expressions can be used to construct complex queries. In addition, to form output of queries into XML documents, a set of constructors are defined. A means to efficiently structure queries are function calls. Besides a number of predefined functions and operators [Wor03e], user-defined functions or external functions can be used. XQuery-based active rules are immediately applicable for database structures based on the schema presented in Appendix C, too.

6.3 XQuery-Based Rule Template

This section presents an XQuery-based rule template for active rules. It comprises a rule grammar for formalized syntax, and a set of functions as query patterns supporting in particular the condition part and the action part.

6.3.1 Rule Grammar

The grammar for the active rules is specified in Definition 6.1 (see below). This grammar uses the idea, although not the notations, of **two-level grammars** originally defined by van Wijngaarden, called W-grammar [ISO96] [Zha03]. W-grammar was originally intended for context-free grammars which are not assumed in this case. A two-level grammar consists of *meta productions* for the first-level grammar, and *hyper productions* for the second-level grammar. The hyper productions use notions defined in the meta productions.

The grammar presented in Definition 6.1 is a second-level grammar, using the grammar of of XQuery ([Wor03c], Appendix A) as the meta productions. The EBNF (Extended Backus-Naur Form) notations used by the XQuery grammar is also adopted.

Definition 6.1: The grammar of active rules describing teaching strategies is defined as a second-level grammar, using meta productions of the XQuery grammar.

Assuming the XQuery grammar is a grammar denoted as $G_x = (V_x, \Sigma_x, R_x, S_x)$, where

- V_x is the set of variables,
- Σ_x is the set of terminals,
- R_x is the set of productions, and
- S_x is the start symbol.

The second-level grammar is described by $G_v = (V_v, \Sigma_v, R_v, S_v)$, where

- V_v, Σ_v, R_v and S_v have respectively the same meaning as V_x, Σ_x, R_x and S_x.
- Variables adopted from the meta productions are enumerated as the following: $V_x \cap V_v = \{Prolog, Char, QName, PathExpr, QueryBody\}$.
- The productions R_v are defined below using the notations of EBNF.

```
RuleSet           ::= Prolog | RuleExpr* | RuleComment*
RuleComment       ::= "(!" (RuleCommentContent | RuleComment)* "!)"
RuleCommentContent::= Char
RuleExpr          ::= "rule" QName "{" EventExpr?
                      ConditionExpr?  ActionExpr "};"
EventExpr         ::= "event:" EventExprContent
EventExprContent  ::= ("insert" | "update") PathExpr
ConditionExpr     ::= "condition:" QueryBody
ActionExpr        ::= "action:" QueryBody
```

Teaching strategies are specified by a set of rules (`RuleSet`), including a prolog, as used in XQuery specifications, zero or more rule expression (`RuleExpr`) and any number of comments (`RuleComment`). A rule has an identification that is defined by a QName. The latter is a type for qualified names (may include prefix indicating namespace) defined in XML DTD and

XML Schema. The event part and the condition part of a rule expression are optional. If the event part is missing, the rule is used probably for the off-line assessment, otherwise for on-line use. Absence of the condition part refers to an unconditional action. Basically, the rule event part is described by path expressions, either preceded by the keyword *insert* or the keyword *update*, because insert and update events are associated with changes of database nodes. The condition part and the action part may use any expressions defined for XQuery, including path expressions and FLWOR expressions, therefore it is defined by `QueryBody`. An example using this template is presented in Section 8.2.5.

6.3.2 Functions

As patterns for the rule description, a set of user-defined functions and external functions are included below. They are aligned to the operations and use cases presented in Section 6.2.2. In addition, an external function named `sessCmd` is included as action execution interface to issue command events.

Function Name	Kind	Description
sessCmd	external	Executes the session command indicated by the parameter.
wcltimeNow	external	Returns the current wallclock time.
simtimeNow	external	Returns the current simulation time.
wcltimeDuration	external	Returns the duration between two wallclock time points specified by the parameters in string form.
simtimeDuration	external	Returns the duration between two simulation time points as specified by the parameters in string form.
wcltimeToDouble	external	Returns the double value for the specified wall-clock time point in string form.
wcltimeToString	external	Returns the string value for the specified wall-clock time point in double.
rootNode	user-defined	Returns the root node of the database XML-document to be evaluated.
allSimSessions	user-defined	Returns the nodes of all SimSession stored in the database.
allSimInstances	user-defined	Returns the nodes of all SimInstance associated with the SimSession identified by the parameter.

continued on next page

continued from previous page

Function Name	Kind	Description
wlifeStart	user-defined	Returns the start of the specified wlife node in string form.
wlifeEnd	user-defined	Return the end of the specified wlife node in string form.
wlifeDuration	user-defined	Returns the wlife duration of the specified wlife node in string form.
wlifeIntersect	user-defined	Returns the wlife intersection of specified wlife nodes in form of a wlife node. No intersection is identified by the start and the end of the returned node setting to -1.0.
inWTInterval	user-defined	Returns true if the time point identified by the first parameter is in the interval specified by the second parameter. Otherwise false is returned.
slifeStart	user-defined	Returns the start of the specified slife node as double.
slifeEnd	user-defined	Returns the end of the specified slife node as double.
slifeDuration	user-defined	Returns the slife duration of the specified slife node as double.
slifeIntersect	user-defined	Returns the slife intersection of specified wlife nodes in form of a slife node. No intersection is identified by the start and the end of the returned node setting to -1.0.
inSTInterval	user-defined	Returns true if the time point identified by the first parameter is in the interval specified by the second parameter. otherwise false is returned.
buildSTInterval	user-defined	Returns a slife element constructed from the specified time points.

Table 6.3: Functions for rule specification

6.4 Architecture for Rule Processing

The knowledge model is reflected into an architecture for rule processing as illustrated in Figure 6.3. This is a refinement of the overall VTIS architecture as presented in Section 3.3. The architecture assumes the processing of active rules coupled with the simulation execution, i.e. on-line assessment. For off-line assessment a subset of the shown components is used.

Core of the architecture is the controller. It is supported by the rule repository that manages all available active rules. Further, the controller is sub-structured into an event detector, a condition evaluator, and an action executor. This structure is in-line with the knowledge model and the

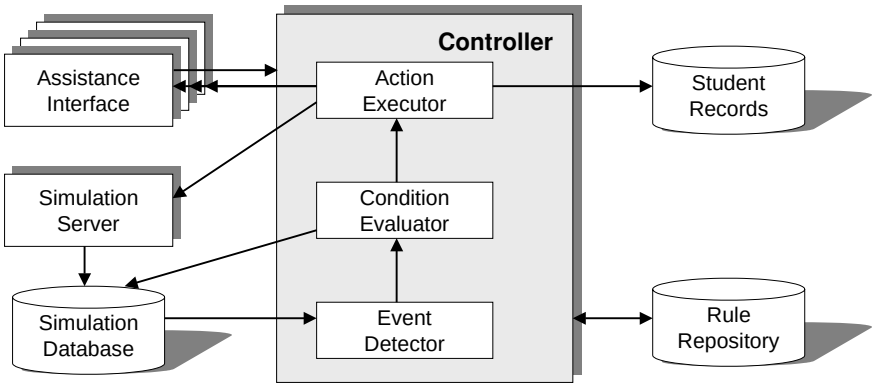

Figure 6.3: Architecture for rule processing

general rule processing concept as shown in Figure 6.2.

The event detector incorporates mainly the triggering phase. Signaling is assumed to be a feature provided by the simulation database. When the simulation database signals that data has been inserted or updated, the event detector compares the signaled event with the event part of available rules. In case some rules are triggered, it invokes the condition evaluator to process the triggered rules. The condition evaluator consults the simulation database to determine the set of selected rules. If the set of selected rules is not empty, the action executor is invoked, which performs actions according to the action part in the selected rules. That is, it issues either command events to control the simulation server, or generates notification and delivers them to the assistance interface, or creates entries for the student records.

6.5 Summary

This chapter presents the final conceptual part of the VTIS approach. It relies on the principle of active rules which are triggered by events related to databases. The idea is to model teaching strategies as active rules, in order to use them coupled with the occurrence in a simulation execution over the simulation database.

An active rule is typically an event-condition-action rule. The construction of the three rule parts is described in a knowledge model. A rule template incorporating a knowledge model is provided, which is based on the grammar of XQuery. An architecture for the rule processing as a refinement of the overall VTIS architecture is also presented.

The following chapter gives an idea how the concepts introduced so far can be realized using available techniques.

Chapter 7

Implementation Concepts

The implementation of the VTIS approach is an extension of the simulation environment developed in a project named VIP (Virtuelles Informatik Praktikum), which has been sponsored by the German Government as part of the Initiative ”New Media in the Education” [Ger]. VIP stands for ”Virtuelles Informatik-Praktikum” (engl. virtual practical courses in computer science). The results of the project have been presented in [LLP02], [LLPM+03b] and [PRS+04].

The VTIS concepts introduced in Chapter 4 are implemented in the VIP simulation environment, which includes an interactive simulation kernel, called RtDaSSF (a real-time extension of the simulation kernel DaSSF), and a generic core API supporting the development of simulation applications. Section 7.1 gives some details on this environment.

VIPNet is a network simulation based on the VIP core API. A set of modules and applications of VIPNet have been developed in the VIP project. The VIPNet API, as well as some modules and applications used for examples in this work, are introduced in Section 7.2.

The implementation concepts for the VTIS database and the rule-based control are presented in Section 7.3 and Section 7.4, respectively.

7.1 VIP Simulation Environment

The core concepts and components of the VIP simulation environment are briefly reviewed in this section. For further details please refer to [LLPM+03b] and [PRS+04].

7.1.1 Architecture

The VIP simulation environment is based on a centralized simulation server architecture. It is also a three-site architecture, as elaborated in Section 4.3.1. The simulation client site provides a Java-based graphical interface for user interaction. The manager is the coordinator in the environment. It instantiates and terminates the simulation servers.

The distributed communication is implemented using CORBA (Common Object Request Broker Architecture) [Obj02a], a widely-used middleware platform. CORBA is an object-oriented framework for distributed computing. It includes the definition language IDL and mappings of

IDL for various programming languages, so that applications using different implementation languages and different operating systems can easily interoperate. For the VIP simulation environment, the multi-language property of CORBA is utilized to combine the Java-based user interface with the C++-based simulation server.

7.1.2 Simulation Kernel RtDaSSF

The simulation kernel of the VIP simulation environment is called RtDaSSF. It is a real-time extension of DaSSF [LN01], which is a C++ implementation of SSF [Cow99] (see also Section 4.1.5). DaSSF is an event-driven, conservative, parallel simulation kernel, constructed for large-scale simulations, such as simulations for ad-hoc networks [LPN+01].

The underlying SSF concept is a crucial argument to select DaSSF for the VIP simulation environment. Since DaSSF was designed for as-fast-as-possible simulations, to enable interactive simulation applications, extensions must be made. As presented in [PRS+04], the timing for scaled real-time execution, and the interface to observe and to control the kernel behavior, are key issues have been addressed.

The principle of scale real-time is explained in Section 4.1.3 and 4.4.1. To implement this, the event scheduling mechanism of DaSSF is adapted by introducing a so-called ”real-time key”. In case of real-time execution, this key is compared with the wallclock time, taking the scale factor into account. If the key is greater or equal to the current wallclock time, the event is fired immediately. Otherwise, it is rescheduled into the event queue, until the next check by the scheduler.

The control interface is modeled by the object types `RtEntity`, `RtTime` and `RtObserver`. `RtEntity` is derived from `SSF_Entity`, an implementation of the SSF Entity class. Additional methods are provided with `RtEntity`, such as `suspendAll`, `resumeAll`, and `setSpeed`, to trigger the kernel state or to change the scale factor, as illustrated in Figure 4.12. `RtTime` is a facility to obtain the current simulation time, or the current wallclock time. The kernel state, and the deviation of the real-time behavior, e.g. slope or error, can be inquired at the `RtObserver` interface.

7.1.3 VIP Core API

The VIP core API was intended as a generic framework for interactive simulations with distributed user environment. Figure 7.1 describes only the view for the simulation model developer. From this point of view, the VIP core API consists basically of four class types: `Entity`, `Port`, `Parameter`, and `Variable`. This structure is in-line with the conceptual model introduced in Section 4.2. Each of the class types is enriched by a set of methods, that implement application generic behavior based on the DaSSF and RtDaSSF functionality.

For example, the `Entity` class inherits from `SSF_Entity`, which serves as a container for logical processes that generates and consumes events. The `Port` class is implemented by a pair of DaSSF `in-channel` and `out-channel`. To connect entity ports, the `Port` class provides `connection` and `disconnect` methods, that use themselves the `mapto` and `unmap` methods of `out-channel`. The `Parameter` and `Variable` classes have no correspondence in

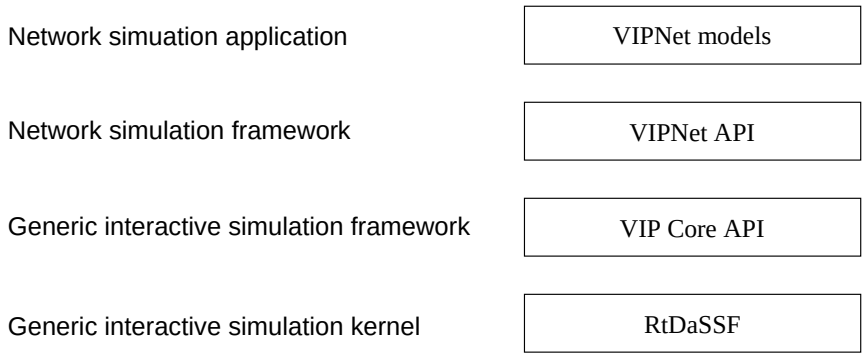

Figure 7.1: Structure of VIP-based simulation

DaSSF, but are necessary to support user interaction. `Parameter` represents the interface to accept user requests for configurational changes. `Variable` provides a means to deliver state update information to the user.

Behind this concise structure are complex mechanisms supporting the communication between the graphical user interface and the simulation applications. Key issues for the spatial distribution, as discussed in Section 4.3.1, are addressed. For the temporal distribution, the major part of the timing and control concepts presented in Section 4.4 are also implemented, in particular the control of simulation sessions.

The generic mechanisms provided at the VIP core API can be used by a variety of interactive simulations. VIPNet has been built as an example.

7.2 VIPNet – A Network Simulation

VIPNet is a network simulation based on the VIP simulation environment. More precisely, VIPNet is composed of an API for network simulations based on the VIP core API, and a collection of modules that assemble simulation applications. This concept is illustrated in Figure 7.1. The construction of the VIPNet API, separated from application specific models, is beneficial for the variability and extensibility of simulation models, as explained in the following.

7.2.1 VIPNet API

Conceptually, the VIPNet API adheres to the ISO-OSI 7-layer-structure for computer networks [Tan02a] [HAN99]. Network components are characterized by their protocols. A protocol describes the syntax and procedure for exchanging messages with its peers. At a service access point (SAP), a protocol may use the service of a lower layer protocol, and may provide service itself to an upper layer protocol. In the latter case, the protocol includes also specification about the form and procedure of service primitives.

The protocol concept is reflected in the structure of the VIPNet API, as illustrated in Figure 7.2. A UML-based specification is shown in Figure 7.3.

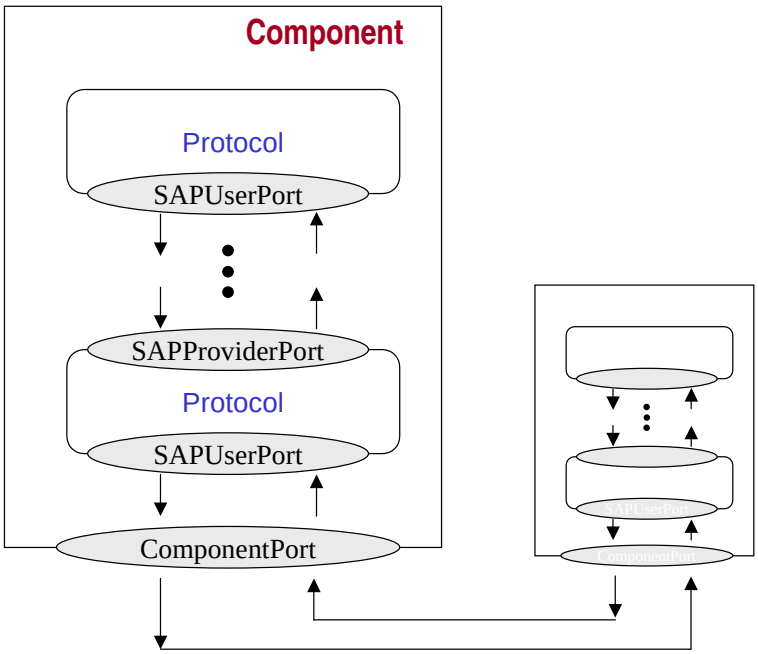

Figure 7.2: VIPNet Component

A `Component` represents a network component. It is structured into one or more `Protocol`. An SAP is represented by a pair of `SAPProviderPort` and `SAPUserPort`. A `Protocol` may have zero or more `SAPProviderPort` and/or `SAPUserPort`. To support complex protocol behavior, a `Protocol` may be structured into multiple `ProtocolSessions`. Timing functions make use of `ProtocolTimer`. `ProtocolMessage` is the base class to describe protocol messages. A `Component` may have one or more `ComponentPort`, each of them representing a connecting point to other `Component`. Semantically, a `ComponentPort` is mapped to an `SAPUserPort` of the lowest `Protocol`, which may be an abstraction of a protocol on any layer. In this manner, a `Component`, opposite to a real network component, needs not to assemble `Protocol`s or mechanisms down to the physical layer, if only the behavior of a higher layer is of the primary interest. The property is for example utilized by the simulation of routing functionality, as explained in the following sub-section.

7.2.2 Modules

Modularity is one of the major goals for the design of the VIPNet API. It is carried forward in the development of simulation models. A simulation model consists of configurable modules. There are two groups of modules: protocol modules, and component modules. A *protocol module* represents a network protocol, e.g. an IP module, or an RIP module. A *component module* is an abstraction of a network component, e.g. a host module, a cable module, a switch module, or a router module. Each of the component module makes uses of appropriate protocol modules. Doing so, VIPNet can be easily extended by developing new modules. In the following, the IP module, the router module, and the host module are briefly introduced.

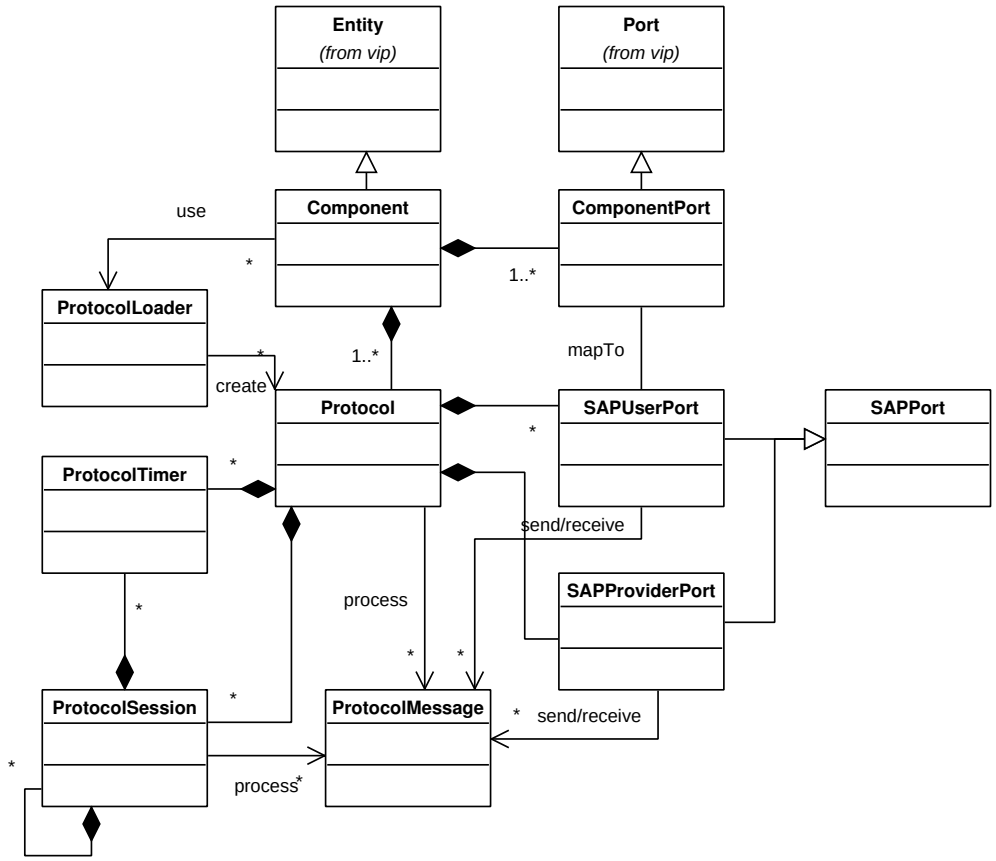

Figure 7.3: VIPNet API

IP Module

The structure of the IP module adheres to the concepts of the Internet protocol family [Com95]. The architecture of the Internet protocols is in-line with the 7-layer architecture of ISO-OSI, but consisting of four layers, with the focus on the internet layer and the transport layer. The internet layer comprises the Internet Protocol (IP) [Int81a] and a set of routing protocols, such as the Routing Information Protocol (RIP) [Int98b], and the Open Shortest Path First (OSPF) protocol [Int98a]. The Internet Control Message Protocol (ICMP) [Int81b] is used additionally to distribute control information. The transport layer consists mainly of the User Datagram Protocol (UDP) [Int80], and the Transmission Control Protocol (TCP) [Int81c].

The IP module covers IP, ICMP, and the routing protocols. The responsibility of IP includes mainly forwarding and reassembly of IP datagrams that are sent and received at network interfaces. IP mechanisms are used by ICMP, routing protocols, and transport protocols. Components implementing this concept are illustrated in Figure 7.4.

The functionality of the IP module can be outlined as follows:

1. `IPProt` is derived from `Protocol`. It provides a `SAPProviderPort` to modules implementing routing protocols (generalized by `RoutingProt`), and transport protocols

(e.g. UDPProt). A IPProt may have one or more SAPUserPort, each represents a network interface.

2. IPDatagram represents IP datagrams, and ICMPMessage represents ICMP messages. A IPDatagram contains the source address and the destination address of the datagram.

3. The essential features of IP are provided by a set of ProtocolSessions, including IPForward and IPReassembly, as well as one or more IPOverX or derivations from IPOverX.

4. IPOverX represents a generic network interface, from which specializations for concrete network types can be built, such as Ethernet. Each IPOverX instance can be associated with one or more SAPUserPort.

5. Information to be sent over IPProt is passed to IPForward, which examines the source and destination addresses, and constructs IPDatagrams to be delivered over IPOverXs.

6. When an IPDatagram arrives at an SAPUserPort, it is forwarded to the associated IPOverX, where the destination address of the datagram is examined. In case the datagram is targeted to one of the addresses specified for the IPProt instance, the datagram is passed to IPReassembly, which delivers reassembled information to upper protocols over the SAPProviderPort.

7. In case the destination addresse of the arrived IPDatagram does not match one of the own addresses of an IPProt instance, and the container of the IPProt instance is not a router, the datagram is discarded. Otherwise, if routing functionality is available, the datagram is passed to the RoutingProt instance, where the route for the datagram is determined. The datagram is then passed to the appropriate IPOverX for the route to be forwarded. RoutingProt is the routing protocol interface to IP. It implements by default static routes, which can be overwritten by mechanisms of derived routing protocol modules, e.g. by the RIP module.

8. ICMPHandler processes in this module ICMP messages indicating errors in IP datagrams.

Router Module and Host Module

The router module and the host module are directly derived from Component. According to Figure 7.2 and Figure 7.4, a router module is constructed with IPProt as the uppermost protocol, combined with RoutingProt or a derivation of RoutingProt as the routing functionality. The IPProt instance of a router module can operate stand-alone or based upon other lower layer functionality. It is noted, that RoutingProt or one of its derivations is in the current implementation only configurable as an option of IPProt, not as a separated protocol. This

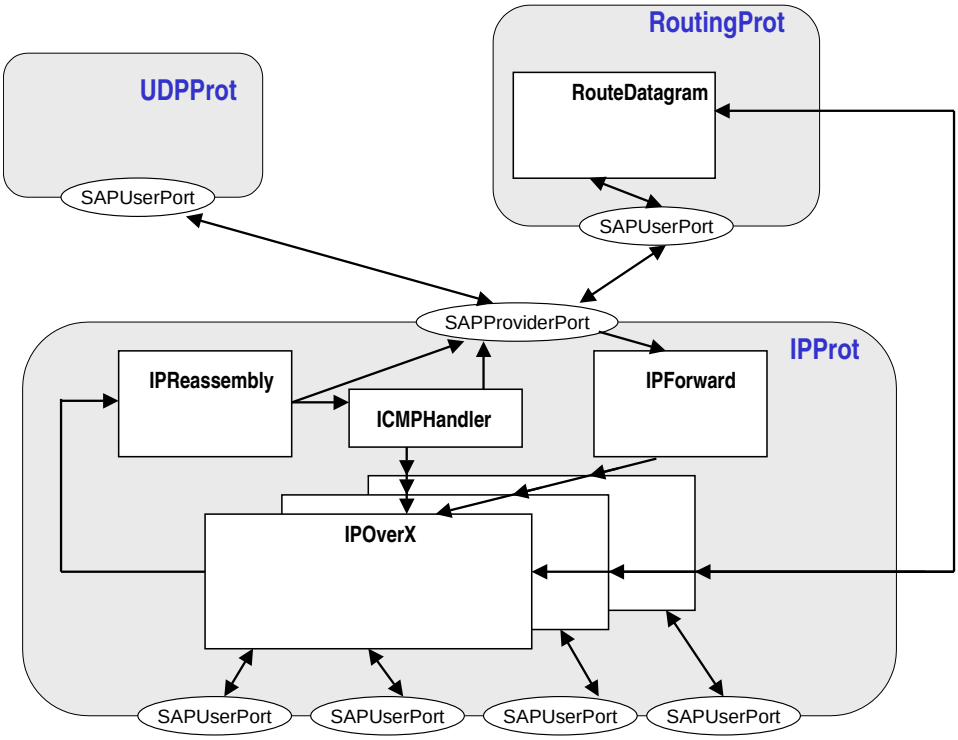

Figure 7.4: VIPNet IP protocol module: interworking between sub-components

is beneficial in particular for the configuration of interface identifiers and addresses. As shown in Figure 7.4, the separation of routing protocols from the basic IP functionality is conceptually already included.

A host can act as an initiator or a consumer of messages. Therefore, a host module is open for any protocol that initiates or consumes messages as the uppermost protocol. For example, an instance of a host module can be assembled by `PingProt` (generates and responses pings), `UDPProt` (representing UDP), and `IPProt`.

Care must be taken for the lowermost protocols of instances of the router module and the host module using in the same application. Principally, if they are connected by protocol-neutral components, such as instances of the ”transparent-cable” module, the lowermost protocols must be identical. Otherwise, the protocol messages exchanged are not compatible. The ”transparent-cable” module is an abstraction of cables in the real-world. It has two ends. Messages arriving at one end will be passed to the other end immediately without changing their content and format. Such a cable is useful to simulate a higher level protocol, such as IP.

7.3 XML-Based Simulation Database

The implementation of the simulation database is used for experiments in this work. It covers the very basic features but adheres to the concepts introduced in Chapter 5.

In principle, the database is a collection of XML-documents that conform to the schemas

presented in Appendix C.

The manipulation of the database XML-documents is based on the XML Document Object Model (DOM). DOM "is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents" [Wor04]. The `Node` interface is the core of DOM. After an XML-document is parsed, it is used to represent a hierarchy of DOM nodes. Using the `Node` interface, the type and content of a node, as well as its parent and child nodes, if existent, can be inquired or updated. The Apache [Apa] Xerces DOM parser is used in this work.

7.3.1 Components

Figure 7.5 illustrates the components of the database, that implement the creation, update, inquiry and storage of the data. As stated earlier, deletion of data is not considered by the data access interface because of the property of the database as a history database.

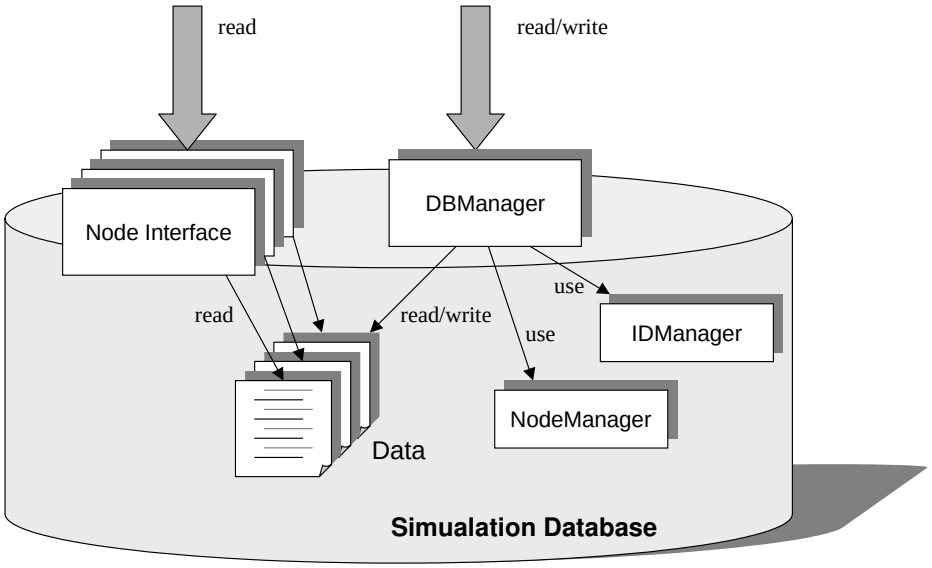

Figure 7.5: Simulation database components

DBManager represents the initial and the write access interface of the database. It loads the specified schemas to validate the data and provides a set of operations (see Figure 7.6) to create and to update data nodes. *IDManager* serves DBManager with the creation and inquiry of node identifiers. It implements the integrity constraints for entity identifiers (Section 5.2.2). For example, "`session1:instance1:entity3_Host3`" is the identifier of an entity, which represents a host component created for the simulation instance named `instance1`, that is again contained in the simulation session named `session1`. *NodeManager* provides the creation and update of nodes and node values, aligned to the generic schema. This interface can be extended for specific schemas. *Node interface* inherits from the DOM `Node` interface. It represents the read access interface to the data. The detailed structure of this interface is presented

in Section 7.3.2. *Data* represents a collection of XML-documents storing desired simulation history data.

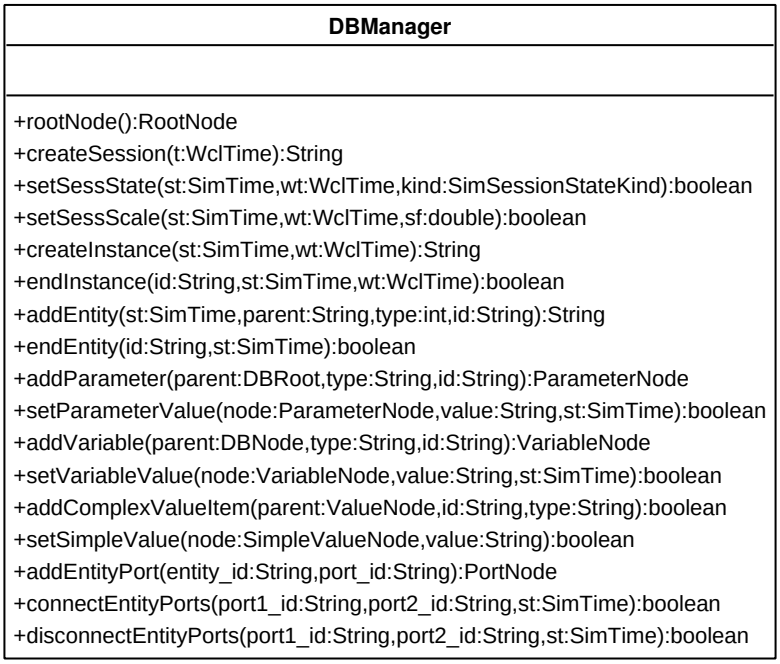

Figure 7.6: Database write access interface

7.3.2 Data Nodes

For the read access of data contained in the stored XML-documents, a set of node interfaces is defined. The root of the inheritance hierarchy is `DOMNode` – the interface of the DOM core. Basically, every node interface in the lower inheritance levels corresponds to an element type or a complex type in the generic schema (please refer also to Figure 5.1). For inquiry on attributes or contained elements, a node interface provides appropriate operations.

The read and write access interfaces of the database are integrated with the VIP core API, in order to generate data at the run-time of a simulation. On the other side, the write access interface generates also events to trigger active rules. The latter point is further elaborated in the following section.

7.4 VTIS Controller

According to Figure 3.5, the controller is the core component for the processing of active rules implementing teaching strategies. The realization of the controller is aligned to the execution model represented in Figure 6.3. A refined architecture is shown in Figure 7.7, with the focus on the interworking between the controller, the simulation server, the simulation database, and the assistance interface at sites of users.

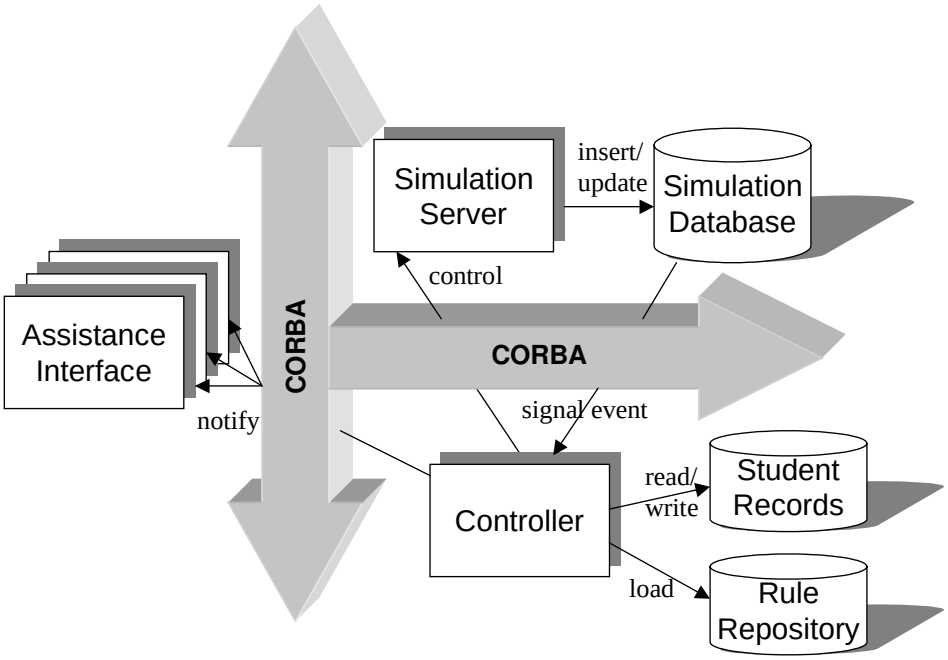

Figure 7.7: Implementation of active rule processing

This architecture relies on the middleware CORBA, which is also used for the VIP simulation environment supporting distributed simulation clients (Section 7.1.1). This architecture is applied in particular to the processing of active rules during the simulation execution. While simulation models are executed on the simulation server, data is inserted or updated in the simulation database. The insert or update operations cause signaling events being generated and delivered to the controller. They result in three kinds of actions after the three-stage processing with event detector, condition monitor and action executor (see also Figure 6.3). One kind are control actions that are executed with the simulation server, e.g. session control actions. The second kind are notifications that are delivered to users over the assistance interface. The third kind produces student records. Thanks to CORBA, the first two kinds of actions can be easily integrated in the VIP simulation environment.

In case the event part of rules is absent or ignored, the rule processing is decoupled with the evolution of the simulation database. Offline assessment supported teaching strategies can be realized in this manner.

In the following, some details to the rule composition, the event signaling interface, and the rule processing components are presented.

7.4.1 Rule Composition

The rule template presented in Section 6.3 is compatible to the grammar of XQuery. Each part of a rule, either the event part, the condition part, or the action part, is composed of a keyword and XQuery expressions. Therefore, rule parts can be easily evaluated by an XQuery parser. For experiments in this work, IPSI-XQ [Fra], an XQuery parser developed by Fraunhofer IPSI, is

used.

In addition, a set of user-defined functions and external functions supporting rule specification have been developed. The specification of the functions is presented in Table 6.3. The function declarations are included in a rule prolog which is directly used for every evaluation of XQuery expression, so that a rule developer is able to concentrate on the essence of rule logics. Please refer to Appendix D for the complete specification of the rule prolog. The user-defined functions are specified using XQuery expressions. The external functions are declared in the rule prolog and implemented using a programming interface of IPSI-XQ for external functions.

7.4.2 Event Signaling

The event signaling interface of the controller is defined by the following IDL interface. The method `signalEvt` is used by DBManager of the database every time an *insert* operation or an *update* operation is executed. The parameter `evtinfo` contains the kind of the operation and a path expression describing the nodes involved. The method `updateRules` notifies the controller that the rule repository has been probably renewed, on which the controller can fetch updated rules.

```
// IDL definition of controller's event signaling interface
module vtis{
  interface Controller{
    oneway void signalEvt(in string evtinfo);
    oneway void updateRules();
  };
};
```

The IDL interface is implemented as part of the controller as a CORBA server, using the programming language Java. The CORBA client using this interface is currently embedded in DBManager of the database, because all *insert* and *update* data manipulation operations are executed over DBManager. An alternative implementation may use the DOM Event interface, which monitors directly the changes on document nodes. The Event interface is however not available in the currently used DOM parser.

7.4.3 Components

The rule processing within the controller is performed by three sub-components: EventDetect, ConditionEval, and ActionExec. They are implemented using the Java Thread feature, so that the processing of rules on signaling events scales well.

To execute actions as discussed at the beginning of Section 7.4, ActionExec uses the CORBA Notification service [Obj02c] to distribute notifications, and the CORBA interface of the simulation server to execute session control actions. The student records are stored as XML-documents.

7.5 Summary

The implementation of the VTIS approach relies on the distributed simulation environment and the network simulation framework, that have been developed in the VIP-project. Additional components supporting the simulation database and the controller are integrated in the distributed simulation environment using a CORBA interface.

The implemented features are used for a comprehensive example to demonstrate the VTIS approach. The example and a plan for evaluating the approach in the practice are discussed in the following chapter.

Chapter 8

Evaluation

An evaluation of the VTIS approach in practice is planned for the up-coming summer semester, namely, in a practical course for computer science. This practical course was offered for the first time in the previous summer semester as part of the VIP-project.

In the following, the content and organization of the practical source are briefly described. An example task for the practical course demonstrates how the concepts and tools of VTIS can be applied.

8.1 Practical Course

The practical course is titled ”Construction and Management of Computer Networks”. The goal of this course is to deepen the knowledge about computer networks, when doing planning, deployment and management of communication infrastructure for a fictive company. It is a joint course carried out by two universities: the Ludwig-Maximilians-University of Munich (LMU), and the Technical University of Aachen (RWTH-Aachen). That is, students and tutors from Munich and Aachen participate in the same course.

The organizational concepts for this course have been developed in the VIP-project. It is a result of the cooperation of computer science and pedagogy professionals. The essential ideas are outlined in the following. Please refer to [LLPM+03a] for further details.

- *Students build consulting teams.* To install an authentic situation, the students participating in the course play consultants in teams. Each team should provide a complete solution for the fictive company. Technical aspects should be considered together with potential costs. The teams sell their solutions in two presentations.

- *Distributed presence phase and self-learning phase.* The presentations mentioned in the previous point take place in the so-called ”distributed presence phases”. At the time of a distributed presence phase, students and tutors from different locations join together in a video-conference. The course is composed of three distributed presence phases, respectively at the beginning, in the middle and at the end of the course. They interleave with the self-learning phases, in which each team works individually on its solution. As members

of a team can be either in Munich or in Aachen, a distributed collaboration environment is used. It consists of the VIP simulation environment as elaborated in the next point, and an ”e-learning platform”. The latter refers to a content-neutral tool that organizes the distribution and the management of e-learning material, such as text documents, slides or video-clips. Usually such a tool also provides features for personal communication. Email and chat tools are for example popular built-in features. For the course in the previous summer semester CVW (Collaborative Virtual Workspace) [The] has been used as an e-learning platform.

- *VIP simulation environment as planning, deployment and management tool.* This part of the distributed collaboration environment particularly supports the self-learning phases. This environment includes simulation models based on the network simulation framework VIPNet.

- *Tele-tutors supported by VTIS-based virtual tutor.* It has been shown in the previous summer semester, that human tutors are very helpful when first experience with e-learning tools is collected. Tutors, either in Munich or Aachen, provide support for students using the e-learning platform. For some tasks, the human tutors are relieved by VTIS-based virtual tutoring functions. This is demonstrated by an example in the following section.

8.2 Example

This section presents an example task for the practical course performed in the VTIS-based learning environment. The object is to specify static route entries for routers that support several sub-nets, as illustrated in Figure 8.1. In the following the basics for this task are first introduced.

8.2.1 Basics

Static routes are in some cases useful, even when automated routing, for example by RIP, is used. The manual specification of route entries in this task helps students to understand the construction of sub-nets, in particular what network addresses, subnet masks and default routes are concerned. These concepts are briefly introduced in the following. Details can be found in a number of books for IP-based computer networks, e.g. [Tan02a] [Com95] [McC98].

The internet address, or **IP address**, is one of the very basic concepts of IP-based networks. An IP address identifies a host uniquely in a network. An IP address (IP version 4 is considered here) is usually written using the ”dotted decimal notation”, i.e. four integers that are separated by dots, for example `192.24.10.5`. Each integer is represented by one octet, so that totally 32 bits are used for such an address. The above example is in binary format: `11000000 00011000 00001010 00000101`. The bits can be divided into a *network portion* and a *host portion*. In an earlier concept, the network portion consisted of fixed number of bits. In this manner, class A, B and C addresses have been defined. The network portion of a class A address is 8 bit long, of a class B address 16 bit long, and of a class C address 24 bit long. The rest of the bits are reserved for host identifiers. According to this classification, class A, B, or C

networks are distinguished. This concept was extended later by subnetting (see below) to solve the problem of address space exhaustion.

Routing refers to the process of choosing a path over which to send IP packets. This process is carried out by a *router*. A router manages a *routing table* which contains *routes* describing how to reach potential destinations. Using the network portion in IP addresses helps to keep the routing table in a manageable size. Typically, a route is described by a pair (N, R), where N is the IP address of a destination network or host, and R is the IP address of the next router along the path to N, called "next hop". A packet is forwarded "hop-to-hop" until the last router which delivers the packet directly to the target host. Therefore, the setting of routes is crucial for correct and efficient delivery of packets. Routing protocols such as RIP and OSPF provide automated update of routes. *Static routes* are usually set by network administrators for specific networks or hosts. They are not affected by automated route update mechanisms.

A 32-bit **network mask** is used to determine the network portion of an IP address. Bits in the mask set to *one* indicate the network part, and those set to *zero* the host part. The network mask is applied to the target address of a packet to be routed using bitwise-AND.

Due to the fixed number of bits for the network portion, the number of networks was quickly exhausted using the ABC-address-scheme. Therefore the concept of **subnetting** has been introduced. In subnetting, an address is divided into an *internet portion* and a *local portion*. The internet portion identifies a site, which is free to interpret the local portion. In this way, hierarchical addressing and routing can be achieved using hierarchical subnets. A subnet mask is usually written using "dotted decimal notation", for example, `255.255.248.0`. The length of bits setting to `1` can be explicitly indicated, which is `21` for this example. Sometimes this subnet mask is represented as `255.255.248.0/21`. In general, for routes a subnet address is specified together with a subnet mask. Supernetting is the inverse of subnetting. It allows a block of contiguous class C address, instead of a single class B address, to be assigned to an organization. Such a block is specified by the lowest address and a 32-bit mask, which operates like a subnet mask. This technique is known as Classless Inter-Domain Routing (CIDR).

8.2.2 Scenario

The scenario for the example task consists of three routers, named Router1, Router2 and Router3. They manage the packet forwarding for a hierarchical structure of sub-nets. The hosts Host1, Host2, Host3 and Host4 serve as sender and receiver of test packets to prove the reachability.

Router1 is configured with three network interfaces. Interface 1 is connected to the subnet Net1 that is represented by Host1. The IP-addresses of interface 1 of Router1, Net1 and Host1 can be found in Figure 8.1, likewise for other hosts, routers and sub-nets.

The second interface of Router1 is connected to Net2. The address spaces of Net1 and Net2 are subsets of NetA. The construction of Router2, Net3 and NetB can be explained analogously.

Router3 represents a uplink to other networks that are not explicitly depicted in the scenario. The interface 0 of Router3 is connected to the interface 0 of Router1. The other interface of Router3 is connected to the interface 0 of Router2.

The task consists in the specification of the routes for Router1, Router2 and Router3. A correct setting is shown in Table 8.1. This table includes routes for all three routers in a compressed

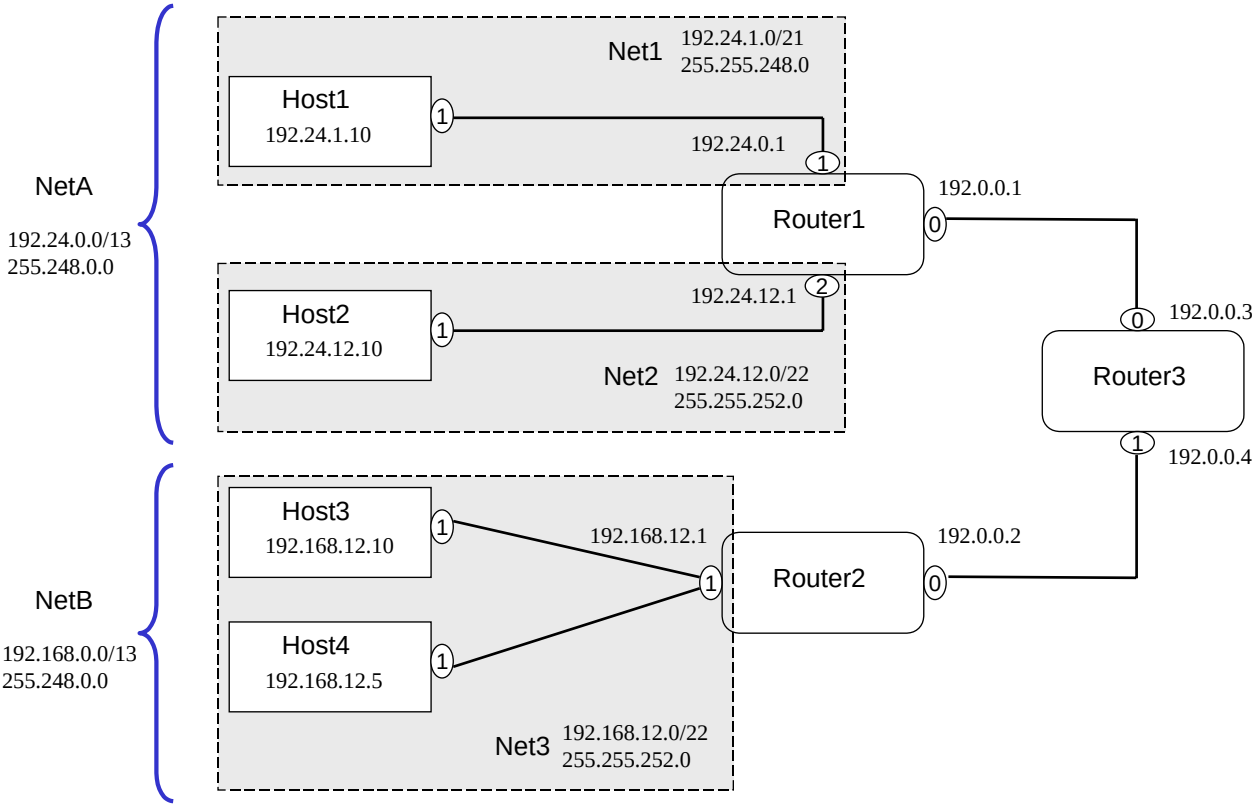

Figure 8.1: Example of practical course

form. In fact, only the segment that concerns its own routes is presented to a router.

Every route entry in the table consists of four elements, namely the IP address of the potential destination network or host, the subnet mask of this network or host, the next-hop IP address to forward packets targeting this network or host, and the appropriate outgoing interface. The next-hop address is set to ”–” for directly connected networks, e.g. Net1 at Router1. The last entry for each router is a default route, in case none of the other explicit routes matches the destination address contained in a packet.

The settings for Router1 and Router2 are rather straight-forward, while for Router3 some crucial points must be taken into account:

1. The address spaces of Net1 and Net2 cover only a subset of the address space of NetA. Therefore, not NetA, but Net1 and Net2 are explicitly specified as potential destinations. Otherwise, Router3 would forward all packets targeting NetA to Router1, which would find no matching routes on its table and sends the packet to the default route. As the default route of Router1 is Router3, and the loop would go on and on. A similar situation would happen to packets targeting NetB, if this is included as potential destination, instead of Net3.

2. The interface 0 of Router1 and Router2 are described by extra routes, because the interface addresses are not included in any of the indicated subnets.

3. Router3 does not have default route. Since the default routes of Router1 and Router2 points to Router3, any packets that can not be delivered by Router1 and Router2 will be discarded at Router3.

Router	IP Address	Subnet Mask	Net-Hop IP Address	Interface
Router1	192.24.1.0	255.255.248.0	–	1
	192.24.12.0	255.255.252.0	–	2
	default	–	192.0.0.3	0
Router2	192.168.12.0	255.255.252.0	–	1
	default	–	192.0.0.4	0
Router3	192.24.1.0	255.255.248.0	192.0.0.1	0
	192.24.12.0	255.255.252.0	192.0.0.1	0
	192.168.12.0	255.255.252.0	192.0.0.2	1
	192.0.0.1	255.255.255.248	–	0
	192.0.0.2	255.255.255.248	–	1
	default	–	–	–

Table 8.1: Static routes of Router1, Router2 and Router3

The routes are validated by test packets sending between the four hosts. The test packets are ”one-way” packets, that are consumed by the target hosts, if found, and are discarded by non-target hosts. In principle, a ping simulation application, as the one included in VIPNet, applies well to the test. Ping is used frequently in real networks to test the reachability and status of a destination. It uses ICMP echo request and echo reply packets. Nevertheless, to keep the example simple, and to demonstrate rule-based data analysis, one-way test packets are used here.

In the following, two representative cases are considered:

Case 1 – known subnet address: A test packet targeting a known subnet address is sent by Host1. Host3 is taken as the example target. That is, the source address of the packet is `192.24.1.10`, and the destination address is `192.168.12.10`. If the routes are set correctly as shown in Table 8.1, a packet sent by Host1 arrives first at Router1, which does not find a matching route but uses the default route to forward the packet to Router3. Router3 determines that the packet can be delivered over Router2, on which it passes the packet to Router2. At Router2, the destination address of the packet matches the address and mask of Net3, so that the packet is passed to the indicated interface 1, from which the packet arrives finally at Host3. A packet with a destination address, that is other than Host3 or Host4, but in the address space of Net3, would be also delivered by Router2. It would be never accepted by any host.

Case 2 – unknown subnet address: A test packet targeting an unknown subnet address, for example, 192.24.10.10, is sent by Host1. If the routes are set correctly as shown in Table 8.1, a packet sent by Host1 is forwarded by Router1 to Router3, because the default route is the only match found at Router1. Router3 does not have any matching route, neither a default route. Therefore, it discards the packet.

8.2.3 Configuration of Simulation Model

The simulation model for the scenario consists mainly of three router module instances and four host module instances. They are identified as shown in Figure 8.1. Each of them is configured with the IP module. The hosts are configured additionally with a module that generates and consumes test packets. Instances of the cable module are used to link the routers and hosts.

As described previously, Host1 is configured as the sender of test packets. In order to observe how test packets are routed in the subnets, the hosts and routers are assigned packet counters, either counter for sent packets, or counter for received packets. In terms of the conceptual simulation model, the packet counters are variables of the belonging entities, which are in this case either hosts or routers. For entities with multiple interfaces, such as a router, a variable for a sent packet counter is identified as `<InterfaceName>_sentPacketCounter`, and a variable for a received packet counter as `<InterfaceName>_rcvdPacketCounter`, where `<InterfaceName>` will be replaced by concrete interface names.

8.2.4 History Data

A segment of an XML-document of the simulation database that records an execution of the example simulation model is presented below. It gives an impression of the construction of such a document, and prepares for the next section of data analysis.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<DBRoot xmlns="http://www.lmu.de/vtis"
dataPath="/proj/vip/workdir/bkspace/VIP-BK/sim/c++/net/app/tests/ssdb/data/"
id="vg1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.lmu.de/vtis sessions.xsd">
  <wlife><start>Tue Feb 17 18:38:47 2004</start> <end>Tue Feb 17 18:38:512004</end></wlife>
  <SimSession id="session1">
    <wlife><start>Tue Feb 17 18:38:47 2004</start><end>Tue Feb 17 18:38:51 2004</end></wlife>
    <state>
      <entry index="1"><sTime>0</sTime><wTime>Tue Feb 17 18:38:47 2004 </wTime>
        <value>Init</value></entry>
      <entry index="2"><sTime>0</sTime><wTime>Tue Feb 17 18:38:47 2004 </wTime>
        <value>RunRecord</value></entry>
      <entry index="3"><sTime>3000</sTime><wTime>Tue Feb 17 18:38:51 2004 </wTime>
        <value>Stopped</value></entry></state>
    <scale>
      <entry index="1"><sTime>0</sTime><wTime>Tue Feb 17 18:38:47 2004 </wTime>
        <value>1</value></entry></scale>
    <SimInstance id="session1:instance1">
      <slife><start>0</start><end>3000</end></slife>
      <wlife><start>Tue Feb 17 18:38:47 2004 </start>
        <end>Tue Feb 17 18:38:51 2004 </end></wlife>
      <entity id="session1:instance1:entity1__Host1" type="Component:Host">
        <slife><start>0</start><end>3000</end></slife>
        <entity id="session1:instance1:entity1__Host1.1" type="Protocol:IPApp">
          <slife><start>0</start><end>3000</end></slife>
          <parameter id="sendRate" type="ulong">
            <entry index="1"><sTime>0</sTime>
              <simpleValue>1</simpleValue></entry></parameter>
          <parameter id="src" type="string">
            <entry index="1"><sTime>0</sTime>
              <simpleValue>192.24.1.10</simpleValue></entry></parameter>
          <parameter id="dsts" type="vector">
            <entry index="1"><sTime>0</sTime>
```

```
          <complexValue><item id="0" type="String">
            <simpleValue>192.168.12.10</simpleValue></item>
            </complexValue></entry></parameter></entity>
      <entity id="session1:instance1:entity1__Host1.2"type="Protocol:IPProt">
        <slife><start>0</start><end>3000</end></slife>
        <parameter id="itfSpec" type="vector">
          <entry index="1"><sTime>0</sTime>
          <complexValue>
            <item id="0" type="Struct"><complexValue>
              <item id="itfAddr" type="Vector"><complexValue>
                <item id="0" type="Struct"><complexValue>
                  <item id="addr" type="Vector"><complexValue>
                    <item id="0" type="Struct"><complexValue>
                      <item id="ipAddr" type="String">
                        <simpleValue>192.24.1.10</simpleValue></item>
                      <item id="sbMask" type="String">
                        <simpleValue>255.255.248.0</simpleValue></item>
                    </complexValue></item></complexValue></item>
                  <item id="iscd" type="Struct"><complexValue></complexValue></item>
                  <item id="itfID" type="String"><simpleValue>Itf1</simpleValue>
                  </item></complexValue></item></complexValue></item>
              <item id="itfCf" type="Struct"><complexValue></complexValue></item>
              <item id="itfType" type="String">
                <simpleValue>NilProt</simpleValue></item>
              </complexValue></item></complexValue></entry></parameter>
        <variable id="NilProt_Itf1_sentPacketCounter" type="ulong">
          <entry index="1"><sTime>0</sTime>
            <simpleValue>0</simpleValue></entry>
          <entry index="2"><sTime>1000</sTime>
            <simpleValue>1</simpleValue></entry>
          <entry index="3"><sTime>2000</sTime>
            <simpleValue>2</simpleValue></entry>
          <entry index="4"><sTime>3000</sTime>
            <simpleValue>3</simpleValue></entry>
          </variable></entity>
    <port id="session1:instance1:entity1__Host1:port1__NilProt_Itf1">
      <connection index="1" inverse="session1:instance1:entity8__Link1:port1__Port1">
      <slife><start>0</start><end>3000</end></slife></connection>
      </port></entity>
...
```

This segment is about 15% of the entire data for a three-second-execution with a packet send rate of 1 packet pro second. Obviously, the amount of data increases quickly with the complexity of the configuration. The performance aspect is one issue to be considered in future work (see also Chapter 9).

This segment includes the history data about the simulation session named `session1`, its simulation instance `session1:instance1`, as well as the entity

`session1:instance1:entity1__Host1` that represents Host1. That is, concatenations are used for the identifiers. To simplify, in the following, only the local part is referred to. The entity `Host1` is configured with two sub-entities, `Host1.1` represents the packet generating module, and `Host1.2` the IP module. Each of them contains a certain number of parameters. `sendRate` is for example a parameter of `Host1.1` to set the packet send rate. The parameter structure for `Host1.2` is rather complex because of the complex interface specification structure. A variable of `Host1.2` is `NilProt_Itf1_sentPacketCounter`. As stated above, it describes the counter of sent packets at the interface named `Itf1`. `NilProt` is added for the IP module to identify the type of the interface that simulates a network interface, which sends and receives in this case directly IP packets.

8.2.5 Rule-Based Data Analysis

A simple rule as presented below demonstrates the benefit of rule-based data analysis.

```
rule "R1"{
  event:
    insert
    vtis:rootNode()//vtis:variable[fn:ends-with(@id,"sentPacketCounter")]
  action:
    <msg text="The following entities received probably packet
    from the sender host:  {
    (:  time last packet was sent :)
    let $tm := max( for $var in vtis:rootNode()//vtis:variable
    where(fn:ends-with($var/@id, "sentPacketCounter"))
    return $var/vtis:entry/vtis:sTime/text()
    )
    (:  entries of received-packet-counters :)
    let $i := (
      for $var in vtis:rootNode()//vtis:variable
      where(fn:ends-with($var/@id, "rcvdPacketCounter"))
      return $var/vtis:entry
    )
    (:  find entities, whose last received packet counter entry
    matches the packet-send-time :)
    let $id := (
      for $j in $i/vtis:sTime/text()
      where($j >= $tm)
      return
        $j/parent::node()/parent::node()/parent::node()/parent::node()/@id
    )
    return $id
  }"/>
};
```

The rule consists of an event part and an action part. The condition part is omitted. The rule can be applied both for off-line and on-line data analysis. For off-line data analysis, the event part is not significant.

The event part defines the triggering of the rule in case of on-line analysis by an insert operation. The insert operation is caused by the increment of the sent packet counter of an entity, which is in this example Host1.

The action part describes the construction of a message that indicates all the entities that have probably received the last packet from Host1. Expressions used in this part are explained by short comments. First, the simulation time for the last packet that has been sent by Host1 is determined. This time is compared with the time associated with the latest entry of the variable of received packet counter. In case that the packet send time matches the packet receive time, the ID of the entity owning the variable of received packet counter is selected for a message, that is delivered as a notification.

Applying the rule to the history data generated for Case 1 of Section 8.2.2, the controller will produce the following result:

```
Message content:
  "The following entities received probably packet from the sender host:
  session1:instance1:entity3__Host3.2
  session1:instance1:entity5__Router1.1
  session1:instance1:entity6__Router2.1
  session1:instance1:entity7__Router3.1"
```

From the message content, the identification of the involving entities can be easily resolved. The order of the entities printed out depends on the order when the entities have been created. It has no direct co-relation to the path of the packet.

Analogously, the result for Case 2 will look like the following:

```
Message content:
  "The following entities received probably packet from the sender host:
  session1:instance1:entity5__Router1.1
  session1:instance1:entity7__Router3.1"
```

8.3 Summary

The example presented in this chapter serves the evaluation of the VTIS concepts and tools in the practice. Although the scenario is a very basic one, some significant benefit can be demonstrated.

One point is that thanks to the modular structure of VIPNet, a simulation model can be easily assembled by configuring the module.

Secondly, the history data is represented in a human-readable form. It can be examined with conventional text editors if saved in files. This property is useful for experiments to setup rules. Although, with increasing complexity of the simulation model, the readability is difficult to retain. At this point, it becomes clear that the rule-based data analysis can be very useful to manage the complexity. In addition, the temporal data model and temporal queries make it possible to find the matching data of objects, that not only are located on different levels of the logical hierarchy, but also have different lifespans. Recalling the discussion on integration approaches in Section 3.1, doing so using a behavior-oriented approach would be much more complicated.

Chapter 9

Conclusions and Outlook

This chapter closes the thesis with a summary of contributions, and an outlook over future work.

This thesis is devoted to the development of modern e-learning systems, which are generally referred to as Learning Technology Systems (LTSs). Such an e-learning system is no longer only a collection of multi-media documents, but ”a designed information space, where educational interactions take place” [Dil00]. Great effort has been put into the design of LTSs in the last few years. Major requirements placed on LTSs include support for hypermedia, construction-oriented learning, and learner-centered environments.

The thesis focuses on one aspect of LTSs, whose importance has been recognized just recently, namely the construction and the integration of **virtual teachers** in LTSs.

It has been shown in case studies, that the lack of adequate teaching support is a crucial problem for the acceptance of information technology-supported learning. Virtual teachers provide a promising means to improve this situation. A virtual teacher automates certain functions a human teacher usually performs. Once a virtual teacher has been created, it allows greater flexibility at less expenses.

The conception and the implementation of a virtual teacher must be considered in a concrete technological context, which is for this thesis **e-learning capable simulation**. The contributions of this thesis are summarized in the following.

Summary of Contributions

A comprehensive set of requirements on e-learning capable simulations has been derived from pedagogical concepts for situated learning. The core of situated learning consists basically of two aspects, namely, authentic learning contexts, and teachers as supporters. For the first aspect, an e-learning capable simulation is required to provide the representation of domain knowledge, as well as an interactive and distributed collaboration environment. The second aspect is concerned with the integration of virtual teachers in an e-learning capable simulation, that is, the formalization of teaching strategies, which model the pedagogical handling of human teachers, as well as the implementation of teaching strategies, that consists in the on-line and off-line assessment about students, and the accomplishment of assisting actions based on the assessment. A review of related work has shown that none of the known approaches addresses the complete

set of requirements.

This thesis covers the various aspects of the requirements with a data-oriented integration approach, called **VTIS (Virtual Teaching Interface for e-learning capable Simulations)**. Concerning the integration of virtual teachers into LTSs, ”separation of learning content from control logic” is the principle followed by the majority of the recent works. It is also the basic principle of the VTIS approach presented in this thesis, whereas ”learning content” refers to the domain knowledge represented by simulation models, and ”control logic” refers to teaching strategies that assemble the functionality of virtual teachers. In contrast to existing approaches, that are classified as behavior-oriented integration approaches, VTIS relies mainly on the history data representing simulation executions. This new concept allows greater adaptivity and extensibility of an e-learning capable simulation over a looser coupling of teaching strategies with simulation models. This concept is also the foundation for a teacher-friendly development interface. VTIS comprises of the following three conceptual parts:

1. **Interactive simulation for virtual user groups.** This part is concerned with the provision of an interactive simulation based shared learning context, that can be spatially and temporally distributed, so that students are able to collaborate in the virtual space from different locations and at individual time schedules. To organize the spatial distribution of such a learning environment, the concept of *simulation environment unit* is introduced. It is reflected in a centralized simulation server architecture. The temporal distribution is provided by the concept of *simulation session*. A simulation session represents the control view of a simulation execution. Mechanisms supporting the control have been carefully studied. Investigation in this part prepares for the modeling of the simulation history data. A conceptual model for simulation applications presented in this part serves the same purpose. This conceptual model provides elementary constructs describing the structure of a simulation model.

2. **Temporal simulation database.** The focus of this part is the modeling of the history data for simulation executions. Basic concepts of temporal databases have been extended to include particular properties of interactive simulations, which are mainly due to the co-relation between the simulation time and the wallclock time. These two time dimensions are incorporated into a generic database structure, that is based on the conceptual model for simulation applications and the control mechanism for simulation sessions. The temporal simulation database provides input for the automated processing of teaching strategies.

3. **Teaching strategies as active rules.** In order to allow teaching strategies to be applied aligned to a simulation execution, they are modelled as active rules that are sensitive of modifications of the history data stored in the simulation database. Thanks to the data-oriented approach, the alignment to simulation executions is not mandatory but an optional feature. By varying the rules, alternative teaching strategies can be easily adapted. The rule template defined is based on the event-condition-action rule pattern. It relies on a data query language and assembles the core of the development interface for teachers. In addition, data queries can be made not only across the structural boundaries of objects in

a simulation model, but also alongside the temporal dimension of the objects. This is a further benefit of the data-oriented approach.

The concepts presented in the thesis have been implemented for an evaluation in practice. The evaluation should be embedded in a practical course that will be organized by the two universities: LMU Munich and RWTH Aachen. An example task for the practical course demonstrates how the VTIS concepts and tools can be applied.

Outlook over Future Work

The VTIS approach has been intended as a generic framework for e-learning capable simulations. The data-oriented integration of virtual teachers is a new concept in this area. Although its strength has been already shown by an example included in the thesis, extensive evaluation of the approach in the practice is still necessary. The evaluation may include the following aspects:

- Network simulations have been considered as examples for the development of the conceptual model for simulation applications. The feasibility of this model representing other types of domain knowledge can be examined in further studies.
- Experience with the development interface for teachers, that consists mainly of a query language based rule template, needs to be collected in more complex case studies.
- More complex application scenarios are also useful to study the performance related properties of the prototype developed in this work.

Some reference models from known e-learning initiatives have been reviewed in the thesis. These are rather general purpose models covering a broad range of technologies. The VTIS approach can be seen as a specialization using simulation as the core technology. Its integration with the reference models is worth to be considered in future work.

With the temporal data model and the rule-based control mechanism, VTIS provides a solid basis for the application of further knowledge-based techniques, e.g. various reasoning schemes of artificial intelligence, in order to achieve more advanced virtual teachers.

The idea of rule-based analysis of history data can be also considered for other application areas. For example for the management and testing of computer systems, incorporating the temporal aspect of the system data can be useful for the study of dynamic properties of a system. Doing so, some issues concerning the preparation of the system data must be carefully considered. One issue is the openness of the system to be studied for the modeling and the collection of the data. A second issue is that a centralized server architecture can not always be assumed, so that the distributed nature of the system data must be taken into account. The research on policy-based management has already shown promising results, that can be included in the consideration.

Appendix A

Mathematical Notations

This appendix includes mathematical notations used in the thesis. Sources of this review are [Sip97], [Nis99] and [NN95].

A very basic concept is **set**. A set contains mutually distinguishable elements, enclosed in curly braces $\{...\}$. Elements of a set are not ordered.

Comparing to a set, a **tuple** is an arrangement of certain elements of a set in some order. Elements of a tuple are enclosed in parentheses (brackets). According to the number of elements contained in a tuple, the tuple is called a pair (two elements), a triple (three elements), a quadruple (four elements), and so on.

The **Cartesian product** for two sets A and B is defined as:

$$A \times B = \{(a, b) \mid a \in A \wedge b \in B\}$$

That is, $A \times B$ is a set of elements, each of them is a pair. The first element of the pair is an element of A, and the second is an element of B. The order of A and B is significant, that is, $A \times B \neq B \times A$, unless $A = B$. In this manner, Cartesian product can be applied to any number of sets, or to a set any times. The latter is referred to as k-fold product. The k-fold product for the set A is defined as $A^k = A \times A \times A \times ... \times A$ (k times).

A **relation** is defined as a set of tuples, all belonging to the same Cartesian product. A binary relation from a set A to a set B is defined as:

$$R \subseteq A \times B$$

It states that R is a subset of the Cartesian product $A \times B$.

Domain and *range* are two operations applied to a relation to determine the elements participating in the relation. The *domain* of the relation R as described above, denoted as $dom(R)$, returns the elements of A that take part in R, that is,

$$dom(R) = \{x \in A \mid \exists y \in B \bullet (x, y) \in R\}$$

The symbol $\bullet$ is a separator used between the quantifier $\exists$ (means "there exists") and the variable x on one hand, and the formula $(x, y) \in R$ on the other hand.

Analogously, the range of R, denoted as $ran(R)$, returns the elements of B that take part in R, that is,

$$ran(R) = \{y \in B \mid \exists x \in A \bullet (x, y) \in R\}$$

The inverse of R, denoted as R^{-1} is defined as:

$$R^{-1} = \{(y, x) \bullet x \in A; y \in B \mid (x, y) \in R\}$$

Relations can be visualized by directed **graph**s. A graph consists of *nodes* and *arcs* [Nis99] (or *edges* [Sip97]). An arc in a directed graph points to a certain direction. In such a graph, a node represents an individual appearing in the relation, and an arc associates two nodes as defined by the relation. There are different representation forms for graphs. Usually, a node is represented as a small circle, and an arc as a line (which is an arrow in case of directly graph) linking circles.

A **function** is a relation which associates a unique element in its range for every element in its domain. A function *f* from *A* to *B* is denoted as:

$$f : A \rightarrow B$$

Depending on the range and domain of a function, it can be classified as:

- *Partial*: $dom(f)$ is a subset of A.
- *Total*: $dom(f)$ is equal to A. In many other definitions of function, a function refers to a total function.
- *Surjective*: $ran(f)$ is equal to B. A surjective function can be partial or total.
- *Injective*: the inverse of f is also a function. An injective function can be partial or total.
- *Bijective*: both total surjective and total injective.

Appendix B

MOF-Based Metamodel

This appendix provides a complete specification of the MOF-based metamodel for representational data schema introduced in Section 5.3. It comprises of two parts, describing data types and classes, respectively.

B.1 Data Types

Primitive Types

The primitive data types defined in the MOF PrimitiveType package are used. In addition, DataType as defined in the MOF model is used as superclass for all the primitive types and the following data types.

SimTime

This data type is used to describe the simulation time. The format of this type is subject to implementation. There is no default value for the SimTime type.

WclTime

This data type is used to describe the wallclock time. The format of this type is subject to implementation. There is no default value for the WclTime type.

SimSessionStateKind

This data type defines an enumeration for the simulation session state. Its value can be `Init`, `RunRecord`, `RunPlay`, `Suspended`, or `Stopped`, whereas `Init` is the default value.

B.2 Classes

STInterval

This is used to describe a time interval in the simulation time.

SuperClasses
none
Attributes
start (Table B.1)

Specifies the start of the interval.	
type:	SimTime
multiplicity:	exactly one

Table B.1: Database metamodel: STInterval attribute start

end (Table B.2)

Specifies the end of the interval.	
type:	SimTime
multiplicity:	exactly one

Table B.2: Database metamodel: STInterval attribute end

WTInterval

This is used to describe a time interval in the wallclock time.

SuperClasses
none
Attributes
start (Table B.3)

Specifies the start of the lifespan.	
type:	WclTime
multiplicity:	exactly one

Table B.3: Database metamodel: WTInterval attribute start

end (Table B.4)

IndexedEntry abstract

This is the base class for entry types that are used to store the history of values. Each entry is assigned an index, which is incremented starting from zero. The highest index points to the most current value.

Specifies the end of the lifespan.	
type:	WclTime
multiplicity:	exactly one

Table B.4: Database metamodel: WTInterval attribute end

SuperClasses
none
Attributes
index (Table B.5)

Specifies the index of this entry.	
type:	Integer
multiplicity:	exactly one
changeable:	no

Table B.5: Database metamodel: IndexedEntry attribute index

STIndexedEntry abstract

This is the base class for entry types that are used to store the history of values in the simulation time. It inherits from IndexedEntry and has an additional attribute for time-stamp in the simulation time.

SuperClasses
IndexedEntry
Attributes
sTime (Table B.6)

Specifies the simulation time when the associated value is set.	
type:	SimTime
multiplicity:	exactly one
changeable:	no

Table B.6: Database metamodel: STIndexedEntry attribute sTime

BTIndexedEntry abstract

This is the base class for entry types that are used to store the history of values, both in the simulation time and in the wallclock time. It inherits from IndexedEntry and has two additional attributes, one for the time-stamp in the simulation time, and the other for the time-stamp in the wallclock time.

SuperClasses
IndexedEntry
Attributes
sTime (Table B.7)

Specifies the simulation time when the associated value is set.	
type:	SimTime
multiplicity:	exactly one
changeable:	no

Table B.7: Database metamodel: BTIndexedEntry attribute sTime

wTime (Table B.8)

Specifies the wallclock time when the associated value is set.	
type:	WclTime
multiplicity:	exactly one
changeable:	no

Table B.8: Database metamodel: BTIndexedEntry attribute wTime

TimedValue

This class is used for an indexed entry in the simulation time for a value of any defined data type. It inherits from STIndexedEntry.

SuperClasses
STIndexedEntry
Attributes
value (Table B.9)

Specifies the value.	
type:	DataType
multiplicity:	exactly one
changeable:	no

Table B.9: Database metamodel: TimedValue attribute value

SimSessionStateEntry

This class is used to for entries that document simulation session state changes, one entry for each change. It inherits from BTIndexedEntry.

SuperClasses
BTIndexedEntry
Attributes
value (Table B.10)

Specifies the state value.	
type:	SimSessionStateKind
multiplicity:	exactly one
changeable:	no

Table B.10: Database metamodel: SimSessionStateEntry attribute value

SimSessionState

This class is used to describe the entire state change history of a simulation session, using SimSessionStateEntry as contained elements.

SuperClasses
none
Contained Elements
SimSessionStateEntry

SimSessionScaleEntry

This class is used to for entries that document simulation session scale factor changes, one entry for each change. It inherits from BTIndexedEntry.

SuperClasses
BTIndexedEntry
Attributes
value (Table B.11)

Specifies the scale factor value.	
type:	float
multiplicity:	exactly one
changeable:	no

Table B.11: Database metamodel: SimSessionScaleEntry attribute value

SimSessionScale

This class is used to describe the entire scale factor change history of a simulation session, using SimSessionScaleEntry as contained elements.

SuperClasses
none
Contained Elements
SimSessionScaleEntry

GlobalElement abstract

This is the base class for database elements, each of them has a unique identifier.

SuperClasses
none
Attributes
id (Table B.12)

Specifies the identifier of this element.	
type:	String
multiplicity:	exactly one
changeable:	no

Table B.12: Database metamodel: GlobalElement attribute id

STElement abstract

This is the base class for database elements that have a lifespan in the simulation time. It inherits from GlobalElement.

SuperClasses
GlobalElement
Attributes
slife (Table B.13)

Specifies the simulation time lifespan of this element.	
type:	STInterval
multiplicity:	exactly one

Table B.13: Database metamodel: STElement attribute slife

WTElement abstract

This is the base class for database elements that have a lifespan in the wallclock time. It inherits from GlobalElement.

SuperClasses
GlobalElement

Attributes
wlife (Table B.14)

Specifies the wallclock time lifespan of this element.	
type:	WTInterval
multiplicity:	exactly one

Table B.14: Database metamodel: WTElement attribute wlife

BTElement abstract

This is the base class for database elements that have a lifespan in the simulation time, and a lifespan in the wallclock time. It inherits from GlobalElement.

SuperClasses
GlobalElement
Attributes
slife (Table B.15)

Specifies the simulation time lifespan of this element.	
type:	STInterval
multiplicity:	exactly one

Table B.15: Database metamodel: BTElement attribute slife

wlife (Table B.16)

Specifies the wallclock time lifespan of this element.	
type:	WTInterval
multiplicity:	exactly one

Table B.16: Database metamodel: BTElement attribute wlife

DBRoot

This class represents the root of the database. It contains zero or more SimSession. It inherits from WTElement.

SuperClasses
WTElement
Contained Elements
SimSession

SimSession

A SimSession serves here as a container for zero or more SimInstance. It inherits from WTElement.

SuperClasses
WTElement
Contained Elements
SimInstance
Attributes
state (Table B.17)

Specifies the state.	
type:	SimSessionState
multiplicity:	exactly one
changeable:	yes

Table B.17: Database metamodel: SimSession attribute state

scale (Table B.18)

Specifies the scale factor.	
type:	SimSessionScale
multiplicity:	exactly one
changeable:	yes

Table B.18: Database metamodel: SimSession attribute scale

SimInstance

SimInstance serves as container for zero or more Entity. It inherits from BTElement.

SuperClasses
BTElement
Contained Elements
Entity

PortConn

This class is used to describe a port connection established between two Ports.

SuperClasses
IndexedEntry
Attributes
inverse (Table B.19)
slife (Table B.20)

Specifies the inverse of the connection.	
type:	String
multiplicity:	exactly one
changeable:	no

Table B.19: Database metamodel: PortConn attribute inverse

Specifies the simulation time lifespan of the connection.	
type:	STInterval
multiplicity:	exactly one
changeable:	yes

Table B.20: Database metamodel: PortConn Attribute slife

Port

Port is used to specify communication ends of Entity. A Port is contained in an Entity. The connections to other Ports are defined by the contained PortConn elements.

SuperClasses
GlobalElement
Contained Elements
PortConn

ParVarBase abstract

This is the base class for Parameter and Variable, which have different semantics but can be handled in the database technically in the same manner. It inherits from GlobalElement.

SuperClasses
GlobalElement
Attributes
type (Table B.21)

Specifies the type.	
type:	String
multiplicity:	exactly one
changeable:	no

Table B.21: Database metamodel: ParVarBase attribute type

Parameter

A Parameter inherits from ParVarBase. It is contained in an Entity. It is used to configure a simulation model, either in the initialization phase, or at run-time. The history of the value is

described by contained TimedValue.

SuperClasses
ParVarBase
Contained Elements
TimedValue

Variable

Variable inherits from ParVarBase. It is used to specify state variables of a simulation model. A Variable is contained in an Entity. The history of the value is described by contained TimedValue.

SuperClasses
ParVarBase
Contained Elements
TimedValue

Entity

This is the elementary construct according to entities in a simulation model. An Entity may contain zero or more Entity, Parameter, Variable and Port. It inherits from STElement.

SuperClasses
STElement
Contained Elements
Entity, Parameter, Variable, Port
Attributes
type (Table B.22)

Specifies the type of the Entity.	
type:	String
multiplicity:	exactly one
changeable:	no

Table B.22: Database metamodel: Entity attribute type

Appendix C

XML Schema for Simulation Database

This appendix includes the generic schema for the VTIS simulation database, specified using XML Schema. In addition, an example for the extension specific part is also included.

C.1 Generic Part

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace = "http://www.lmu.de/vtis"
xmlns="http://www.lmu.de/vtis"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" version="Dec, 2003">

  <xs:simpleType name="SimTime">
    <xs:restriction base="xs:double"/>
  </xs:simpleType>

  <xs:simpleType name="WclTime">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>

  <xs:simpleType name="SimSessionStateKind">
    <xs:restriction base="xs:token">
      <xs:enumeration value="Init"/>
      <xs:enumeration value="RunPlay"/>
      <xs:enumeration value="RunRecord"/>
      <xs:enumeration value="Suspended"/>
      <xs:enumeration value="Stopped"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="STInterval">
    <xs:sequence>
      <xs:element name="start" type="SimTime"/>
      <xs:element name="end" type="SimTime" nillable="true"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="WTInterval">
```

```
  <xs:sequence>
    <xs:element name="start" type="WclTime"/>
    <xs:element name="end" type="WclTime" nillable="true"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="IndexedEntry" abstract="true">
  <xs:attribute name="index" type="xs:integer" use="required"/>
</xs:complexType>

<xs:complexType name="STIndexedEntry" abstract="true">
  <xs:complexContent>
    <xs:extension base="IndexedEntry">
      <xs:sequence>
        <xs:element name="sTime" type="SimTime"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="BTIndexedEntry" abstract="true">
  <xs:complexContent>
    <xs:extension base="IndexedEntry">
      <xs:sequence>
        <xs:element name="sTime" type="SimTime"/>
        <xs:element name="wTime" type="WclTime"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="ComplexValueItem">
  <xs:sequence>
    <xs:group ref="ValueNodeContent"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="type" type="xs:string" use="required"/>
</xs:complexType>

<xs:complexType name="ComplexValue">
  <xs:sequence>
    <xs:element name="item" type="ComplexValueItem"
      minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:group name="ValueNodeContent">
  <xs:choice>
    <xs:element name="simpleValue"/>
    <xs:element name="complexValue" type="ComplexValue"/>
  </xs:choice>
</xs:group>

<xs:complexType name="TimedValue">
  <xs:complexContent>
```

```
    <xs:extension base="STIndexedEntry">
      <xs:sequence>
        <xs:group ref="ValueNodeContent"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="SimSessionStateEntry">
  <xs:complexContent>
    <xs:extension base="BTIndexedEntry">
      <xs:sequence>
        <xs:element name="value" type="SimSessionStateKind"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="SimSessionState">
  <xs:sequence>
    <xs:element name="entry" type="SimSessionStateEntry"
      minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="SimSessionScaleEntry">
  <xs:complexContent>
    <xs:extension base="BTIndexedEntry">
      <xs:sequence>
        <xs:element name="value" type="xs:float"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="SimSessionScale">
  <xs:sequence>
    <xs:element name="entry" type="SimSessionScaleEntry"
      minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="GlobalElement" abstract="true">
  <xs:attribute name="id" type="xs:string" use="required"/>
</xs:complexType>

<xs:complexType name="STElement" abstract="true">
  <xs:complexContent>
    <xs:extension base="GlobalElement">
      <xs:sequence>
        <xs:element name="slife" type="STInterval"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
```

```
</xs:complexType>

<xs:complexType name="WTElement" abstract="true">
  <xs:complexContent>
    <xs:extension base="GlobalElement">
      <xs:sequence>
        <xs:element name="wlife" type="WTInterval"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="BTElement" abstract="true">
  <xs:complexContent>
    <xs:extension base="GlobalElement">
      <xs:sequence>
        <xs:element name="slife" type="STInterval"/>
        <xs:element name="wlife" type="WTInterval"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="DBRoot">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="WTElement">
        <xs:sequence>
          <xs:element ref="SimSession"
            minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="dataPath" type="xs:string"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<xs:element name="SimSession">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="WTElement">
        <xs:sequence>
          <xs:element name="state" type="SimSessionState"/>
          <xs:element name="scale" type="SimSessionScale"/>
          <xs:element name="SimInstance"
            type="SimInstance" minOccurs="0"
            maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<xs:element name="SimInstance">
```

```
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="BTElement">
          <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:group ref="EntityList"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="PortConn">
    <xs:sequence>
      <xs:extension base="IndexedEntry">
        <xs:sequence>
          <xs:element name="slife" type="STInterval"/>
        </xs:sequence>
      <xs:attribute name="inverse" type="xs:string"/>
    </xs:extension>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Port">
  <xs:complexContent>
    <xs:extension base="GlobalElement">
      <xs:sequence>
        <xs:element name="connection" type="PortConn"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="ParVarBase" abstract="true">
  <xs:complexContent>
    <xs:extension base="GlobalElement">
      <xs:sequence>
        <xs:element name="entry" type="TimedValue"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="type" type="xs:string"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="Parameter">
  <xs:complexContent>
    <xs:extension base="ParVarBase">
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="Variable">
  <xs:complexContent>
```

```
    <xs:extension base="ParVarBase">
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:group name="EntityContent">
  <xs:choice>
    <xs:element name="parameter" type="Parameter" minOccurs="0"/>
    <xs:element name="variable" type="Variable" minOccurs="0"/>
    <xs:element name="port" type="Port" minOccurs="0"/>
    <xs:element name="entity" type="Entity" minOccurs="0"/>
  </xs:choice>
</xs:group>

<xs:complexType name="Entity">
  <xs:complexContent>
    <xs:extension base="STElement">
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:group ref="EntityContent"/>
      </xs:sequence>
      <xs:attribute name="type"
        type="xs:string" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

</xs:schema>
```

C.2 Specific Part

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.lmu.de/vtis"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.lmu.de/vtis"
elementFormDefault="qualified" version="Dec, 2003">

<xs:include schemaLocation="ssdb.xsd"/>

  <xs:group name="EntityList">
    <xs:choice>
      <xs:element name="entity" type="Entity" minOccurs="0"/>
    </xs:choice>
  </xs:group>

</xs:schema>
```

Appendix D

XQuery Prolog for Active Rules

This appendix includes the XQuery prolog for the processing of active rules supporting the VTIS approch.

```
import schema "sessions.xsd";
declare namespace xs = "http://www.w3.org/2001/XMLSchema";
declare namespace vtis = "http://www.lmu.de/vtis";
declare default element namespace "http://www.lmu.de/vtis";

declare function vtis:sessCmd($val as xs:string)
as xs:boolean external;

declare function vtis:wcltimeNow()
as xs:string external;

declare function vtis:simtimeNow()
as xs:string external;

declare function vtis:wcltimeDuration($st as xs:string, $et as xs:string)
as xs:string external;

declare function vtis:simtimeDuration($st as xs:string, $et as xs:string)
as xs:string external;

declare function vtis:wcltimeToDouble($val as xs:string)
as xs:double external;

declare function vtis:wcltimeToString($val as xs:double)
as xs:string external;

declare function vtis:rootNode() as element(vtis:DBRoot){
  doc("sessions.xml")/vtis:DBRoot
};

declare function vtis:allSimSessions()
as element(vtis:SimSession)*{
  vtis:rootNode()//vtis:SimSession
};

declare function vtis:allSimInstances($ss as xs:string)
as element(vtis:SimInstance)*{
  vtis:rootNode()//vtis:SimSession[@id=$ss]/vtis:SimInstance
```

```
};
declare function vtis:wlifeStart($val)
as xs:string{
  $val/vtis:start/text()
};
declare function vtis:wlifeEnd($val)
as xs:string{
  $val/vtis:end/text()
};
declare function vtis:wlifeDuration($val as element(wlife,
vtis:WTInterval))
as xs:string{
  let $st := vtis:wlifeStart($val)
  let $et := vtis:wlifeEnd($val)
  return vtis:wcltimeDuration($st, $et)
};
declare function vtis:wlifeIntersect($val as element(wlife, vtis:WTInterval)*)
as element(wlife, vtis:WTInterval){
  let $stmax := max(
    for $i in $val
    return
      let $t := vtis:wcltimeToDouble(vtis:wlifeStart($i))
      return $t
  )
  let $etmin := min(
    for $i in $val
    return
      let $t := vtis:wcltimeToDouble(vtis:wlifeEnd($i))
      where ($t != -1.0)
      return $t
  )
  let $st := (
    if($stmax <= $etmin)
      then $stmax
    else -1.0
  )
  let $et := (
    if($stmax <= $etmin)
      then $etmin
    else -1.0
  )
  return
    <vtis:wlife>
      <vtis:start>{vtis:wcltimeToString($st)}</svtis:tart>
      <vtis:end>{vtis:wcltimeToString($et)}</vtis:end>
    </vtis:wlife>
};
declare function vtis:inWTInterval($t, $iv)
as xs:boolean{
```

```
  let $ivs := vtis:wcltimeToDouble(vtis:wlifeStart($iv))
  let $ive := vtis:wcltimeToDouble(vtis:wlifeEnd($iv))
  let $td := vtis:wcltimeToDouble($t)
  return
    if($td >= $ivs)
      then
        if($ive != -1.0)
          then
            if($td <= $ive)
              then "true"
            else "false"
        else "true"
    else "false"
};
declare function vtis:slifeStart($val)
as xs:double{
  let $i := $val/vtis:start/text()
  return xs:double($i)
};
declare function vtis:slifeEnd($val)
as xs:double{
  let $i := $val/vtis:end/text()
  return xs:double($i)
};
declare function vtis:slifeDuration($val)
as xs:double{
  let $st := vtis:slifeStart($val)
  let $et := vtis:slifeEnd($val)
  return
    if($et eq -1)
      then xs:double(vtis:simtimeNow()) - $st
    else $et - $st
};
declare function vtis:slifeDuration1($val)
as xs:string{
  let $st := $val/vtis:slife/vtis:start/text()
  let $et := $val/vtis:slife/vtis:end/text()
  return vtis:simtimeDuration($st, $et)
};
declare function vtis:slifeIntersect($val as element(slife, vtis:STInterval)*)
as element(slife, vtis:STInterval){
  let $stmax := max(
    for $i in $val
    return
      let $t := vtis:slifeStart($i)
      return $t
  )
  let $etmin := min(
    for $i in $val
```

```
    return
      let $t := vtis:slifeEnd($i)
      where ($t != -1.0)
      return $t
  )
  let $st := (
    if($stmax <= $etmin)
      then $stmax
    else -1.0
  )
  let $et := (
    if($stmax <= $etmin)
      then $etmin
    else -1.0
  )
  return
    vtis:buildSTInterval($st, $et)
};
declare function vtis:inSTInterval($td, $iv)
as xs:boolean{
  let $ivs := vtis:slifeStart($iv)
  let $ive := vtis:slifeEnd($iv)
  return
    if($td >= $ivs)
      then
        if($ive != -1.0)
          then
            if($td <= $ive)
              then "true"
            else "false"
        else "true"
    else "false"
};
declare function vtis:buildSTInterval($t1, $t2)
as element(slife, vtis:STInterval){
  if($t1 <= $t2)
    then
      <vtis:slife>
        <vtis:start>$t1</vtis:start>
        <vtis:end>$t2</vtis:end>
      </vtis:slife>
  else
    <vtis:slife>
      <vtis:start>$t2</vtis:start>
      <vtis:end>$t1</vtis:end>
    </vtis:slife>
};
```

Bibliography

[AAC+99] S. Abiteboul, B. Amann, C. Cluet, A. Eyal, L. Mignet, and T. Milo. Active views for electronic commerce. In *Proceedings of the 25th. Very Large Data Bases Conference*, pages 138–149, Edinburgh, Scotland, UK, Sep. 1999.

[Adk02] S. S. Adkins. *The 2002 U.S. market for e-learning simulation, executive summary*. brandon-hall.com, Mar. 2002.

[Adv] Advanced Distributed Learning Initiative (ADL). Home page. http://www.adlnet.org.

[Adv01] Advanced Distributed Learning Initiative (ADL). *Sharable Content Object Reference Model (SCORM). The SCORM overview, version 1.2*, Oct. 2001.

[AHW95] A. Aiken, J. M. Hellerstein, and J. Widom. Static analysis techniques for predicting the behavior of active database rules. *ACM Transactions on Database Systems (TODS)*, 20(1):3–41, Mar. 1995.

[Apa] Apache Xerces C++ Parser. Home page. http://xml.apache.org/xerces-c/index.html.

[AQM+96] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data. *Journal of Digital Libraries*, 1(1):68–88, 1996.

[ARI] ARIADNE Foundation. Home page. http://www.ariadne-eu.org.

[Avi] Aviation Industry CBT Committee (AICC). Home page. http://www.aicc.org/index.html.

[BBE+99] S. Bajaj, L. Breslau, D. Estrin, K. Fall, S. Floyd, P. Haldar, M. Handley, A. Helmy, J. Heidemann, P. Huang, S. Kumar, S. McCanne, R. Rejaie, P. Sharma, K. Varadhan, Y. Xu, H. Yu, and D. Zappala. Improving simulation for network research. Technical Report 99-702b, University of Southern California, Computer Science Department, Sep. 1999.

[BC98] P. Barford and M. E. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of Performance '98/ACM SIGMETRICS '98*, pages 151–160, Madison WI, USA, 1998.

[BCD89] J. S. Brown, A. Collins, and P. Duguid. Situated cognition and the culture of learning. *Educational Researcher*, 18(1):32–41, 1989.

[BCS01] A. Bonifati, S. Ceri, and S.Paraboschi. Pushing reactive services to XML repositories using active rules. In *Proceedings of the 10th. International World Wide Web Conference*, Hong Kong, China, May 2001.

[BFGM98] E. Bertino, E Ferrari, G. Guerrini, and I. Merlo. Extending the ODMG object model with time. In *Proceedings of the 12th. European Conference on Object-Oriented Programming (ECOOP'98)*, pages 41–66, Brussels, Belgium, Jul. 1998.

[BINN01] J. Banks, J. S. Carson II, B. L. Nelson, and D. M. Nicol. *Discrete-event system simulation*. Prentice-Hall, Inc., 3rd. edition, 2001.

[BP03] C. Burger and M. Papesch. Eine interaktive und kollaborative Selbstlernumgebung für den Bereich der Sicherheitsprotokolle (An interactive and collaborative self-learning environment for security protocols). In *Proceedings of the 1st. Fachtagung "e-Learning" der Gesellschaft für Informatik (DeLFI 2003)*, München, Germany, Sep. 2003.

[BPS+03] C. Binstock, D. Peterson, M. Smith, M. Wooding, C. Dix, and C. Galtenberg. *The XML Schema complete reference*. Addison-Wesley, 2003.

[BPW02] J. Bailey, A. Poulovassilis, and P. Wood. An event-condition-action language for XML. In *Proceedings of WWW 2002*, Hawaii, USA, May 2002.

[BR01] C. Burger and K. Rothermel. A framework to support teaching in distributed systems. *Journal on Educational Resources in Computing (JERIC)*, 1(1), 2001.

[BS97] J. Begole and C. A. Shaffer. Internet based real-time multiuser simulation: Ppong! Technical Report TR-97-01, Virginia Polytechnic Institute and State University, Department of Computer Science, Feb. 1997.

[BTW+94] J. B. Black, W. Thalheimer, H. Wilder, D. de Soto, and P. Picard. Constructivist design of graphic computer simulations. *National Convention of the Association for Educational Communications and Technology*, 1994.

[CAW99] S. S. Chawathe, S. Abiteboul, and J. Widom. Managing historical semistructured data. *Theory and Practice of Object Systems*, 24(4):1–20, 1999.

[CBN89] A. Collins, J. S. Brown, and S. E. Newman. Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. In L. B. Resnick, editor, *Knowing, Learning, and Instruction. Essays in the Honour of Robert Glaser*, pages 453–494. Erlbaum, Hillsdale, NJ, USA, 1989.

[CDF+03] D. Chamberlin, D. Draper, M. Fernández, M. Kay, J. Robie, M. Rys, J. Siméon, J. Tivy, and P. Wadler. *XQuery from the experts. A guide to the W3C XML query language*. Addison-Wesley, 2003.

[CDK01] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed systems. Concepts and design*. Addison-Wesley, 3rd. edition, 2001.

[Cea00] R. G. G. Cattell and et al. *The object database standard: ODMG 3.0*. Morgan Kaufmann Publ., 2000.

[CGMH+94] S. S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The Tsimmis project: Integration of heterogeneous information sources. In *Proceedings of the 100th. Anniversary Meeting of the Information Processing Society of Japan*, pages 7–18, Tokyo, Japan, Oct. 1994.

[Cis01] Cisco Systems. Internet learning solutions group e-learning glossary, 2001.

[CL99] C. G. Cassandras and S. Lafortune. *Introduction to discrete event systems*. Kluwer Academic Publ., 1999.

[Com95] D. E. Comer. *Internetworking with TCP/IP. Principles, protocols, and architecture*, volume 1. Prentice-Hall, Inc., 1995.

[Cow99] J. H. Cowie. *Scalable Simulation Framework API reference manual, version 1.0*, Mar. 1999.

[Coy02] F. P. Coyle. *XML, web services, and the data revolution*. Addison-Wesley, 2002.

[CZ00] A. B. Chaudhri and R. Zicari. *Succeeding with object databases. A practical look at today's implementations with Java and XML*. Addison-Wesley, 2000.

[Dal54] E. Dale. *Audio-visual methods in teaching*. Holt, Rinehart & Winston, New York, USA, 1954.

[DDLS00] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. Ponder: A language for specifying security and management policies for distributed systems, version 2.3. Technical Report 2000/1, Imperial College of Science, Technology and Medicine, University of London, Department of Computing, Oct. 2000.

[Def96] Defense Modeling and Simulation Organization. *The High Level Architecture rules, version 1.0*. Washinton, DC, USA, 1996.

[DFP+94] S. Das, R. M. Fujimoto, K. Panesar, D. Allison, and M. Hybinette. GTW: A Time Warp system for shared memory multiprocessors. In *Proceedings of the 1994 Winter Simulation Conference*, 1994.

[DHM+93] P. Dillenbourg, M. Hilario, P. Mendelsohn, D. Schneider, and B. Borcic. Intelligent learning environments. Technical Report NFP23 program, project No. 4023-2701, University of Geneva, Switzerland, 1993.

[Dil00] P. Dillenbourg. Virtual learning environments. In *Proceedings of the EUN Conference 2000, Workshop on Virtual Learning Environments*, 2000.

[DIS94] DIS Steering Committee. Institute for simulation and training, Orlando, FL, USA. *The DIS vision – A map to the future of distributed simulation*, 1994.

[Dis99] Distributed Management Task Force, Inc. *Common Information Model (CIM) specification, version 2.2*, Jun. 1999.

[Dit03] H. Ditton. Einführung in die Erziehungswissenschaft (Introduction to educational science). Lecture presentation at Ludwig-Maximilians-University Munich, 2003.

[DW98] D. F. D'Souza and A. C. Wills. *Objects, components, and frameworks with UML. The CTALYSIS approach*. Addison-Wesley, 1998.

[E-L03] E-Learning Guru. E-learning glossary. http://www.e-learningguru.com, 2003.

[EN97] R. Elmasri and S. B. Navathe. *Fundamentals of database systems*. Addison-Wesley, 3rd. edition, 1997.

[FDP+97] R. M. Fujimoto, S. R. Das, K. S. Panesar, M. Hybinette, and C. Carothers. *Georgia Tech Time Warp (GTW), version 3.1. Programmer's manual for distributed network of workstations*. Georgia Institute of Technology, College of Computing, Atlanta, GA, USA, Jul. 1997.

[FGUC97] S. Franks, F. Gomes, B. Unger, and J. Cleary. State saving for interactive optimistic simulation. In *Proceedings of the 11th Workshop on Parallel and Distributed Simulation*, Jun. 1997.

[FJ98] D. Forrester and N. Jantzie. Learning theories. http://www.ucalgary.ca/˜gnjantzi/learning_theories.htm, 1998.

[FM02] F. Fischer and H. Mandl. Lehren und Lernen mit neuen Medien (Teaching and learning with new media). In *Handbuch Bildungsforschung (Handbook of Educational Research)*, pages 623–637. Leske + Budrich, 2002.

[Fra] Fraunhofer IPSI. IPSI-XQ – The XQuery Demonstrator. Home Page. http://www.ipsi.fraunhofer.de/oasys/projects/ipsi-xq.

[FS01] J. Fricke and W. Schiffmann. Telematiklabor für Digitalschaltungen (Tele-Lab for digital circuit). In *Proceedings of GI/ITG Informatik 2001, Band 2*, pages 1149–1153, Wien, Austria, 2001.

[FTM01] F. Fischer, P. Tröndle, and H. Mandl. Using the Internet to improve university education: problem-oriented web-based learning and the MUNICS environment. Technical Report 138, Ludwig-Maximilians-University Munich, Institute for Empirical Pedagogy and Pedagogical Psychology, Jul. 2001.

[Fuj00] R. M. Fujimoto. *Parallel and distributed simulation systems*. John Wiley & Sons, Inc., New York, 2000.

[FV01] K. Fall and K. Varadhan. *The ns manual*. UC Berkeley, LBL, USC/ICI, and Xerox PARC, Sep. 2001.

[Gan99] S. Gançarski. Database versions to represent bitemporal databases. In *Proceedings of the Database and Expert Systems Applications Conference (DEXA'99)*, Florence, Italy, 1999.

[GD93] A. J. Gonzalez and D. D. Dankel. *The engineering of knowledge-based systems. Theory and practice*. Prentice-Hall, Inc., 1993.

[Ger] German Government Programme. New media in education. http://www.gmd.de/PT-NMB.

[Ger97] M. Gertz. *A bibliograhy on integrity constraints*. Institut für Informatik, Universität Hannover, Hannover, Germany, Sep. 1997.

[GS03] D. Gao and R. T. Snodgrass. Syntax, semantics, and query evaluation in the τXQuery temporal XML query language. Technical Report TR-72, TimeCenter, Mar. 2003.

[GY88] S. K. Gadia and C. S. Yeung. A generalized model for a relational temporal database. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 251–259, Chicago, IL, USA, Jun. 1988.

[GZ92] R. Gunzenhäuser and A. Zimmermann. DCE: A knowledge-based tutoring and advisory system. Tutoring strategies and architecture. In I. Tomek, editor, *Proceedings of the 4th. International Conference of Computer Assisted Learning (ICCAL'92)*, pages 247–257, Wolfville, Nova Scotia, Canada, Jun. 1992.

[HAN99] H. G. Hegering, S. Abeck, and B. Neumair. *Integrated management of networked systems. Concepts, architectures, and their operational application*. Morgan Kaufmann, 1999.

[Her01] B. Herzig. Lernföderliche Potenziale von Multimedia: Medienbezogene, lerntheoretische und didaktische Aspekte (Potentials of multimedia for learning). In M. K. W. Schweer, editor, *Aktuelle Aspekte Medien Pädagogischer Forschung (Current aspects of media pedagogical research)*, pages 149–186. Westdeutscher Verlag, Wiesbaden, Germany, 2001.

[HK03] M. Hitz and G. Kappel. *UML at work. Von der Analyse zu Realisierung*. dpunkt.verlag, Heidelberg, 2nd. edition, 2003.

[HL97] M. J. Hannafin and S. Land. The foundations and assumptions of technology-enhanced, student-centered learning environments. *Instructional Science*, 25:167–202, 1997.

[IEE01] IEEE Learning Technology Standards Committee (LTSC). *Standard for learning technology. Learning Technology Systems Architecture (LTSA), P1484.1/D9*, Nov. 2001.

[ILP01] F. Imhoff and C. Linnhoff-Popien. Internet-based teleteaching using the ip-based german broadband science network. In *Proceedings of the International Conference on Intelligent Multimedia and Distance Education*, Fargo, ND, USA, Jun. 2001.

[IMS] IMS Global Laerning Consortium. Home page. http://www.imsglobal.org.

[IMS03a] IMS Global Learning Consortium. *IMS Content Packaging best practice and implementation guide, version 1.1.3*, Jun. 2003.

[IMS03b] IMS Global Learning Consortium. *IMS Content Packaging information model, version 1.1.3*, Jun. 2003.

[IMS03c] IMS Global Learning Consortium. *IMS Content Packaging XML binding, version 1.1.3*, Jun. 2003.

[IMS03d] IMS Global Learning Consortium. *IMS Learning Design best practice and implementation guide, version 1.0*, Jan. 2003.

[IMS03e] IMS Global Learning Consortium. *IMS Learning Design information model, version 1.0*, Jan. 2003.

[IMS03f] IMS Global Learning Consortium. *IMS Learning Design XML binding, version 1.0*, Jan. 2003.

[IMS03g] IMS Global Learning Consortium. *IMS Simple Sequencing information and behavior model, version 1.0*, Mar. 2003.

[Int80] Internet Engineering Task Force. *RFC 768: User Datagram Protocol (UDP)*, Aug. 1980.

[Int81a] Internet Engineering Task Force. *RFC 791: Internet Protocol (IP)*, Sep. 1981.

[Int81b] Internet Engineering Task Force. *RFC 792: Internet Control Message Protocol (ICMP)*, Sep. 1981.

[Int81c] Internet Engineering Task Force. *RFC 793: Transmission Control Protocol (TCP)*, Sep. 1981.

[Int98a] Internet Engineering Task Force. *RFC 2328: OSPF Version 2*, Apr. 1998.

[Int98b] Internet Engineering Task Force. *RFC 2453: RIP version 2*, Nov. 1998.

[ISO96] ISO/IEC. *Extended BNF, 14977, draft*, 1996.

[Jac99] P. Jackson. *Expert systems*. Addison-Wesley, 3rd. edition, 1999.

[JD96] L. Jerinic and V. Devedzic. An object-oriented shell for intelligent tutoring lessons. In A. Díaz de Ilarraza Sánchez and I. Fernández de Castro, editors, *Proceedings of the 3rd. International Conference of Computer Aided Learning and Instruction in Science and Engineering (CALISCE'96)*, pages 69–77, San Sebastian, Spain, Jul. 1996.

[JD00] L. Jerinic and V. Devedzic. The friendly intelligent tutoring environment – teacher's approach. *ACM Special Interest Group on Computer-Human Interaction. SIGCHI Bulletin*, 32(1):83–94, Jan. 2000.

[JeB+98] C. S. Jensen, C. Dyreson (editors), M. Böhlen, J. Clifford, R. Elmasri, S. K. Gadia, F. Grandi, P. Hayes, S. Jajodia, W. Käfer, N. Kline, N. Lorentzos, Y. Mitsopoulos, A. Montanari, D. Nonen, E. Peressi, B. Pernici, J. F. Roddick, N. L. Sarda, M. R. Scalas, A. Segev, R. T. Snodgrass, M. D. Soo, A. Tansel, P. Tiberio, and G. Weiderhold. *The consensus glossary of temporal database concepts. February 1998 version*, 1998.

[Jef85] D. R. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, Jul. 1985.

[Jen] E. D. Jensen. Real-time for the real world. http://www.real-time.org.

[Joh96] W. L. Johnson. Pedagogical agents in virtual learning environments. http://citeseer.ist.psu.edu/johnson95pedagogical.html, 1996.

[Kle99] A. Kleinschmidt. Virtual tutoring in distance education – An approach of centres for distance education for students of the FernUniversität Hagen. In European Association of Distance Teaching Universities, editor, *Proceedings of the ESC'98*, pages 29–32, Heerlen, 1999.

[KLIK99] K. H. Kim, J. Liu, M. Ishida, and I. Kim. Distributed object-oriented real-time simulation of ground transportation networks with the TMO structuring scheme. In *Proceedings of IEEE CS 23rd International Computer Software & Applications Conference (COMPSAC'99)*, Phoenix, AZ, USA, Oct. 1999.

[Kro01] F. W. Kron. *Grundwissen Pädagogik (Basics of pedagogy)*. Reinhardt, München, Germany, 2001.

[KT92] D. Kopec and R. B. Thompson, editors. *Artificial intelligence and intelligent tutoring systems*. Ellis Horwood Limited, Chichester, West Sussex, England, 1992.

[KT98] M. Kamel and C. Trudeau. Springfield: A real-time parallel simulation of mobile telecommunications. In *Proceedings of the Canadian Conference on Electrical and Computer Engineering (CCECE'98)*, Waterloo, ON, Canada, May 1998.

[KT99] J. H. Koch and G. Teege. Problem based learning in computer science. In *Proceedings of the 2nd. International Conference on New Learning Technologies*, Bern, Switzerland, Aug. 1999.

[LH00] S. M. Land and M. J. Hannafin. Student-centered learning environment. In D. Jonassen and S. Land, editors, *Theoretical Foundations of Learning Environments*, pages 1–23. Lawrence Erlbaum, Mahwah, NJ, USA, 2000.

[Liu03] J. Liu. *Improvements in conservative parallel simulation of large-scale models*. PhD thesis, Dartmouth College, Department of Computer Science, Hanover, New Hampshire, USA, Feb. 2003.

[LLC+99] Y.-H. Low, C.-C. Lim, W. Cai, S.-Y. Huang, W.-J. Hsu, S. Jain, and S. J. Turner. Survey of languages and runtime libraries for parallel discrete-event simulation. *Simulation*, 72(3):170–186, 1999.

[LLP02] M. Li and C. Linnhoff-Popien. Interactive simulation in e-learning. In *Proceedings of the IASTED International Conference on Applied Modelling and Simulation*, pages 229–234, Cambridge, MA, USA, Nov. 2002.

[LLPM+03a] M. Li, C. Linnhoff-Popien, S. E. Matalik, T. Seipold, C. Pils, and F. Imhoff. Practic related e-learning – the VIP framework. *Informatica. Special issue: Information and Communication Technology at European Universities. Editors: V. Mahnič, K. Sarlin*, 27(3):263–274, Oct. 2003.

[LLPM+03b] M. Li, C. Linnhoff-Popien, S. E. Matalik, T. Seipold, C. Pils, and F. Imhoff. Practic related e-learning – the VIP framework. In *Proceedings of the 9th. International Conference of European University Information Systems (EUNIS 2003)*, Amsterdam, NL, Jul. 2003.

[LN01] J. Liu and D. M. Nicol. *Dartmouth Scalable Simulation Framework. User's manual, version 3.1*, Aug. 2001.

[LPN+01] J. Liu, L. F. Perrone, D. M. Nicol, M. Liljenstam, C. Elliott, and D. Pearson. Simulation modeling of large-scale ad-hoc sensor networks. In *Proceedings of the European Simulation Interoperability Workshop*, 2001.

[McC98] J. D. McCabe. *Practical computer network analysis and design*. Morgan Kaufmann, 1998.

[MGF00] H. Mandl, C. Gräsel, and F. Fischer. Problem-oriented learning: Facilitating the use of domain-specific and control strategies through modeling by an expert. In W. Perrig and A. Grob, editors, *Control of Human Behaviour, Mental Processes and Awareness*, pages 165–182. Lawrence Erlbaum, Hillsdale, NJ, USA, 2000.

[NEE95] M. A. Nascimento, R. Elmasri, and M. H. Eich. IVTT – A bitemporal indexing structure based on incremental valid time trees. Technical Report 95-CSE-04, Department of Computer Science and Engineering. Sourthern Methodist University, Dallas, TX, USA, Apr. 1995.

[NGF96] R. Nkambou, G. Gauthier, and C. Frasson. CREAM-Tools: An authoring environment for curriculum and course building in an intelligent tutoring system. In A. Díaz de Ilarraza Sánchez and I. Fernández de Castro, editors, *Proceedings of the 3rd. International Conference of Computer Aided Learning and Instruction in Science and Engineering (CALISCE'96)*, pages 186–194, San Sebastian, Spain, Jul. 1996.

[Nis99] N. Nissanke. *Introductory logic and sets for computer scientists*. Addison-Wesley, 1999.

[NJY97] D. M. Nicol, M. M. Johnson, and A. S. Yoshimura. The IDES framework: A case study in development of a parallel discrete-event simulation system. In *Proceedings of the 1997 Winter Simulation Conference*, 1997.

[NN95] H. R. Nielson and F. Nielson. *Semantics with applications – A formal introduction*. John Wiley & Sons, Inc., 1995.

[Nør02a] K. Nørvåg. Algorithms for temporal query operators in XML databases. In *Proceedings of XML-Based Data Management Workshop (XMLDM)*, pages 36–47, Prague, Czech Republic, Mar. 2002.

[Nør02b] K. Nørvåg. Temporal query operators in XML databases. In *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC'2002)*, Madrid, Spain, Mar. 2002.

[Obj00] Object Management Group (OMG). *Distributed Simulation Systems (DSS) specification, version 1.1*, 2000.

[Obj02a] Object Management Group (OMG). *Common Object Request Broker Architecture (CORBA/IIOP), version 3.0.2, formal/02-12-02*, 2002.

[Obj02b] Object Management Group (OMG). *Meta Object Facility (MOF) specification, version 1.4*, Apr. 2002.

[Obj02c] Object Management Group (OMG). *Notification Service Specification, version 1.0.1, formal/02-08-04*, 2002.

[Obj02d] Object Management Group (OMG). *XML Metadata Interchange (XMI) specification, version 1.2*, 2002.

[Obj03a] Object Management Group (OMG). *Unified Modeling Language (UML) specification, version 1.5*, Mar. 2003.

[Obj03b] Object Management Group (OMG). *XML version 1 production of XML Schema specification, version 1.3*, 2003.

[Onl04] Merriam-Webster Online. Merriam-webster dictionary. http://www.m-w.com/, 2004.

[OQT01] B. Oliboni, E. Quintarelli, and L. Tanca. Temporal aspects of semistructured data. In *Proceedings of 8th. International Symposium on Temporal Representation and Reasoning (TIME01)*, Cividale Del Friuli, Italy, Jun. 2001.

[O'S82] T. O'Shea. Intelligent systems in education. In D. Mitche, editor, *Introductory Readings in Expert Systems*, pages 147–176. Gordon and Breach Science Publishers, New York, 1982.

[OS95a] A. Oberweis and V. Säger. A graphical query language for simulation database. *Journal of Microcomputer Applications*, 17(4), 1995.

[OS95b] G. Özsoyoğlu and R. T. Snodgrass. Temporal and real-time databases: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 7(4):513–532, Aug. 1995.

[PD99] N. W. Paton and O. Díaz. Active database systems. *ACM Computing Surveys*, 31(1):63–103, 1999.

[PF01] K. S. Perumalla and R. M. Fujimoto. Interactive parallel simulation with the JANE framework. *Future Generation Computer Systems, Elsevier Science*, 17(5):525–537, 2001.

[Plo81] G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, Aarhus, Denmark, 1981.

[Pro95] TeleSim Project. *SimKit – ATM-TN application programmer's interface, version 1.1*. Jade Simulations International Corporation for WurcNet Inc., Mar. 1995.

[PRS+04] C. Pils, J. Rapp, T. Seipold, A. Vouffo-Feudjio, and M. Li. Interactive real-time simulation within the VIP-Framework. Internal report, RWTH-Aachen, Fraunhofer FOKUS, LMU Munich, 2004.

[Rad03] I. Radisic. *Ein prozessorientierter, policy-basierter Ansatz für ein integriertes, dienstorientiertes Abrechnungsmanagement (A process-oriented, policy-based approach for integrated, service-oriented accounting management)*. PhD thesis, Ludwig-Maximilians-Universität München, Germany, 2003.

[Rep00] A. Repenning. AgentSheets: An interactive simulation environment with end-user programmable agents. In *Proceedings of Interaction 2000*, Tokyo, Japan, 2000.

[SBU00] R. Simmonds, R. Bradford, and B. Unger. Applying parallel discrete event simulation to network emulation. In *Proceedings of the 14th. Workshop on Parallel and Distributed Simulation (PADS 2000)*, Bologna, Italy, May 2000.

[Sch97] R. Schulmeister. *Hypermedia learning systems. Theory – didactics – design*. http://www.izhd.uni-hamburg.de/paginae/Book/default.html, 1997.

[Sch01] M. K. W. Schweer, editor. *Aktuelle Aspekte Medien Pädagogischer Forschung (Current aspects of media pedagogical research)*. Westdeutscher Verlag, 2001.

[SGJM99] E. Shaw, R. Ganeshan, W. L. Johnson, and D. Millar. Building a case for agent-assisted learning as a catalyst for curriculum reform in medical education. In *Proceedings of the International Conference on Artificial Intelligence in Education*, Jul. 1999.

[Sip97] M. Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.

[SJG99] E. Shaw, W. L. Johnson, and R. Ganeshan. Pedagogical agents on the web. In *Proceedings of the 3rd. International Conference on Autonomous Agents*, pages 283–290, May 1999.

[SSC01] Y. Shang, H. Shi, and S. Chen. An intelligent distributed environment for active learning. *Journal on Educational Resources in Computing (JERIC)*, 1(2), 2001.

[Sti00] M. J. Stiles. Towards virtual universities. In *Proceedings of European Universities Information Systems Congress (EUNIS 2000)*, Poznan, Poland, Apr. 2000.

[Sti01] K. Stiller. Möglichkeiten und Grenzen des Medieneinsatzes in Lehr-Lern-Prozessen (Opportunities and limits of the application of media in teaching-learning-processes). In M. K. W. Schweer, editor, *Aktuelle Aspekte Medien Pädagogischer Forschung (Current aspects of media pedagogical research)*, pages 119–148. Westdeutscher Verlag, Wiesbaden, Germany, 2001.

[Tan02a] A. S. Tanenbaum. *Computer networks*. Prentice-Hall, Inc., 4th. edition, 2002.

[Tan02b] A. S. Tanenbaum. *Distributed systems. Principles and paradigms*. Prentice-Hall, Inc., 2002.

[Tap98] D. Tapscott. *Growing up digital: The rise of the net generation*. McGraw Hill, New York, 1998.

[TCG+93] A. U. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass. *Temporal databases. Theory, design, and implementation*. The Benjamin/Cummings Publishing Company, Inc., 1993.

[The] The MITRE Corporation. Collaborative virtual workspace (cvw). http://cvw.sourceforge.net.

[TJS98] K. Torp, C. S. Jensen, and R. T. Snodgrass. Effective timestamping in databases. Technical Report TR-4rev, TimeCenter, Oct. 1998.

[Var02] A. Varga. *OMNeT++ – Discrete event simulation system, version 2.2*. Budapest University of Technology and Economics, Faculty of Electrical Engineering and Informatics, Department of Telecommunications, Mar. 2002.

[Wie02] N. Wiegand. Investigating XQuery for querying across database object types. *SIGMOND Record*, 31(2), 2002.

[Wol01] K. D. Wolf. Internet based learning communites – moving from patchwork environments to ubiquitous learning infrastructures. In S. Dijkstra, D. Jonassen, and D. Sembill, editors, *Multimedia Learning, Results and Perspectives*, pages 189–223. Peter Lang, 2001.

[Woo92] B. P. Woolf. Building knowledge based tutors. In I. Tomek, editor, *Proceedings of the 4th. International Conference of Computer Assisted Learning (ICCAL'92)*, pages 46–60, Wolfville, Nova Scotia, Canada, Jun. 1992.

[Wor00] World Wide Web Consortium (W3C). *Document Object Model (DOM) level 2 events specification, version 1.0*, Nov. 2000.

[Wor01a] World Wide Web Consortium (W3C). *XML schema part 0: Primer*, May 2001.

[Wor01b] World Wide Web Consortium (W3C). *XML schema part 1: Structures*, May 2001.

[Wor01c] World Wide Web Consortium (W3C). *XML schema part 2: Datatypes*, May 2001.

[Wor03a] World Wide Web Consortium (W3C). *SOAP (Simple Object Access Protocol) version 1.2 part 0: Primer*, Jun. 2003.

[Wor03b] World Wide Web Consortium (W3C). *XML path language (XPath) 2.0*, Nov. 2003.

[Wor03c] World Wide Web Consortium (W3C). *XQuery 1.0: An XML query language*, Nov. 2003.

[Wor03d] World Wide Web Consortium (W3C). *XQuery 1.0 and XPath 2.0 data model*, Nov. 2003.

[Wor03e] World Wide Web Consortium (W3C). *XQuery 1.0 and XPath 2.0 functions and operators*, Nov. 2003.

[Wor03f] WorldWideLearn. E-learning glossary. http://www.worldwidelearn.com, 2003.

[Wor04] World Wide Web Consortium (W3C). *Document Object Model (DOM) level 3 core specification, version 1.0*, Feb. 2004.

[WZ03] F. Wang and C. Zaniolo. Temporal queries in XML document archieves and Web warehouses. In *Proceedings of the 10th. International Symposium on Temporal Representation and Reasoning and the 4th. International Conference on Temporal Logic*, Cairns, Queensland, Australia, Jul. 2003.

[ZD02] S. Zhang and C. E. Dyreson. Adding valid time to XPath. In *Proceedings of the 2nd. International Workshop of Databases in Networked Information Systems (DNIS 2002)*, pages 29–42, Aizu, Japan, Dec. 2002. Springer.

[Zha03] W. Zhao. Two-level grammar as the formalism for middleware generation in internet component broker organizations. In *Proceedings of GCSE/SAIG Young Researchers Workshop, held in conjunction with the First ACM SIGPLAN Conference on Generative Programming and Component Engineering*, Pittsburgh, PA, USA, Oct. 2003.

Nomenclature

API Application Programming Interface

CBN-FM Framework by Collins, Brown and Newman

CIDR Classless Inter-Domain Routing

CORBA Common Object Request Broker Architecture

DBMS Database Management System

DIS Distributed Interactive Simulation

DOM Document Object Model

DTD Document Type Definition

ECA Event-Condition-Action (rule)

EER Enhanced Entity-Relationship (model)

ER Entity-Relationship (model)

ETOILE Experimental Toolbox for Interactive Learning Environment

GET-BITS Generic Tools for Building ITS

GVT Global Virtual Time

HiSAP Highly interactive Simulation of Algorithms and Protocols

HLA High Level Architecture

ICMP Internet Control Message Protocol

IDL Interface Definition Language

IP Internet Protocol

ITS Intelligent Tutoring System

KBS Knowledge-Based System

LTS Learning Technology System

LTSA Learning Technology System Architecture

LTSC Learning Technology Standards Committee

MOF Meta Object Facility

ODL Object Definition Language

ODMG Object Data Management Group

OMG Object Management Group

OODB Object-Oriented Database

OQL Object Query Language

OSPF Open Shortest Path First

PDES Parallel Discrete-Event Simulation

RIP Routing Information Protocol

SAP Service Access Point

SEU Simulation Environment Unit

SOAP Simple Object Access Protocol

SQL Structured Query Language

SSF Scalable Simulation Framework

TCP Transmission Control Protocol

UDP User Datagram Protocol

UML Unified Modelling Language

VIP Virtuelles Informatik-Praktikum (project)

VIPNet A network simulation upon VIP simulation environment

VTIS Virtual Teaching Interface for e-learning Simulations

XMI XML Metadata Interchange

XML Extensible Markup Language

Index