Managing Cloud Ecosystems

Brokering, Deployment, and Consumption

Mathias Slawik

Zusammenfassung

Die kontinuierlich wachsende Verbreitung von Cloud-Diensten ließ in den vergangenen Jahren eine Reihe von großen Cloud-Ökosystemen entstehen. Prominentestes Beispiel ist das Cloud-Ökosystem der Amazon Web Services (AWS) mit mittlerweile über 8.000 Partnern und mehr als 18 Mrd. USD Umsatz in 2017. Neben diesen "Giganten" adressieren viele kleinere Ökosysteme speziellere Herausforderungen. Ein Beispiel hierfür ist das TRESOR Cloud-Ökosystem, welches sichere Cloud-Dienste für den Gesundheitssektor bereitstellt. Ein anderes Beispiel ist das CYCLONE Cloud-Ökosystem, das einen gut integrierten "Werkzeugkasten" bietet, um föderierte Multi-Cloud-Lösungen einfach und sicher bereitzustellen und zu verwalten.

Diese Arbeit stellt vier besondere Herausforderungen der Vermittlung, Bereitstellung und Verwendung von Cloud-Diensten in Cloud-Ökosystemen in den Mittelpunkt. Diese Herausforderungen werden unter Einbeziehung aller Beteiligten detailliert analysiert, um Lücken in bestehenden Ansätzen zu identifizieren. Auf Basis dieser Analyse werden mehrere quelloffene Softwarekomponenten und Informationssysteme gestaltet und umgesetzt, um Lösungen unter praxisnahen Bedingungen in den beiden genannten Cloud-Ökosystemen bereitzustellen. Durch das Ziel der allgemeinen Anwendbarkeit der Ergebnisse der Dissertation sollen konkrete Verbesserungen beim Management von Cloud-Ökosystemen erreicht werden. Darüber hinaus fördert die umfangreiche Evaluierung der Entwicklungen zahlreiche neue Erkenntnisse hervor, die zukünftige Forschungs- und Entwicklungstätigkeiten unterstützen.

Im Einzelnen sind die vier adressierten Herausforderungen dieser Arbeit:

1. Die Beschreibung und Vermittlung von Cloud-Diensten aus *nutzerzentrischer* Perspektive.

   Hierfür werden auf Basis von sechs Praxis-Anwendungsfällen eine Reihe von Informationssystemen erstellt und evaluiert, die potentielle Anwender beim Finden, Vergleichen und Auswählen von Cloud-Diensten unterstützen. Basis aller Anwendungssysteme ist eine neu-entwickelte textuelle domänenspezifische Sprache, welche zahlreiche Verbesserungen gegenüber alternativen Ansätzen mit sich bringt. Unter Einbeziehung von potentiellen Nutzern werden die entstehenden Lösungen qualitativ und experimentell evaluiert und deren hohe Praxisrelevanz demonstriert.

2. Die Ende-zu-Ende Absicherung von HTTP Nachrichtenkörpern über Cloud Intermediäre hinweg.

   Solche Intermediäre, beispielsweise Proxies, Lastverteiler und Firewalls, finden sich häufig in modernen Cloud Ökosystemen. Sie stellen aus Sicht der Informationssicherheit besondere Herausforderungen dar, da sie oftmals TLS-Verbindungen terminieren und daher Zugriff auf den unverschlüsselten Inhalt der Kommunikation erlangen. Um diese Herausforderungen zu adressieren, wurde das Trusted Cloud Transfer Protocol (TCTP) entwickelt, welches eine transparente Verschlüsselung von HTTP Nachrichtenkörpern "Ende-zu-Ende", d.h., zwischen Browser und Serverprozess ermöglicht. Die Evaluierung des Protokolls in praxisnahen Szenarien zeigt eine hohe Leistungsfähigkeit und einfache Implementierbarkeit.

3. Die Einhaltung von Sicherheit und Konformität der Cloud-Nutzung.

   Gerade in sensiblen Bereichen, wie dem Gesundheitswesen, muss die Nutzung von Cloud-Diensten besonderen Anforderungen bezüglich Sicherheit und Konformität zu rechtlichen und organisatorischen Regularien gerecht werden. Um diese Anforderungen jederzeit umzusetzen und durchgehend zu kontrollieren, wurde ein verteilter Cloud-Proxy implementiert, der als vertrauenswürdiger Mediator in jedwede Cloud-Nutzung eingebunden ist und eine Vielzahl von funktionalen Modulen zur Verfügung stellt. Hierin integriert der Proxy auch die Entwicklungen der beiden vorangegangenen Schwerpunkte: TCTP zur Verankerung der Nachrichtensicherheit sowie einen Cloud-Broker zur Anbindung der ausgewählten und gebuchten Cloud-Dienste. Weiterhin werden mithilfe von XACML Unternehmensrichtlinien abgebildet und während der Dienstnutzung durchgesetzt. Die Implementierung des Proxies wurde im TRESOR Ökosystem erfolgreich im praxisnahen Einsatz erprobt und evaluiert.

4. Die sichere Bereitstellung und Verwaltung von Multi-Cloud-Lösungen in föderierten Umgebungen.

   Aus der Komplexität von Cloud-Infrastrukturen folgt meist auch ein großer Mehraufwand für ihre sichere Bereitstellung und Verwaltung. Dies wird besonders in föderierten Umgebungen deutlich, in denen mehrere Cloud-Systeme zum Einsatz kommen. Um diese Aufwände möglichst gering zu halten, wurde eine Multi-Cloud-Sicherheitsarchitektur entwickelt und im CYCLONE-Projekt produktiv umgesetzt. Die Architektur beinhaltet eine Reihe von integrierten Komponenten, welche insbesondere die Bereitstellung und Verwaltung von Multi-Cloud-Applikationen sowie die Verwendung föderierter Identitäten in Web-, aber auch Konsolenanwendungen erheblich vereinfachen. Um die betriebswirtschaftlichen Vorteile der Architektur zu ergründen, wird diese im Anschluss aus ökonomischer Sicht analysiert. Zuletzt wird eine zentrale Komponente der Architektur, der CYCLONE Federation Provider, einer tiefgehenden Sicherheitsanalyse unterzogen.

Abstract

In recent years, the ever-growing proliferation of cloud services led to the creation of large cloud ecosystems. The most prominent example is the cloud ecosystem of Amazon Web Services (AWS), which now has over 8,000 partners and more than $ 18 billion in revenue in 2017. In addition to these "giants", many smaller ecosystems address more specific challenges. An example is the TRESOR cloud ecosystem, which provides secure cloud services for the German healthcare sector. Another example is the CYCLONE cloud ecosystem, which provides a well-integrated "toolbox" to easily and securely deploy and manage federated multi-cloud applications.

This thesis focuses on four unique challenges of brokering, deploying, and consuming cloud ecosystem services. These challenges are analyzed in detail with the involvement of all stakeholders to identify gaps in existing approaches. Based on this analysis, several open-source software components and information systems are designed and implemented to provide solutions under real-world conditions in the two aforementioned cloud ecosystems. The dissertation's goal of general applicability should lead to concrete improvements in the management of cloud ecosystems. In addition, the extensive evaluation activities reveal many new findings that will support future research and development activities.

The four addressed challenge areas of this thesis are:

1. The description and brokering of cloud services from a *user-centric* perspective.

   On the basis of six practical use cases, a series of information systems are created and evaluated that support potential users in discovering, assessing and selecting cloud services. The basis of all these systems is a newly developed textual domain-specific language that features numerous improvements over alternative approaches. Involving potential users, the qualitative and experimental evaluation of the resulting solutions demonstrates their outstanding practical relevance.

2. End-to-end protection of HTTP message bodies across cloud intermediaries.

   Intermediaries, such as proxies, load balancers, and firewalls, are commonly found in modern cloud ecosystems. From an information security

point of view, they present particular challenges, as they often act as TLS server connection ends and thus gain access to the unencrypted communication content. To address these challenges, the Trusted Cloud Transfer Protocol (TCTP) was developed, which enables transparent encryption of HTTP message bodies "end-to-end" between the browser and the origin server. The evaluation of the protocol indicates high performance and easy implementability.

3. Secure and compliant cloud service consumption

   Especially in sensitive areas, such as the health care sector, the use of cloud services has to meet special requirements regarding security and compliance with legal and organizational regulations. In order to implement these requirements at all times and to control them continuously, a distributed cloud proxy was implemented that acts as a trusted mediator in any cloud usage scenario. It provides a variety of functional modules that also integrate developments of the two previous challenge areas, i.e., TCTP to ensure end-to-end message security and a cloud broker to integrate selection and booking of cloud services with their consumption through the proxy. Furthermore, corporate regulations are mapped onto XACML policies for their continuous enforcement during service usage. The implementation of the proxy was successfully deployed and evaluated in practice within the TRESOR ecosystem.

4. Secure deployment and management of federated multi-cloud applications.

   Rising cloud infrastructure complexity usually leads to increased efforts for secure application deployment and management. This is particularly evident in federated multi-cloud environments. To minimize those efforts, a multi-cloud security architecture was developed and applied in practice within CYCLONE. It includes a number of integrated components that greatly simplify the deployment and management of multi-cloud applications and the utilization of federated identities by web and console applications. In order to emphasize the reduced management efforts, the architecture is additionally analyzed from an economic perspective. Finally, a key component of the architecture, the CYCLONE Federation Provider, is subjected to a thorough security modelling and threat analysis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Within the last decade, the Cloud Computing paradigm has become ubiquitous and fundamentally transformed how computing, storage, and networking resources are provided. In fact, IDC predicts that "by the end of 2018, over half of enterprise-class businesses will subscribe to more than five different public cloud services."[102] The transformation by the Cloud Computing paradigm does not only incorporate novel business models of huge public cloud offerings of Amazon, Microsoft, and Google. The refinement of the techniques and concepts underpinning those offerings produced many novel concepts to implement modern applications. For example, immutable application deployments, containerized microservices, and ubiquitous JavaScript-based frameworks flourished through the emerging of the Cloud. Modern application development methodologies amplify these stunning changes, for example, the "The Twelve-Factor App"[1] methodology. Recent publications, for example, [25, 40, 39], suggest that the Cloud is already evolving towards the Intercloud, a "cloud of clouds", which will break down the barriers between cloud providers and consumers.

To harness the Cloud Computing paradigm, "cloud ecosystems" unite cloud services, providers, consumers, and possibly other stakeholders, for example, solution partners, auditors, and escrow holders. As an example, the Amazon cloud ecosystem provides more than 70 cloud services to serve over 1 mio enterprise customers, attracts over 8,000 partner network members, and generates a projected $18bn revenue in 2017.[2]

Besides the well-known ubiquitous cloud ecosystems of Amazon, Google, and Microsoft, there are also non-generic ecosystems which target specific areas. Two examples of such variety are the TRESOR and CYCLONE ecosystems wherein many of the contributions of this thesis have been deployed. Both utilize Cloud Computing innovations to tackle unresolved challenges that are of high relevance for their specific area. TRESOR provides a secure cloud computing ecosystem for the German healthcare sector while CYCLONE estab-

[1]https://12factor.net/

[2]https://expandedramblings.com/index.php/amazon-web-services-statistics-facts/

lishes a middleware stack to ease the management of multi-cloud and federated cloud applications. Specifically, both ecosystems address Cloud Computing challenges faced by small and medium-sized enterprises (SMEs). There are fundamental reasons for having SMEs as the application domain. First, SMEs play an important economic role. In Europe, for example, 99.8% of all enterprises are SMEs and 66.5% of the EU workforce is employed by an SME [177]. Besides this, TRESOR targets the German health sector, which is dominated by SMEs. Second, SMEs receive significant benefits from Cloud services as investigated by Lacity et al. in [92]. This provides a compelling reason for SMEs to participate in SaaS ecosystems and use federated multi-cloud applications as targeted by TRESOR and CYCLONE.

The guiding idea of this thesis is to address four main unresolved cloud challenges that are significant for the management and operation of cloud ecosystems in general, and that are also of particular importance for the stakeholders of the TRESOR and CYCLONE ecosystems. In comparison to paramount ecosystems such as the Amazon cloud services, TRESOR and CYCLONE demonstrate an exceptional focus on clearly defined cloud challenges whose practical relevance is ensured by direct stakeholder involvement instead of generic marketing activities. This thesis incorporates these stakeholders' requirements when designing and implementing its contributions and also uses these ecosystems as an evaluation environment. The results of this thesis therefore provide both advancements for cloud ecosystems in general as well as specific benefits for the TRESOR and CYCLONE stakeholders.

To introduce this thesis, the remainder of this chapter first presents the target ecosystems TRESOR and CYCLONE in Section 1.1 before the main research questions are stated in Section 1.2. Afterwards, the applied research method and the resulting thesis structure are explained in Section 1.3. The introduction concludes with a summary of the research contributions in Section 1.4.

1.1 Addressing Contemporary Cloud Challenges: The TRESOR and CYCLONE Ecosystems

The TRESOR project was conducted from March 2012 till December 2015 to enable the use of Cloud Computing in the health sector while complying to strict constraints such as enterprise policies and legal regulations. It created an infrastructure that allows protection of sensitive data, for example, patient records, by strict rules as well as integrating previously isolated applications using a trusted cloud platform. Customer requirements and service capabilities are mediated through a cloud broker on a trusted marketplace. The two main tasks of the project were (1.) to build a scalable and standardized exchange of IT resources and services between different actors in a cloud ecosystem [96, 26] as well as (2.) providing an adequate level of legal protection and IT security.

The CYCLONE European Innovation Action focuses on three main areas that present challenges in multi-cloud settings: (1) application deployment and management, (2) authentication and authorization using federated identities, and (3) software-defined network management. The project outcome is a software stack that comprises preexisting production-ready tools as well as additional extensions tackling multi-cloud issues. It contains solely open-source software in order to maximize its utility and also provides a basis for further collaboration. CYCLONE helps application developers and operators to solve their multi-cloud hardships by providing readily usable software and accompanying supporting documentation. In fact, all of the developments are hosted in the project GitHub Repository[3]. CYCLONE also provides comprehensive documentation on its website[4]. Another project task is to apply the stack to enhance diverse use cases in academic and commercial production- and near-production settings to ensure the eventual applicability of the project results and highlight CYCLONE's utility in existing DevOps environments.

To better understand the contributions of this thesis to these ecosystems and the general body of knowledge, the following subsections introduce both ecosystems briefly. For reference, more information about the concepts and components of TRESOR and CYCLONE can be found in the appendix.

1.1.1 TRESOR - Providing a Secure Cloud Ecosystem for the German Healthcare Sector[5]

The German health sector is one of the largest sectors of employment that provides jobs for around 4 mio people [108]. As technical demands on medical services rise, providing IT support through systems and services becomes far more important. However, available resources for investment are short in supply [31]. The area is defined by a number of small- and medium-sized regional enterprises, such as practices and hospitals. These enterprises often host their own IT infrastructure, which leads to incompatibilities and interoperability challenges. Maintaining these infrastructures is oftentimes not only laborious and costly but also constrained by substantial legal requirements regarding data privacy, data safety, compliance, interoperability, scalability, and availability [31].

All of these considerations provide many reasons for the health sector to look for alternative methods to implement IT services. One way of doing so is moving the infrastructure to the Cloud and use computing, storage, and applications "on demand" over the Internet. As Cloud Computing caused structural changes in the IT market over the last decade, the health sector expects its application to deliver the same benefits seen in other areas: cost reduction, better scalability, higher performance and availability, globally available services with low latency, and flexible billing of IT resources based on concrete usage [113, 80]. At the end, it is expected to deliver an increase in quality of service for stationary and mobile healthcare personnel [31].

However, there are fundamental challenges when applying Cloud Computing: Usually, there is the risk of subjecting cloud consumers to their cloud providers, e.g., through the use of proprietary APIs and formats, the so called "lock-in effect" [34]. Moreover, cloud applications sometimes become isolated and therefore cannot provide much needed interoperability between the different health service platforms [123]. Both cloud providers and customers are looking for ways how to move and operate cloud applications in accordance with applicable law. Besides this, lack of trust in existing concepts for data protection and safety are an obstacle for faster market development in this area

[3]http://github.com/cyclone-project

[4]www.cyclone-project.eu

[5]**TR**usted **E**cosystem for **S**tandardized and **O**pen cloud-based **R**esources

[26, 90]. At last, there are further unresolved challenges, especially for Cloud Computing in SMEs and other sectors [44].

Cloud computing promises many advantages, foremost, it is recognized by the TRESOR use case partners as a viable way to reduce operational costs. These cost reductions often are associated with a lack of features, industry standards, interoperable solutions, compliance to legal requirements, as well as security and privacy functionality. When those shortcomings impact providers and users, it leads to untrustworthy business relationships [191].

TRESOR has identified disadvantages and risks within Cloud Computing that are the main reasons for the slow cloud adoption within the sensitive domain of the German health sector:

- ***Privacy, Legal, and Compliance Issues***

  Sensitive domains require a number of assurances regarding, for example, data privacy, legal compliance, and secure auditing. Some of them are reflected within acts, such as Payment Card Industry - Data Security Standards (PCI DSS), Sarbanes-Oxley (SOX), and the Health Insurance Portability and Accountability Act (HIPAA). Using prominent Cloud Computing platforms could be classified as "operations outsourcing over organizational and country borders". Therefore, as [147, 67, 35] point out, special care has to be taken that the outsourcing provider fulfills these requirements. Furthermore, hardware virtualization, storage abstraction, multi-tenancy, and container technologies allow flexible utility computing models, yet contribute to these issues themselves [29, 9, 147]. At last, the jurisdictions of most laws and provisions are limited to certain national borders and regions. As globally distributed Cloud Computing environments cause these borders to become indistinct, the risk for enterprises not being compliant to these requirements is increasing.

- ***High Integration Efforts and Lock-in Effects***

  As with most other IT services, migrating to and using Cloud Computing services introduces follow-up costs, as shown in [85] and [66]. Some of these costs are hidden, for example, costs for making services compliant to regulations, backup, restore, and disaster recovery procedures. Furthermore, there is a manifold of possible scenarios and associated requirements to integrate cloud services into existing enterprise architectures. When cloud services are not adaptable, varied, and extensible enough, integration becomes a hardship. Lock-in effects, on the other hand, arise from the lack of industry standards enforcement and make migration to other providers difficult. The bankruptcy of a cloud service provider is still a partially unresolved issue, see for example the case study found in [43].

- ***Lack of Transparency***

  Many cloud consumers from sensitive sectors need to diligently asses a wealth of aspects about specific cloud services. In order to do so, they need to have as much information as possible about the cloud services and their providers, for example, contingency procedures, such as their backup, restore, and disaster recovery. However, these are almost never made transparent to the cloud consumer, making it challenging and troublesome to assess and select cloud services.

1.1.2 CYCLONE - Ecosystem Middleware for Multi-cloud and Federated Cloud Applications

DevOps teams have to consider a lot when working in the cloud: deployments need to work everywhere, identities need to come from anywhere, and networks need to connect to anyone. Understandably, a life in DevOps is complex when everything needs to work anytime on any cloud and stakeholders expect continuous high productivity. Just as the weather system known as "cyclone" bridges many clouds, the holistic middleware stack of the CYCLONE ecosystem prevents multicloud-induced headaches. It supplies a comprehensive management stack, for example, a deployment manager, a practical identity federation, as well as a network manager that connects VMs independent of any specific infrastructure. This section explains the ecosystem as well as the stakeholders and main project requirements. The appendix and the project deliverables [173] provide further details.

There are countless ways to deploy and manage applications running on any cloud on a single VM. Most providers and tool vendors put special emphasis in their marketing material how easy this is on their platform using their tools. Nowadays simple `docker pull`, `aws ec2 run-instances`, and `kubectl run` commands achieve more impressive results much faster than years before. However, the highest barriers that need to be surmounted prevail when deploying and managing a large number of heterogeneous application components on multiple VMs on different clouds. This multi-cloud challenge presents unmet requirements that are not yet addressed satisfactory for cloud practitioners.

Various cloud layers have to work in concert in order to manage and deploy complex multi-cloud applications, executing sophisticated workflows for cloud resource deployment, activation, adjustment, interaction, and monitoring. While there are ample solutions for managing individual cloud aspects (e.g. network controllers, deployment tools, and application security software), there are no well-integrated suites for managing the entire cloud stack, especially within multi-cloud and multi-provider scenarios. Therefore, the establishment of ecosystems without having such integrated suites proves a very challenging endeavour.

At last, the "Intercloud" is trending: a globally integrated Cloud of Clouds sharing APIs, protocols, and data formats. A proposal to use existing standards and common mechanisms to achieve the "Intercloud Root" was made by Bernstein, et al. in [25]. Demchenko[6], et al. defined the Intercloud Architecture Framework in [40], addressing Intercloud issues, as well as defining models and architecture patterns for federated access control within Intercloud environments in [39]. A future Intercloud could incorporate tightly integrated cloud platforms, such as OpenStack[7] and the VMWare Software-Defined Data Center[8]. While these tools provide integrated Cloud management, they fail to deliver open and standardized APIs, protocols, and data formats, and

[6] Also involved in CYCLONE
[7] https://www.openstack.org
[8] http://www.vmware.com/software-defined-datacenter/

their components are difficult to replace. Furthermore, Intercloud scenarios require cross-cloud (e.g., public-private) interoperability, compatibility, and interchangeability which is currently challenging to implement. Thus, Application Service Providers (ASPs) as well as their customers are constricted in their deployment of well-integrated Cloud solutions, and the Intercloud awaits its implementation in practice.

Within the CYCLONE ecosystem *cloud application service providers* use CYCLONE components to offer diverse functionality to *cloud developers* and *cloud operators* who implement, deploy, and manage cloud applications for *cloud application end-users*. For CYCLONE to address the challenges faced in federated environments and multi-cloud applications, it needs to provide components within the following areas:

- **Federated identity.** Multi-cloud security can be best achieved when there is a common authentication system that is used by the cloud application end-users to log in using their federated identities.

- **End-to-end encryption.** Common HTTP intermediaries found in cloud ecosystems, such as reverse proxies and load balancers, often act as TLS server connection ends, accessing HTTP/TLS plaintext. To secure this communication, there has to be an end-to-end encryption of sensitive HTTP entity bodies, i.e., an encryption of data between user agent and origin server.

- **Distributed logging.** Managing multi-cloud applications effectively requires consolidating all event logs of the involved software, that is, a distributed logging system.

- **Deployment description.** There has to be a method of describing cloud application deployments, at best supporting scripting, multi-cloud deployment, orchestration, as well as custom application lifecycle hooks.

- **Management APIs.** Any IaaS solution should provide an easy to use, comprehensible, and rich set of management APIs for computing, storage, and network.

- **VM marketplace.** To allow collaboration between end-users, VM appliance creators, and DevOps engineers, the IaaS solution should incorporate a comprehensive VM marketplace.

- **Network service management platform.** As cloud applications have limited control and visibility of network resources, it is challenging to achieve service delivery automation, resource management, and on-demand network connectivity. To implement advanced network services, there has to be a network service management platform, integrated into the employed IaaS offering.

- **Brokering.** To support cloud application developers in finding suitable services in the vast Intercloud, there has to be a service formalization, a service vocabulary, and a brokering component. It should adapt to dynamically changing Intercloud services' properties.

1.2 Research Questions - Cloud Challenges

This thesis provides answers to the following four main research questions which represent cloud challenges that have neither been answered satisfactory for cloud ecosystems in general nor for the specific demands of the target ecosystems.

RQ1: How can cloud services be discovered, assessed, and selected using user-centric cloud service registries?

Many potential cloud consumers are overburdened by the persistent challenges of discovering, assessing, and selecting contemporary Cloud Service offerings: the cloud market is vast and fast-moving, the selection criteria are ambiguous, service knowledge is scattered through the Internet, and features and prices are complex and incomparable.[155, 160] As Section 2.1.3 explains in detail, much research has been carried out to create cloud service registries that help users select cloud services for eventual consumption, especially within the field of Semantic Web services. However, the question remains how contemporary cloud ecosystems can feature a cloud service registry that is *user-centric*, i.e., that caters to the needs of the specific businesses and private users within such ecosystems, instead of rather generic concerns as it is often observed in the related work. Reducing efforts for cloud service brokering is especially relevant for SMEs that operate under tight resource constraints. At last, having such user-centric registries allows advanced matchmaking scenarios as proposed by Zilci et al. in [195, 196].

RQ2: How can HTTP entity-bodies be secured end-to-end through HTTP intermediaries?

Contemporary cloud ecosystems incorporate HTTP intermediaries, such as reverse proxies, load balancers, intrusion prevention systems, and web application firewalls (WAF). These act as TLS server connection ends and access HTTP/TLS plaintext to carry out their functions. This raises many concerns: increased security efforts, the risk of losing confidentiality and integrity, and potentially unauthorized data access.[152]

The analysis in Section 2.2 shows how current HTTP entity-body encryption technologies address these concerns by providing end-to-end security between user agents and origin servers through HTTP intermediaries. However, they are associated with disparate deficiencies, for example, inefficient presentation languages, message-flow vulnerabilities, and the circumvention of HTTP streaming. To provide well-secured cloud ecosystems, there needs to be a novel protocol that can overcome these shortcomings.

RQ3: How can cloud proxies ensure secure and compliant cloud service consumption?

The biggest challenge when consuming cloud services in sensitive areas, such as the healthcare sector, is constantly ensuring compliance to the large set of governing legal regulations and enterprise policies. This challenge can be addressed by establishing trusted ecosystems that provide compliance guarantees to all participants through a consistent implementation of cross-cutting functions, for example, authentication, authorization, accounting, monitoring, and auditing. As stated by Thatmann in [172, 171], this consistency can be achieved through a trusted mediator that is always involved when end-users consume cloud services. The RESTful architecture[54], which is typical for SaaS solutions, suggests implementing such mediator as a versatile HTTP cloud proxy. However, this thesis needs to work out in detail how such a proxy can effectively manage cloud service consumption according to the constraints of sensitive sectors.

RQ4: How can applications be deployed and managed securely in federated, multi-cloud environments?

There is widespread usage of cloud technologies within contemporary application deployments, for example, using VMs and containers for componentization of applications, relying on highly scalable public cloud infrastructures, and embracing the immutable infrastructure paradigm to structure scalable cloud services, possibly deployed to multiple clouds.[158] As the architecture of these applications becomes increasingly distributed and complex, the corresponding security architectures need to be refined and extended further, especially within federated environments spanning multiple clouds.[156] However, there is a lack of security architectures that consist of well-integrated components that make federated identities easily accessible and that allow DevOps engineers to deploy applications transparently to multiple clouds.

1.3 Research Method and Thesis Structure

The research presented in this dissertation is grounded in a comprehensive research framework, "Design Science in Information Systems Research"[75] by Hevner et al., visualized in Figure 1.1. It provides a definition of IS research as the development of new theories and artifacts, rigorously applying previous knowledge to address business needs within an appropriate application environment. These theories and artifacts are iteratively justified and evaluated using methods such as case studies, experiments, and simulations.

The thesis structure shown in Figure 1.2 represents the application of the research method proposed by Hevner et al. onto the four research questions, i.e., the targeted cloud challenges. Each challenge is consistently approached using three main steps that are explicated in the following:

1. **Requirements & Related work**

   As the first research activity, Section 2 provides a detailed analysis of each respective cloud challenge. Based on the observations in the contemporary cloud ecosystems presented in Section 1.1, this analysis includes a clear definition of the stakeholder requirements as well as an introduction of other related works. Contrasting both discloses the relevant gaps in the knowledge base that the subsequently created artifacts should respond to.

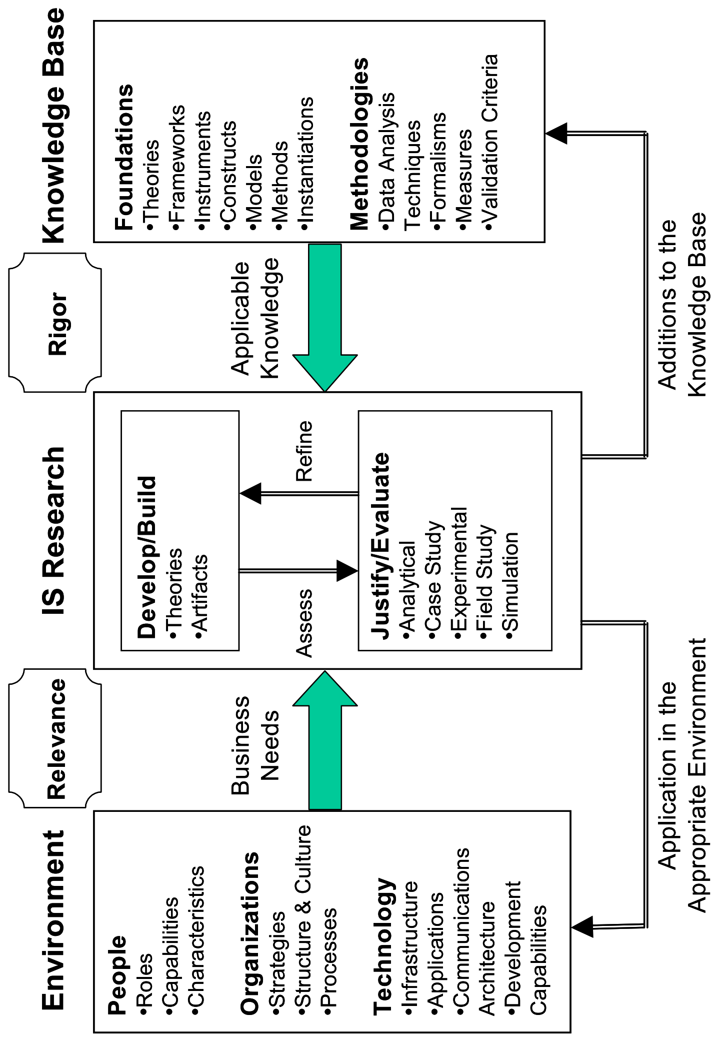


Figure 1.1: Research Framework

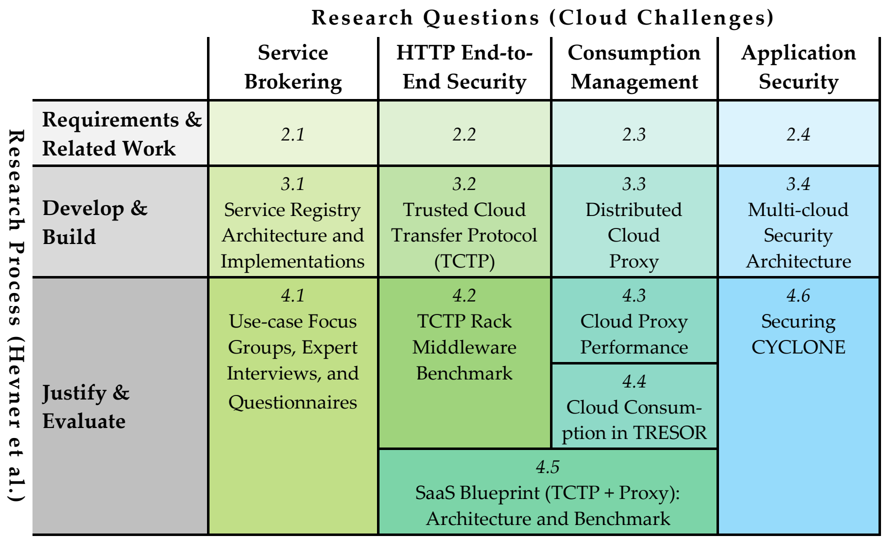

Research Questions (Cloud Challenges)
Service Brokering
HTTP End-to-End Security
Consumption Management
Application Security
Research Process (Hevner et al.)
Requirements & Related Work
2.1
2.2
2.3
2.4
Develop & Build
3.1 Service Registry Architecture and Implementations
3.2 Trusted Cloud Transfer Protocol (TCTP)
3.3 Distributed Cloud Proxy
3.4 Multi-cloud Security Architecture
Justify & Evaluate
4.1 Use-case Focus Groups, Expert Interviews, and Questionnaires
4.2 TCTP Rack Middleware Benchmark
4.3 Cloud Proxy Performance
4.4 Cloud Consumption in TRESOR
4.5 SaaS Blueprint (TCTP + Proxy): Architecture and Benchmark
4.6 Securing CYCLONE

Figure 1.2: Thesis structure

2. **Develop & Build**

   To provide adequate solutions for the observed challenges, this dissertation establishes a number of artifacts that are subsequently deployed to real-world ecosystems. Section 3 provides an in-depth explanation of the design and the implementation of the four main artifacts:

   - A user-centric cloud service registry architecture and its implementations (Section 3.1)
   - The Trusted Cloud Transfer Protocol (TCTP) that enables end-to-end entity-body security through HTTP intermediaries (Section 3.2)
   - The distributed cloud proxy which ensures secure and compliant cloud service consumption (Section 3.3)
   - A security architecture for federated, multi-cloud applications (Section 3.4)

3. **Justify & Evaluate**

   Section 4 presents a detailed use case-driven evaluation of the created artifacts using appropriate evaluation activities. It explains how they have been deployed within the target ecosystems to determine their suitability to address the challenges in practice. Most of the evaluation was done with direct stakeholder involvement, e.g., through project workshops, instead of purely theoretical or unrelated considerations. Some evaluation activities targeted additional professionals that were not directly involved with the projects, further strengthening the relevance of this dissertation.

1.4 Contributions of this Thesis

This thesis provides four main contributions to the state-of-the-art that result from addressing each of the research questions:

User-centric Cloud Service Registries for Real-World Use Cases

The analysis of real-world requirements of six use cases in Section 2.1.4 highlights the lack of *user-centric* registries in the related work, where most approaches are rather generic and unspecific. This dissertation offers a service registry architecture (see Section 3.1) that provides a textual domain specific language (SDL-NG) to let any user describe services easily (see Section 3.1.1) as well as business vocabularies that reflect common service selection criteria (see Section 3.1.2).

This architecture is subsequently employed in novel service brokering and matchmaking scenarios that support users in their selection process. For example, one implementation of the architecture is the Open Service Compendium (OSC), a crowd-sourced cloud service registry. The evaluation activities that are explained in Section 4.1 attest how all of this solves real-world challenges in diverse near-production settings well.

The implication is that a substantial benefit for service registry users can be created by following a simple architecture that is focused on their concrete needs instead of aiming for highest sophistication and broadest applicability as observed in many of the related works.

TCTP, A Secure Communication Protocol for Common Cloud Ecosystems

In Section 3.2, this dissertation introduces the Trusted Cloud Transfer Protocol (TCTP), which is a novel approach to entity-body encryption that overcomes contemporary deficiencies when cloud service consumption is carried out through HTTP intermediaries. The pivotal idea of TCTP, which is explained in Section 2.2.4, are HTTP application layer encryption channels (HALECs) that offer TLS functionality on the HTTP application layer. Such HTTP application layer TLS has never been observed in any related work.

The evaluation in Section 4.2 additionally reveals beneficial performance characteristics of the principal TCTP implementation while Section 4.5 demonstrates the suitability of TCTP to secure SaaS services in medical cloud ecosystems.

A Distributed Cloud Proxy for Secure and Compliant Cloud Service Consumption

This thesis contributes a distributed cloud proxy that provides cross-cutting security functionality to all ecosystem constituents, for example, federated authentication, XACML-based authorization, as well as facilities for monitoring and auditing. The proxy makes use of other thesis developments, for example, it relies on TCTP to prevent access of confidential data. When integrated with a service registry implementation, such as the TRESOR Broker, it also manages the runtime consumption of services that were brokered and booked in the service registry. The evaluation of the proxy deployment in Section 4.4 demonstrates the technical feasibility of the concept as well as the usefulness of its implementation in a real-world ecosystem.

A Readily Instantiable Cloud Security Architecture Featuring Open Source Components

This dissertation establishes a comprehensive economical security architecture that builds upon up-to-date protocols and open-source software and is readily instantiable and pertinent to requirements of concrete users (see Section 3.4). The feasibility of the architecture is highlighted in Section 4.6 by applying it within the CYCLONE ecosystem, deploying federated Bioinformatics applications within a cloud production environment. Section 4.6.3 additionally emphasizes the reduced management efforts in order to highlight the economic benefits of the architecture.

Chapter 2

Challenges, Approaches, and Related Work

The following subsections detail the four main challenges that are addressed by this dissertation. These challenges are related to the main TRESOR and CYCLONE project goals, the work items that I have been responsible for in the last years, as well as the publications that were authored in this period.

Each of the following sections first introduces the challenge in a general way before explicating its requirements and the related work in these areas. In order to provide a better understanding for the subsequent component design and development in Section 3, this section also summarizes how this thesis tackles these challenges.

2.1 Establishing User-centric Cloud Service Registries

When enterprises contract and consume cloud services, three activities need to be carried out: First, the services have to be *discovered* within the vastness of the Internet. Afterwards, the discovered services need to be *assessed* by matching them against business requirements. Finally, services need to be *selected*, that is, the best service has to be identified for subsequent booking and consumption. This can be done, for example, by making a shortlist and ranking services.

All of these activities present challenges for potential customers: First, the fast-moving, vast cloud market hampers the discovery of potential services. Second, highly ambiguous cloud service selection criteria contribute to laborious service assessment. At last, complex and opaque price structures and feature combinations contribute to ambitious comparison efforts.

As one of the major characteristics of Cloud Computing [107] is *on-demand self-service*, cloud consumers and providers are exempt from having human interaction in order to provision computing resources. Therefore, the description of cloud offerings becomes a crucial basis for service selection by cloud consumers. These service descriptions found within high-volume SaaS marketplaces, such as Salesforce AppExchange [144] and the Google Apps Marketplace

[68], rely on non-formalized information, e.g., free text, images, and some structured fields, e.g., author and category. While such content is appropriate for marketing purposes, other uses are impeded as unstructured text is insufficient for comprehensive search and uniform service comparison. Therefore, to carry out service selection activities in a satisfactory manner, the description of services needs to be formalized.

There are many proposals for cloud service registries that help users with these challenges, for example, cloud selection helpers, such as PlanForCloud[1] and CloudHarmony[2]. Additionally, there are approaches by academics, such as the service registry presented by Spillner and Schill in [165]. However, none of the existing proposals sufficiently meets common cloud user requirements: business pertinence, tooling simplicity and adaptability, versatile data retrieval, modeling capabilities, and service matchmaking functions.

It can be observed that approaches from the field of Semantic Web services are often based on abstract requirements and the application of generic concepts onto imaginary use cases. Instead, this thesis provides specific service registries based on concrete requirements of real-world use cases that are evaluated with stakeholders and potential end users. A good label for this philosophy is *user-centric*, which means that the business needs of concrete users are addressed by the results of this thesis: a comprehensive service description language, simple and adaptable software components, versatile data retrieval mechanisms, sufficient modeling capabilities for the target domain, and a constraint-based service matchmaker.

In line with Hevner's research method, this work focuses on the applicability of the research results, mainly through the involvement of all related stakeholders. This involvement includes group discussions at project meetings, partaking in a focus group on cloud service assessment, as well as presenting and discussing the work at diverse events as well as academic conferences. At last, some implementation and evaluation tasks were supported by students whose theses were supervised in the context of TRESOR and CYCLONE, such as [17], the integration of a service registry with a geographical information system, and [5], the creation and evaluation of a service selection questionnaire.

All evaluation methods gathered feedback that continuously refined and maturated the approach and helped in focussing on the main challenges, requirements, and constraints of practitioners in real-world use cases. These use cases also help to identify research gaps in existing approaches that do not fit well to the main practical challenges.

Adhering to the research method of Hevner et al., "progress is made iteratively as the scope of the design problem is expanded. As means, ends, and laws are refined and made more realistic, the design artifact becomes more relevant and valuable." [75]. Consequently, the research initially adressed the first use case before the work was iteratively expanded to the other five, continually broadening the scope of the approach as well as increasing the number of users for whom it provides a relevant contribution. Each additional use case can further grow the target group and therefore the value and significance of our approach.

[1]https://planforcloud.rightscale.com

[2]https://cloudharmony.com

The main contribution in the area of this challenge is therefore the description and evaluation of a novel service registry architecture and its implementation in diverse use cases, based on a thorough exposition and analysis of contemporary cloud service challenges. In effect, this contribution additionally supports other researchers and practitioners who are also tasked with the implementation of cloud service registries. In summary, the research implies that the simple and focused manner in which the registry architecture and its components are designed and implemented provides an expedient way to help users discover, assess, and select cloud services.

To affirm the applicability of the work, the following subsections clearly specify the challenge environment: first, the use cases in which the contributions were devised are described as well as the involved stakeholders. Second, the main challenge areas and stakeholder requirements that guide the subsequent activities are explained also. This section concludes with a summary and analysis of the work related to service registries. Later in this thesis, Section 3 also describes the proposed service registry components while Section 4 includes the performed evaluation activities.

2.1.1 Introduction: Use Cases, Stakeholders and Implementation Concepts

There are in total six main use cases in which cloud service registries were designed, developed, and evaluated in order to address related requirements and challenges. They represent both original use cases of the targeted projects as well as extensions thereof, for example, as a result of collaborations with other researchers and the supervision of student projects and theses in related areas.

Use Case 1: TRESOR Service Broker and Marketplace

Within TRESOR, a service registry was created that contains the descriptions of medical SaaS services so that the health centers can assess the suitability of these services to their specific requirements in detail. Throughout the project, this service registry was called the "TRESOR Broker". The implementation is supported by the SDL-NG, a flexible service description language, as well as a domain-specific business vocabulary that is focused on relevant aspects of medical SaaS offerings. The service registry provided an API for the "TRESOR Marketplace", a user-facing website where health centers can initiate, manage, and settle their service bookings.

Use Case 2: Cloud Storage Broker

The Cloud Storage Broker helps business and private consumers to better understand and compare cloud storage offerings, allowing them to select an optimal service for eventual booking. It is based upon the TRESOR Broker which was extended with a storage vocabulary and additional user interfaces for filtering and service comparison. The resulting implementation is partly based on the work of Knaack [87].

Use Case 3: CYCLONE IaaS Registry

In CYCLONE, DevOps engineers create deployment descriptions of their applications within the Nuvla Application Deployment Platform[3]. Nuvla queries a service registry for appropriate Infrastructure-as-a-Service (IaaS) offerings to allow DevOps teams to assess and choose a target service that meets their application deployment requirements in terms of, for example, CPU cores, memory, or network bandwidth. To create this registry, the TRESOR Service Broker was extended with additional APIs to interface with Nuvla as well as a domain-specific business vocabulary describing IaaS offerings, such as Amazon EC2 and Microsoft Azure.

Use Case 4: Open Cloud Computing Map

The Open Cloud Computing Map (OCCM)[4] displays and persists knowledge about geographical locations of cloud services. More specifically, it provides cloud service providers' computing centers' locations to its users as easily consumable graphical maps. The service registry was integrated into the OCCM by creating an OCCM data retrieval component which queries the registry for service information. Additionally, a way to map SDL-NG descriptions onto the RDF schemas used by the OCCM was also built. Implementation details can be found in [17].

Use Case 5: Open Service Compendium

The Open Service Compendium (OSC) allows any Internet user to describe, search for, and compare service offerings through a state-of-the-art interface. It provides a wiki-like experience for authoring service descriptions and therefore utilizes crowd-sourcing to extend the reach and impact of the approach.

Use Case 6: Dynamic Questionnaires and Property Statistics in the OSC

To help potential cloud service consumers to assess and select services in the OSC, it was extended by "static" and "dynamic questionnaires". The former are created manually by the cloud registry operators according to their knowledge about the services and the way their users select them. They are supported by a statistics module that allows analyzing the data to create meaningful questionnaires. The latter are created automatically on-the-fly, based on the available repository data as well as the answers given to previous questions.

As a result of both questionnaires, users are presented a list of services that is filtered based on the given answers. Users can also go to the list and filter it according to their needs. Implementation details can be found in [5].

Stakeholders and Implementation Concepts

The concrete stakeholders of each use case addressed by this thesis are generalized into three distinct stakeholder groups: *cloud service consumers*, *cloud service providers*, and *cloud service registry operators*. Cloud service consumers are potential customers of the cloud services offered by cloud service providers. Both rely

[3]https://nuv.la

[4]http://opencloudcomputingmap.org/

on a service registry that persists the *service descriptions* for each offering and supports discovery, assessment, and selection activities. A service description is a formal representation that describes properties of service offerings in the form of *service attributes*. These descriptions can be created by the cloud service provider or other interested parties. A *service description language* (SDL) specifies the underlying syntax, especially the way service attributes are defined and used. Each *business vocabulary* contains a set of concrete attributes that fit to its respective kind of service, e.g., storage or infrastructure services. Given these prerequisites, a cloud service consumer can interact with descriptions in different ways, ranging from simple browsing to complex *service matchmaking*. The latter is based on queries that are defined by consumers who are in need for particular cloud services. They can specify a set of constraints on attribute values that are desirable from their perspective. The evaluation of these constraints on a set of services and the subsequent creation of a ranked list is called *service matchmaking*.

The use cases feature two kinds of cloud service consumers: businesses and private individuals. For example, in Use Case 1, health center employees research about privacy-compliant operating room scheduling services, while Use Case 2 supports private individuals in their search for cheap cloud storage services to be used, for example, to store funny cat videos. Use Case 3 supports DevOps engineers looking for the cheapest cloud to deploy their applications. While the cloud service providers in all use cases are commercial, the components can also support non-commercial providers, such as internal departments or community clouds such as Cloudy [12]. In each of the use cases, the registry is operated by different entities. In Use Case 1, for example, the responsible company for the TRESOR ecosystem would also host the registry while the storage broker (Use Case 2) and the OSC (Use Case 5) should be hosted on community-donated resources in order to provide a system that is independent from any specific provider.

In the course of the work on this thesis, both a generic as well as several use-case-specific vocabularies were created. Attributes shared by all use cases are, for example, the service name, a textual description, the cloud model, and information about the provider. Each use case additionally features a domain-specific vocabulary. For example, cloud storage properties such as supported file types, maximum capacity, or available sharing and encryption mechanisms for Use Case 2. In Use Case 4, the vocabulary represents data center locations and distinctions between head office and subsidiaries.

2.1.2 Challenges and Requirements for User-centric Service Registries

After having presented the use cases, this section briefly describes the scope of this part of the thesis by iterating the challenges as well as the requirements for user-centric service registries. In [128], Pohjola and Kilkki show that a few essential features provide the majority of perceived user benefits of communication services. Every feature added on top contributes less to the perceived value than those before, while adding complexity to the service, impacting maintenance efforts, and making user interactions more laborious. Therefore, the implementation focused on the avoidance of *feature creep* and the resulting complex and hard to maintain *bloatware*.

In [155], insight is given how the scope of the approach presented in this thesis was continually limited to three main target challenges and five main requirements that are presented in the next subsections. The first step was a formal UML-based modeling process for analyzing user requirements. The project stakeholders and all involved programmers repetitively defined, presented, and refined the set of requirements, resulting in 186 reconciled project requirements. Still, for the limited project resources these were too many and additionally too broad in scope. Therefore, further prioritization lead to the selection of challenges and requirements that would be the target of the performed work. Three factors guided this process: the use case stakeholders' statements at project meetings and workshops, the feedback from other researchers and practitioners about external presentations, and the results of all supervised theses. In effect, this thesis does not address every conceivable challenge, yet it is on track to create a system that provides major benefits without being too bloated or complex to use.

The following sections detail the main challenges which resulted from the iterative concentration of the scope. Afterwards, the requirements of the stakeholders that should be met in order to address the aforementioned challenges are detailed.

Challenge 1: Fast-moving Vastness

The cloud market is *vast* and *fast-moving*. Current forecasts demonstrate its increasing *vastness*: the total end-user spending on public cloud services is expected to grow by almost 60% between 2015 and 2018 to a staggering $290bn[5]. Some cloud vendors are also astonishingly large: Amazon Web Services, for example, has more than 1 million customers, achieved more than 40 percent year-over-year revenue growth, and generates an estimated yearly revenue of $4 billion[6]. The Google Memorial[7] highlights the velocity of a fast-moving cloud market participant: it lists 66 discontinued services which were sometimes highly popular, for example, the Google Reader service had more than 24 million users[8] before it was suddenly discontinued in 2013. These examples highlight that the cloud market is too vast to obtain an optimal overview and it is too fast-moving to keep up with ever-changing service offerings.

Challenge 2: Ambiguous Criteria and Scattered Knowledge

Assessing service offerings raises two questions: *what criteria to use* and *where to get the required information*. Deciding what criteria to use is hard: they are sometimes highly ambiguous, for example, data privacy criteria as shown by Selzer [146]. Additionally, they are sometimes identified empirically, yet neither integrated into service description languages, nor existing marketplaces and registries. Gathering information about these criteria is also a challenging task: First of all, companies conceal knowledge about unfavorable service aspects. For example, cloud backup providers label services "unlimited", while they

[5]http://www.ft.com/cms/s/2/b3d40e7a-ceea-11e3-ac8d-00144feabdc0.html
[6]http://www.geekwire.com/2014/amazon-web-services-passes-milestone-1m-customers/
[7]http://www.lemonde.fr/pixels/visuel/2015/03/06/google-memorial-le-petit-musee-des-projets-google-abandonnes_4588392_4408996.html
[8]http://googlesystem.blogspot.de/2013/03/google-reader-data-points.html

have in fact bandwidth and storage limits[9]. Some of them do not state explicit limits, but instead impose a fair use clause that allows this provider to cancel the contract with a user anytime if this user, for example, reaches a user-unknown threshold. Other backup providers conceal their service limits deep in their *End User License Agreements* (EULA). Second, some companies provide insufficient information in order to follow strategic goals. Sometimes, only external websites such as private blogs offer workarounds, which are not always discovered by companies assessing these cloud services.

Challenge 3: Complex and Incomparable Features and Prices

In his seminal 1956 paper, Smith outlined that product differentiation and market segmentation are viable marketing strategies [161]. This observation still holds true more than sixty years later: to compete with cloud market leaders, service providers differentiate products and segment their market. One example is on-line storage, which is segmented into highly related categories, such as remote backup, cloud storage, and file sharing.

Different needs of consumers are addressed by several features and pricing schemes. For example, cloud storage services often allow flexible sharing of data but incur additional costs for extending the free quota. Backup service providers, on the other hand, allow unlimited data storage for a fixed price but have only limited sharing functionality. Thus, comparing different services becomes challenging if cloud consumers need to both share and backup large volumes of data. The price structure and feature combinations can also become complex: for example, when considering all VM sizes, regions, OS, and booking options (on-demand, 1Y/3Y reserved), there are over 10,000 different Amazon EC2 configuration variants and resulting costs.

Requirement 1: Business Pertinence

An important key requirement of all stakeholders is the *business-pertinence* of information systems. This denotes that service descriptions, business vocabularies, and the registry functionality are pertinent to the business domains, use cases, and related requirements of the targeted users. This thesis argues that this should be achieved through a thorough analysis of concrete use cases and the adoption of empirical insights such as studies on user behavior.

Requirement 2: Tooling Simplicity and Adaptability

Especially within resource-restricted environments, such as the health sector, any tooling complexity that leads to additional efforts has to be avoided. Examples are counter-intuitive and complex editors, service models that are far more extensive than they need to be, and a poorly maintained language implementation which is hard to install, maintain, and use. In this line of thought, [183] summarizes the impact of complexity and simplicity on implementation difficulty in in the context of Extreme Programming.

Furthermore, all parts of the information system need to be also adaptable to the rapidly changing aspects of cloud computing — with reasonable efforts. Besides the obvious area of service descriptions, the vocabulary is also subject

[9]http://pcsupport.about.com/od/software-tools/tp/unlimited-online-backup.htm

to change. At best, both should be easily modifiable using the same concept, for example, by using a textual domain-specific language.

Requirement 3: Versatile Data Retrieval

For every service, there are multiple data sources, often in a human-readable format, that should be included in the formal service description. For example, on-demand self-service cloud computing offerings often feature a user-pertinent documentation in the form of HTML pages. Furthermore, the different service prices, for example, spot prices, are published on the cloud service provider's website, making them a prime target for inclusion in a formal service description and their automated update on changes.

This reuse of existing data sources considerably reduces the creation and management effort of service descriptions. Therefore, the approach should include *scraping* functions that can extract data from human-readable sources as a dynamic part of a service description.

Requirement 4: Sophisticated modeling capabilities

Service descriptions may include complex attributes which require sophisticated capabilities to implement, mainly cost calculation and handling service variants.

When looking at the details of cost calculation in contemporary Cloud Computing, InfoWorld's contributing editor Peter Wayner states that "in some cases, the cost engineering can be more complex than the software engineering." [185]. Section [sec:theonewithamazon] previously illustrated the service variants' complexity of existing cloud offerings using the example of Amazon EC2 with its 10,000+ different variants.

Requirement 5: Service Matchmaking Functions

In the majority of cases, cloud service providers and consumers have incomplete knowledge and hence not all attribute values are included in the service descriptions and queries. Moreover, cloud consumers need the flexibility to express attribute values fuzzily, set priorities, and take the final service selection decision based on the ordered and evaluated list of services. All these requirements need to be integrated into the service matchmaking as they should be considered to find the most fitting services for a specific query.

2.1.3 Related Work

This section summarizes other works from related fields and contrasts them with the approach of this thesis. First, it gives an overview of the Semantic Web as this area features many works having similar goals. It is also most often mentioned by stakeholders, reviewers, and other experts when the work is presented to them. Secondly, existing user-relevant cloud service selection criteria are described that could be incorporated into the description language. As the approach includes domain-specific vocabularies and related service matchmaking, an overview about these topics is given as well. At last, commercial services are described that cover parts of the challenge areas before an overall analysis of the related work regarding the requirements concludes this section.

2.1.3.1 The Semantic Web

The Semantic Web was brought forward by Berners-Lee as early as 1994 as an extension to the then-emergent World Wide Web. The general idea is "allowing documents which have information in machine-readable forms, and allowing links to be created with relationship values" [24]. Nowadays, the W3C serves as an umbrella for Semantic Web technologies, for example, through the Semantic Web Interest Group[10] and the W3C data activity[11]. The following subsections describe a number of approaches relevant to this dissertation:

Semantic Web Services (SWS)
Service registries similar to those presented in this thesis play a major role in *Semantic Web Services (SWS)*. There are seminal works about SWS by Fensel et al. [51] as well as Studer et al. [168]. Studer et al. summarize the focus areas of SWS as reasoning-based matching of service functionality, harmonizing data formats and protocols, as well as automated Web Service composition [168]. The basis of SWSs are semantic service description languages, which exist in quite a large number, for example, the *Web Ontology Language for Web Services* (OWL-S) [103], the *Web Service Modeling Ontology* (WSMO) [143], *Semantic Annotations for WSDL and XML Schema* (SAWSDL) [49], the *Web Service Semantics* (WSDL-S) [3], and the *Semantic Web Services Language* (SWSL) [15], to name only a few. Furthermore, there were prominent European research projects maturating the area of SWS in the past, especially DIP [174], SUPER [37], and SOA4All [162].

Semantic Marketplaces
SWSs are envisioned to be traded on future cloud marketplaces, for example, on those postulated by Akolkar et al. [4], who emphasize the need for intelligence. Akolkar proposes to apply semantic technologies in "a vast knowledge base" using "recent advances in NLP (Natural Language Processing), Information Retrieval, and Machine Learning to interpret and reason over huge volumes" [4]. An example technology for this set of skills is the *DeepQA* software architecture for content analysis and reasoning [78] that is used in the famous IBM Watson computer program for artificial intelligence [77]. While this postulation has only limited immediate business-pertinence, Akolkar refers to the previous work of Legner [94], who asserts the need for "more sophisticated classification schemes which reflect the vocabulary of the target customers".

Recent research approaches sharing the mindset of Akolkar et al. are the *Web Service Modelling Ontology for the Internet of Services* (WSMO4IoS) and the *Linked Unified Service Description Language* (Linked-USDL). WSMO4IoS is based on the *Web Service Modeling Language* (WSML) and has been developed by Spillner and Schill with the goal to be "easily usable, freely available, versatile, extensible, and scalable" [165]. The authors claim that other languages fail to deliver these properties.

[10]https://www.w3.org/2001/sw/interest/
[11]https://www.w3.org/2013/data/

Linked-USDL is proposed by Pedrinaci et al. in [125] and extends its predecessor USDL presented by Oberle et al. in [117]. USDL is based on the Eclipse Modeling Framework [175] and was merged into the *FI-Ware Marketplace and Repository Generic Enablers*, which is a European initiative for building software ecosystems. These enablers provide APIs to manage USDL service descriptions and perform service matchmaking [55]. However, according to Pedrinaci et al., USDL failed to gain adoption due to "complexity, difficulties for sharing and extending the model" [126]. Because of these supposed flaws, they developed the advanced Linked-USDL, which is based on the *Web Ontology Language* (OWL) [184] and enables the integration of existing technologies.

Finally, Breskovic et al. propose to create standardized descriptions in order to establish electronic commodity markets for cloud services [30] by using formal concepts like the *Computing Resource Definition Language* (CRDL) [139]. Other work focuses on specific aspects of semantic service descriptions, for example, price and cost modeling as addressed by Kashef and Altmann [85, 6].

Other Semantic Applications

Hepp proposed the *GoodRelations OWL* [73], which contains a vocabulary for product, price, store, and company data. It is meant to be used as "an e-commerce extension for the schema.org vocabulary" [72]. In [141], Rodríguez-García et al. use natural language processing to automatically annotate cloud service descriptions as a basis for creating a semantic registry. However, searching is limited to topic-based queries instead of selection criteria. In [189], Zhang et al. create a declarative cloud recommender system based on a mash-up ontology using simple SQL queries on a relational database. In [190], the authors use a simplified analytical hierarchical process to find optimal IaaS offerings. However, their service descriptions only incorporate a limited set of numeric cloud characteristics and do not consider pertinent selection criteria. Furthermore, they neither iterate the requirements and constraints of their application area nor evaluate their system with concrete users.

Discussion of Semantic Web Research

Compared to the research presented in this thesis, the related Semantic Web research mainly differs in its scope and requirements. It can be observed that the majority tends to aim for the highest conceivable sophistication. For example, Fensel states in [51] the need to offer a significant automation degree of service discovery, ranking, selection, composition, invocation, as well as mediation facilities for data, protocols, and processes. Studer explains that the basis for SWSs should be "powerful, logic-based, representation languages" and that the general goal is to make these "semantics machine-processable" [168]. Recent implementations follow this mindset: Linked-USDL aims "to maximise to the extent possible the level of automation that can be achieved during the life-cycle of services" [125]. WSMO4IoS has the goal of "covering as many XaaS domains as possible" and "unify these services as much as possible while restricting the domain-specific service characteristics as little as possible" [165].

Additionally, there are fundamental issues with SWS. For example, in [86] Klan highlights the challenges of SWS technologies when used for service elicitation, for example, the argument why formal logic-based SWS languages are not appropriate for end-users. Following this view, the SDL-NG description language refrains from using any procedural representation of service knowledge. Instead, it solely relies on simple attribute/value assignments using properties that are relevant for the targeted stakeholders. Klan also states that the elicitation process needs to be incremental, as users "typically do not have a complete picture of the service functionality they desire". Especially the questionnaires in Use Case 6 provide such an incremental process.

This thesis assumes that the broad scope and the far-reaching aims of SWS research prevent the kind of solution simplicity and fitness for concrete challenges that is aimed for in this work. Considering this, the service registry research in this thesis results in simple tools and uses methods that limit its requirements to the essential needs of concrete end users. Even if it then cannot rely on advanced capabilities of semantic technologies, such as automated reasoning or ontology mediation, it also does not bear the associated complexity that occurs with a more comprehensive feature set.

The evaluation shows that this decision did not prevent the creation of repositories fitting well to the requirements of its users. Furthermore, the performance characteristics of the service registry components suggest both good scalability and resource efficiency. All of this highlights that user-centric service registries can be created well without relying on SWS technologies.

To conclude, there is an ongoing debate about the usefulness and feasibility of the Semantic Web in general: in [100], Floridi carries out an in-depth philosophical assessment and explains "why the Semantic Web won't work", citing issues such as its reliance on strong artificial intelligence if ambitious (and therefore technically unfeasible) or being a descendant of Leibnitz's "lingua characteristica" if being modest. There was also an intense public debate between Shirky [148] and Ford [56] about the promises and capabilities of the Semantic Web.

2.1.3.2 User-relevant Cloud Service Selection Criteria

Repschläger et al. have conducted two studies regarding relevant selection criteria of cloud consumers. In [135], they present selection criteria for SaaS based on literature review, an extensive market analysis, and an evaluation guided by expert interviews. The insights of these studies resulted in the development of a *Cloud Requirements Framework* (CRF), which is outlined in [136]. It provides a well-grounded conceptual basis for structuring SaaS, PaaS, and IaaS selection criteria. Another approach is the *Cloud Service Check* [57], which is a German catalog of cloud selection criteria with an extensive rationale providing guidance to assess different cloud service offerings. It is a result of the German research project Value4Cloud [58].

The *Service Measurement Index* (SMI) is provided by the Cloud Service Measurement Initiative Consortium [33]. It is far more extensive than similar approaches and, depending on consumer requirements, awards scores for each key performance indicator (KPI) a cloud service achieves. For example, the score for the *scalability* KPI represents the capacity of the service to handle the expected request volume, ranging from 0, a meager 10% capacity, and 10, an outstanding 190% capacity, being able to handle almost double the expected request volume.

As one of the requirements is business pertinence, the business vocabulary incorporates the selection criteria enumerated in the CRF. The rationale given in the Cloud Service Check could allow a registry to provide more comprehensive selection support to end users than just showing the raw data. For example, explaining users why and when offline capability is important for them is more beneficial than just informing them that a service can be used offline. At last, the data within service registries can provide the basis for implementing a partially automated service measurement according to the SMI, reducing required efforts of conducting such a measurement.

2.1.3.3 Domain Specific Languages (DSLs)

The presented approach relies on *domain-specific languages* (DSLs) to persist service descriptions and business vocabularies. Fowler defines a DSL as "a computer programming language of limited expressiveness focused on a particular domain" [59]. It is usually based on a host language such as C++, Scala, or Ruby, which leads to the differentiation between *external* and *internal* DSLs. An external DSL is a dedicated language having its own syntax, for example, regular expressions, SQL, and XML. It requires code translation into the host language to be understood. An internal DSL, on the other hand, uses the host language as a meta-model for its implementation and thus basically forces the developer to use the host language in a particular business-pertinent style. Examples are "fluid APIs" [48] and C++ template metaprogramming. Fowler recognizes a strong DSL culture in the programming language Ruby.

Specifically, the service registry architecture relies on an internal Ruby DSL as both the limited expressiveness, which results in reduced complexity, and the focus on a particular domain, for example, selection criteria or service attributes, matches very well with the overall scope and requirements.

2.1.3.4 Matchmakers, Marketplaces, and Selection Helpers

Service matchmakers focus either on numeric QoS properties or apply ontology-based matchmaking. The challenges for numeric approaches are the efficiency of interval comparison, partial matching, eliminating non-matching values, and comparing the distance of nodes, as highlighted in the works of Kritikos and Plexousakis in [91], Eleyan and Zhao in [47] as well as Zilci et al. [196]. Fuzzy set theory based approaches, such as [111] by Mobedpour and Ding, enhance numeric QoS approaches to allow service consumers to specify QoS queries as *good*, *medium*, and *poor* instead of specific values. Ontology-based approaches focus on matching the APIs and method signatures rather than QoS properties. Exemplary approaches are shown by Liu et al. in [98] and by Jie et al. in [82].

Representative high-volume SaaS marketplaces are Salesforce AppExchange [144] and the Google Apps Marketplace [68], based on their revenue and the number of users. Instead of an elaborate cloud service formalization, they utilize data models with a few attributes, such as free-text, images, provider info, and a categorization.

Cloud selection helpers support cloud users in assessing different aspects of cloud providers. One example is PlanForCloud[12] which allows users to create

[12]https://planforcloud.rightscale.com

deployment descriptions and specify their planned usage of servers, storage, and databases. CloudHarmony[13] is a bundle of services offered by Gartner and consists of a provider directory, a benchmark database for network performance, and a service status dashboard. CloudSpectator[14] offers performance measurements for different IaaS providers. Cloudorado[15] provides another selection helper for computing and storage with a high number of selection criteria.

2.1.4 Comparing Use Case Requirements to the Related Work

As a conclusion, this section now analyzes how the use case requirements relate to the work presented previously.

Requirement 1: Business Pertinence
The presented works in Section 2.1.3.2 are highly pertinent to the needs of businesses as they were elicited and validated in close collaboration with both users and consumers of cloud services. However, most of them cannot be modeled using ontologies from SWS and other fields as they are lacking the required classes and properties.

Furthermore, no prominent SWS project did focus on business pertinence for service consumers, providers, or operators. For example, the SUPER project enabled semantic modeling of business processes, but did not consider marketplaces or specific customer requirements [74]. The SOA4All project proposed the Minimal Service Model [127] that provides a lightweight high-level model for service operations and messages. The drawback of such high-level models is that they are far too broad to express specific service criteria such as payment methods and supported data formats. The authors address this issue by stating that other ontologies should be referenced for that purpose. However, they do not suggest any concrete ontology. Therefore, the presented high-level models do not provide an immediate benefit to the respective use case. At last, while the DIP project established an SWS broker [32] within a seemingly realistic e-government use case, there are some shortcomings: there is neither an in-depth requirements analysis nor do the authors iterate any design constraints of their target application area. The authors' "main aim was to test the advantages of SWS in term of interoperability" [140] and not how well the system performs under realistic design constraints.

Limitations in business pertinence also persist for cloud selection helpers. PlanForCloud contains only services supported by RightScale software. CloudHarmony is quite extensive, yet lacks information about pricing and other business-pertinent selection criteria. The criteria that are available to potential users are also limited, for example, PlanForCloud offers only price, while CloudSpectator supports only a multi core score. Cloudorado features an extensive list of criteria, some reflecting those from Section 2.1.3.2. Yet, there is neither an imprint nor any information about the company behind it. Furthermore, the comparison always favors servers from a specific provider (atlantic.net), so the neutrality can be put into question. None of these platforms offers matchmaking functionality.

[13]https://cloudharmony.com
[14]http://cloudspectator.com
[15]https://www.cloudorado.com/

Requirement 2: Tooling simplicity and adaptability

In [59], Fowler assembles a wealth of examples for both internal and external DSLs, emphasizing how DSLs enable adaptable software systems and raise developer productivity substantially. Quite a lot of DSLs show how the reduction of application scope creates a simple tooling. Examples are the Sinatra Ruby DSL[16] for creating simple web applications or the Apache Thrift Type and Interface Definition Language[17].

In the area of Semantic Web research there are many existing service editors and tools, for example, WSMO Studio [149], OWL-S Editor [46], Linked-USDL Editor [95], and the Internet Reasoning Services III (IRS-III) broker [32]. All of them are not aimed at regular Internet users, but should be used by other SWS researchers. This has severe implications on their simplicity and adaptability. Meanwhile, almost all aforementioned SDLs and related tools have been abandoned, including OWL-S Editor (2005), OWL-S (2006), SAWSDL (2007), WSMO (2008), IRS-III (2011), and USDL (2011). This has serious consequences for their use and adaptation in current and future technologies and platforms, due to the accrued technological debt in the years since their development discontinuation.

Requirement 3: Versatile data retrieval

Linked Data Principles [23] establish how to retrieve and aggregate data from external sources and use it to interlink thousands of datasets, for example, in the Open Mobile Network [182]. However, Linked Data is based on contemporary ontologies and query languages that do not allow the definition of scraping within a service description. Instead, all of it has to be performed using code that is separated from the service description. This has many drawbacks such as increased maintenance overhead and isolated repositories for either scraping code or service descriptions. In contrast to this, the proposed SDL-NG is based on an internal DSL and thereby allows the integration of program code and scraping logic into service descriptions very easily.

Requirement 4: Modeling capabilities

The general modeling capabilities within service marketplaces are very limited, as most employ merely unstructured text. Modeling *service variants* with structured languages such as DSLs, XML, or JSON, presents some challenges. Without having a rich variant model, describing real world cloud services becomes a major challenge, which has been imposingly demonstrated by a Linked-USDL service description for Amazon's EC2 service[18]. Although considering only one type of instance in one of Amazon's computing centers, the service description comprises not less than 1899 lines of semantic data. For the whole EC2 offering, the resulting service description length can be estimated to be in the range of 300.000 lines. This underlines the prohibitive complexity of real world semantic variance descriptions leading to inefficient processing and low comprehensibility. GoodRelations supports the modeling of product variants, which can reduce redundancies when different variants share the

[16]http://www.sinatrarb.com/

[17]http://thrift.apache.org/

[18]https://github.com/service-business-framework/Marketplace-RI/blob/master/src/main/webapp/rdf/cloudServices/Amazon_EC2_001.rdf

same attributes, but still requires the manual coverage of all variants separately. A promising approach is to use feature models to handle variability as proposed by Kang et al. in [84], which has been used for an external DSL in the FAMILIAR project proposed by Acher et al. [2]. A feature model represents the configurable qualities of services and their dependencies. As an example, it could be used to model the free and paid versions of a cloud service offering, containing the shared and distinct properties. However, so far no contemporary SWS approach relies on such feature models.

Requirement 5: Matchmaking

In [196], Zilci compares and analyzes the suggested service matching approaches to related requirements. The main observation is that the related work from academia has a strong focus on numeric Quality of Service (QoS) properties such as availability and response time [104], while the application areas of the use cases employ mostly textual descriptions, keywords, and category searches. Moreover, the related work on numeric QoS properties is highly fragmented: although each approach solves parts of the intervals and fuzzy QoS requests problems, there is no integrated solution for all query types. Service selection criteria that include enumerated types, for example, a list of predefined payment types, are not considered in related work. Moreover, the different types of constraints should be solved in a single constraint satisfaction problem (CSP).

2.2 Enabling End-to-end Security for HTTP Services with TCTP

This section begins with a technology "roundup" to introduce the problem area before analyzing the challenges and deriving relevant requirements. As a conclusion, TCTP is introduced "at a glance" and compared to the related work. In the course of this dissertation, Section 3.2 provides a detailed description of the designed protocol while Section 4.2 evaluates the implementation performance.

2.2.1 Introduction: Technology "Roundup"

This section provides a short technology "roundup" of the most pertinent technologies TLS, HTTP, and HTTP/S to prepare the subsequent content.

Transport Layer Security (TLS)

Transport Layer Security (TLS), which evolved from Secure Sockets Layer (SSL) [61], is a protocol that provides "privacy and data integrity between two communicating applications".[42] Besides encryption, it also includes the TLS Handshake Protocol which provides reliable authentication of both communicating parties as well as the secure negotiation of an encryption algorithm and cryptographic keys. The RFC states that TLS is "application protocol independent", and "does not specify how protocols add security with TLS". In effect, TLS is not confined to any conceptual layer of communication, for example, the OSI presentation layer, as it transforms arbitrary data into encrypted and authenticated TLS records.

Hypertext Transfer Protocol (HTTP)

"The Hypertext Transfer Protocol (HTTP) is a stateless application-level request/response protocol that uses extensible semantics and self-descriptive message payloads for flexible interaction with network-based hypertext information systems." [52, sec. 1]. Most often, HTTP is used with client/server messaging, that is, between "a program that establishes a connection to a server for the purpose of sending one or more HTTP requests" [52, sec. 2.1] (the client) and "a program that accepts connections in order to service HTTP requests by sending HTTP responses" [52, sec. 2.1].

In this scenario, two kinds of HTTP software interact:

- *User agents*: "any of the various client programs that initiate a request, including (but not limited to) browsers, spiders (web-based robots), command-line tools, custom applications, and mobile apps" [52, sec. 2.1]
- *Origin servers*: "the program that can originate authoritative responses for a given target resource" [52, sec. 2.1]

HTTP over TLS (HTTP/S)

HTTP/S was "designed to provide channel-oriented security" for HTTP [137, sec. 1]. "Conceptually, HTTP/TLS is very simple. Simply use HTTP over TLS precisely as you would use HTTP over TCP." [137, sec. 2]. The RFC is in fact very short, having only four pages of text in its main body. It contains some clarifications about client and server behavior when using HTTP/S, as well as the definition of endpoint identification and the "https" URI scheme.

2.2.2 Challenges and Requirements for HTTP Entity-body Security

The HTTP protocol also "enables the use of intermediaries to satisfy requests through a chain of connections" [52, sec. 2.3] and designates "proxies" and "gateways" (a.k.a. "reverse proxies") to reside between user agents and origin servers. Contemporary RESTful Cloud Computing ecosystems very often incorporate HTTP intermediaries, such as the Amazon Cloud Load Balancer [7], HAProxy [170], and the distributed cloud proxy presented in Section 3.3. Examples for intermediary functionality includes: load balancing, reverse proxying, SLA monitoring, audit logging, intrusion prevention, and request-based billing.

When a cloud service is accessed through HTTP/S these intermediaries have to act as the TLS server connection ends as they need to access the HTTP plaintext to operate on its contents. As current cloud management functions require access to URLs, HTTP methods, headers, and response codes, those could not be implemented if the intermediary would tunnel HTTPS traffic[101]. Repealing these intermediaries is also no viable option in contemporary cloud solutions, as this severely diminishes the capabilities for managing cloud consumption.

Challenges with TLS Server Connection Ends

Even though TLS connections achieve security on the transport layer, there are many issues with them having access to message plaintext. It is important to recognize here that none of the following concerns is limited to public, private, single-, or multi-stakeholder clouds:

- **Risk of losing confidentiality and integrity.**

  When intermediaries act as TLS server connection ends, the confidentiality and integrity of all HTTP communication is dependent on each of those intermediaries. As architectures become more complex and sophisticated, the risks rise with each additional intermediary.

- **Increased security efforts.**

  Intermediaries, as well as origin servers have to be protected with similar effort. Every additional service that is accessed through an intermediary makes a security attack onto these systems more worthwhile. Thus, the overall efforts of securing such architectures rise.

- **Legal obligations.**

  Within sensitive areas, provisions require end-to-end confidentiality of exchanged data records, such as those contained in an HTTP payload. Under German law, for example, disclosing health records contained in the HTTP payload to these intermediaries would be a criminal act[64].

- **Risk of unauthorized access.**

  The risk of unauthorized access using session hijacking and reusing intercepted credentials rises as intermediaries process cookies or HTTP Basic authentication as plaintext.

- **Need-to-know principle.**

  The basic need-to-know principle denotes that every component should only have access to the information that it needs to carry out its function. This principle is not met by such Cloud Computing architectures, as only the origin servers need to have access to the entity-body and not the intermediaries. However, most of the management functions carried out by intermediaries only need information contained in the HTTP headers and not the whole HTTP entity-body.

Requirements for Entity-body Encryption

In order to address the preceding issues, the entity-body encryption achieved by TCTP needs to meet the following seven requirements. These are detailed specializations of rather generic demands for TCTP: that it should be efficient, secure enough, and usable for TRESOR, CYCLONE, and similar project contexts:

- **Efficient presentation (R1)**

  Every decrease of transmitted or processed data positively impacts the performance of network applications. Thus, any entity-body encryption should employ the most efficient presentation language for the encrypted data.

- **Message-flow protection (R2)**

  Keyed hashing such as the HMAC scheme used in TLS (see [18]) can detect and prevent replayed or reordered encrypted content by authenticating messages. Any entity-body encryption should likewise avert these kinds of behavior to prevent security vulnerabilities.

- **Encryption capability discovery (R3)**

  To prevent additional round-trips for messages declining or requiring encryption, the encryption capabilities of a server should be discoverable before exchanging application data. This allows a user agent to decide for which URLs it is acceptable, required, or forbidden to send an encrypted entity-body.

- **Streaming capabilities (R4)**

  Entity-body encryption should not prevent HTTP streaming. That is, it should have the ability to authenticate and process fragments of an entity-body. This also permits handling messages as they arrive, which is more efficient than waiting until they are received completely.

- **Secure key exchange (R5)**

  The secure exchange of keys is a most basic requirement for an entity-body encryption. At best, this exchange should happen "in-band", for example, using the same communication channel as the subsequent encryption.

- **Algorithm negotiation (R6)**

  Different security algorithms should be negotiable to allow distinct capabilities and prevent using algorithms that are at a later time found to offer insufficient security.

- **Implementation support by existing software (R7)**

  Any mechanism for entity-body encryption would have to be added to user agents and origin servers. These implementation efforts can be considerably decreased if supporting libraries and components are available for applying the technology.

Further Considerations

Any secure on-line exchange of keys requires at least one round-trip to the server. Furthermore, unnecessary round-trips for failed requests, for example, requests where encryption is required but not applied or vice versa, can be reduced by an encryption capability discovery mechanism - at the cost of an additional one-time round-trip to the server. As the required number of round-trips for those functions is a conceptual constraint, further round-trip reduction is not considered as a requirement for HTTP entity-body encryption.

2.2.3 Related Approaches to Entity-body Encryption

There are in general two approaches to achieve end-to-end security for HTTP-based applications. Both are explicated in the following paragraphs.

Applying an Encryption Scheme within the HTTP Application

In this approach, applications use an established encryption scheme before transmitting ciphertext over a regular HTTP or HTTP/S connection. Two methods come to mind: S/MIME and XML Encryption and Signature:

- **Secure/Multipurpose Internet Mail Extensions (S/MIME)**

  S/MIME, as defined in [131], provides methods to sign and encrypt arbitrary data. Most often, it is used within electronic mail. However, it is not conceptually limited to this use case and therefore also applicable for HTTP communication.

- **XML Encryption [79] and Signature [14]**

  Both protocols are W3C recommendations for cryptographic functions on XML and arbitrary data. They are applied as SOAP [70] message encryption and signing within the WS-Security extensions[114]. As exemplified in the WS-Security based OSCI [89] protocol, TLS is often applied additionally in order to achieve transport layer security.

Using a Modified Communication Protocol
Additionally, there are some proposals for extended hypertext transfer protocols that address the issue of end-to-end security. The HTTPSec protocol provides a number of mechanisms to realize HTTP entity-body encryption, while being compatible to HTTP/1.1. The specification is not available on-line anymore, but still accessible through the Internet Archive in [60]. S-HTTP as defined in [138] is a protocol encapsulating and encrypting an HTTP request in an S-HTTP request. While this contradicts the basic prerequisite of giving intermediaries access to the HTTP plaintext for management, a modified version of S-HTTP is conceivable. It should be noted that there is no sign of these protocols being used at the moment, making those approaches quite obscure.

2.2.4 TCTP “At a Glance”

Before describing TCTP in detail in Section 3.2, this section provides a high-level overview providing enough detail for the subsequent analysis of the related work.

TCTP encrypts and authenticates the HTTP payload using TLS at the application layer. In effect, all headers are still accessible by Cloud Computing intermediaries while all of the issues mentioned in the preceding section are addressed by the entity-body encryption. Encryption keys and cipher suites are negotiated by wrapping the TLS Handshake Protocol into HTTP payload and sending it through the intermediaries. The reliance on this secure handshake minimizes the risk that any intermediary intercepting these messages can act as a man-in-the-middle and compromise TCTP security. The term “cloud” was included into the name of the protocol to highlight the most compelling application domain and not to limit TCTP to cloud environments.

Figure 2.1 provides an overview of communication secured by TCTP. The payloads of all HTTP messages are transformed into encrypted and authenticated TLS records, whose communication paths are represented by the black line. This encryption employs keys exchanged between user agent and origin server through the intermediary, denoted by the grey lines. The HTTP messages containing an unencrypted HTTP header and payload encrypted by TCTP are sent through HTTP/TLS connections, so that the headers are also secured on the transport layer. The intermediary has access to the HTTP headers which allow it to perform management functions while the transmission of the entity

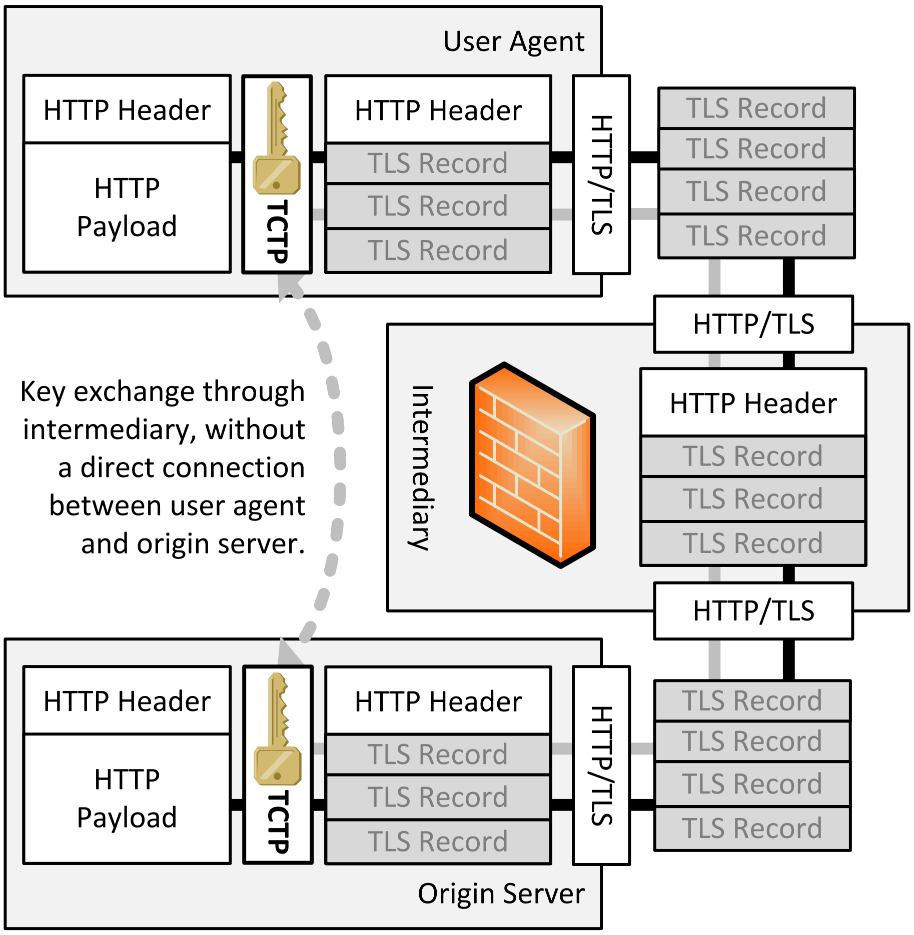


Figure 2.1: TCTP “at a glance”

body, for example, sensitive patient data, stays private between the origin server and the user agent.

In summary, there are four main benefits of using TCTP:

1. TCTP can be deployed immediately, as it is fully HTTP compliant
2. TCTP can be rapidly implemented, as required TLS libraries are widely available
3. TCTP does not introduce unknown security risks, as it relies on the mature TLS protocol
4. TCTP can be highly efficient, for the same reasons TLS is efficient in its binary message format, its key exchange as well as its encryption, which is even accelerated by specific instructions of modern CPUs

2.2.5 Comparing TCTP to Related Approaches

All of the related technologies present disparate deficiencies regarding the requirements for entity-body encryption, as summarized in Table 2.1. The symbols $\oplus$, $\odot$, and $\ominus$ specify the degree of requirement fulfillment, respectively "full", "partial" and "none".

Table 2.1: Evaluation of different entity-body encryption technologies

Req.	TCTP	S/MIME	XML Enc/Sig	HTTPSEC	S-HTTP
R1	$\oplus$	$\ominus$	$\ominus$	$\ominus$	$\ominus$
R2	$\oplus$	$\ominus$	$\ominus$	$\odot$	$\ominus$
R3	$\oplus$	$\odot$	$\odot$	$\odot$	$\odot$
R4	$\oplus$	$\odot$	$\odot$	$\ominus$	$\odot$
R5	$\oplus$	$\ominus$	$\ominus$	$\oplus$	$\oplus$
R6	$\oplus$	$\ominus$	$\ominus$	$\ominus$	$\oplus$
R7	$\oplus$	$\oplus$	$\oplus$	$\ominus$	$\ominus$

The following paragraphs detail the information contained in the table in the form [RX$\oplus$], where RX stands for the requirement and $\oplus$, $\ominus$ or $\odot$ specify the degree of fulfillment.

TCTP

TCTP fulfills all of the requirements: The TLS presentation language employed by TCTP is the most space efficient of all investigated technologies [R1$\oplus$]. HTTP application layer encryption channels protect the message-flow [R2$\oplus$]. The TCTP discovery mechanism allows encryption capability discovery [R3$\oplus$]. As the entity-bodies are fragmented by TLS, they can be processed in a streaming fashion [R4$\oplus$]. The TCTP handshake provides in-band secure key exchange with forward secrecy [R5$\oplus$], and algorithm negotiation [R6$\oplus$]. The implementation is supported by mature libraries (for example, OpenSSL [180]), operating system components (for example, Microsoft Windows Secure Channel [109]), and programming language integrations (for example, Java Secure Socket Extension (JSSE) [120]) [R7$\oplus$]. Building upon widely used software accelerates the implementation of TCTP, as shown by the TCTP prototype in Section 4.2.

Secure/Multipurpose Internet Mail Extensions

The verbose headers and message structure of S/MIME are not as efficient as TCTP [R1$\ominus$]. Binary representation is possible, but 7bit encoding is recommended by the RFC. Signing operates only on single messages [R2$\ominus$]. A discovery mechanism is not specified, but could be realized by HTTP content negotiation, which would require additional round-trips to the server [R3$\odot$]. To support streaming, messages would have to be separated into MIME multipart segments incurring additional effort [R4$\odot$]. The RFC does neither specify secure key exchange [R5$\ominus$] nor algorithm negotiation [R6$\ominus$]. There are a number of S/MIME libraries, for example the JBoss RESTEasy framework [81] for entity-body encryption in RESTful applications [R7$\oplus$].

XML Encryption and Signature

XML Encryption and Signature embodies XML and either Base64 or XML-BOP MIME overhead [R1$\ominus$]. Signing operates only on single messages [R2$\ominus$].

Regarding Discovery the same conclusions as for S/MIME hold true [R3$\odot$]. As with S/MIME, XML data has to be costly separated to be processable in a streaming fashion [R4$\odot$]. "XML Encryption does not provide an on-line key agreement negotiation protocol" [79, sec. 5.5, par. 2] [R5$\ominus$]. Algorithms cannot be negotiated [R6$\ominus$]. The implementation is supported by a number of libraries [R7$\oplus$].

HTTPSec

HTTPSec uses verbose HTTP headers containing Base64 encoded content [R1$\ominus$]. The encrypted data can be in binary form. For message-flow protection only a weak message counter is fed to the MAC computation [R2$\odot$]. Only ad-hoc discovery is specified [R3$\odot$]. Streaming is not possible, as the MAC is part of the HTTPSec header [R4$\ominus$]. Secure key exchange is possible [R5$\oplus$], but not algorithm negotiation [R6$\ominus$]. There are neither existing HTTPSec implementations nor supporting libraries [R7$\ominus$].

Secure HTTP

First of all, S-HTTP messages are not compatible to HTTP and therefore cannot be processed by any Cloud Computing intermediary. As of now, this would prohibit its use for entity-body encryption. However, it is certainly a related technology that can be sensibly analyzed. S-HTTP security headers are verbose [R1$\ominus$] while encrypted data can also be in binary form. S-HTTP does not protect the message flow [R2$\ominus$]. S-HTTP only allows for ad-hoc discovery [R3$\odot$]. S-HTTP specifies a MAC header computed over the whole entity, which prevents streaming, but the underlying CMS by now supports the fragmentation of data and could therefore allow streaming processing, if the RFC would be updated [R4$\odot$]. Secure key exchange [R5$\oplus$] and algorithm negotiation [R6$\oplus$] are supported. No implementation of S-HTTP, nor supporting libraries could be found [R7$\ominus$].

2.3 Managing Cloud Service Consumption through a Distributed Cloud Proxy

One of the main challenge areas in cloud ecosystems is providing required assurances to all stakeholders to allow secure and trustworthy cloud computing, for example, in the health domain. This requires management of the end-users' cloud consumption by introducing cloud proxies as trusted and secure mediators. Before presenting the approach of this thesis, this section highlights the requirements that need to be met in this challenge area.

2.3.1 Requirements for Proxies in Cloud Ecosystems

There are four fundamental requirements for cloud proxies that need to be met in order to use them successfully within cloud ecosystems. These requirements were extracted and generalized from the 178 TRESOR project requirements which have been compiled in diverse workshops. Through this involvement of cloud stakeholders, they can be considered well grounded in practice.

Requirement 1: REST-compliant HTTP processing

One of the constraints of cloud proxies is that they need to be compliant to RESTful applications [54] and JSON APIs that use HTTP as a communication protocol. When starting the work in this area in 2012, the largest directory of public cloud APIs identified 70% of all listed as being based on REST. [129] Currently, in 2017, the prevalent rise of Microservices and JSON APIs continues the trend of avoiding SOAP in common cloud APIs, see for example [116].

Requirement 2: Managing cloud consumption

HTTP is well suited to control service consumption by proxies, for example, resources are addressed by URIs which allows enforcement of authorization rules based on matching those URIs. Using HTTP in a RESTful manner provides a unified mechanism making proxies universally applicable for any cloud service. This is unlike SOAP/RPC which employs application-specific interfaces which would require extensive modification of proxies for each service. Meaningful REST URIs also allow generic capabilities, such as resource-based logging and accounting for any service. Furthermore, HTTP includes information that can be integrated into the SLA monitoring, for example, the HTTP status code which conveys the distinction between successful and failed requests.

Requirement 3: Provide implicit security guarantees

The management of cloud service consumption through cloud proxies enables service providers to rely on implicit guarantees, such as the correct client authentication or the compliance to all policies of service clients. This can release cloud services that are accessed through such proxies from many duties, for example, the implementation of AAA functionality. In this case, the proxy would either locally authenticate users or rely on existing single sign-on (SSO) solutions, such as Open ID Connect [118].

Requirement 4: Independent management

Cloud proxies should be distributed in such a manner that management capabilities can be fulfilled by a 3rd party that is independent from consumer and provider. This is beneficial especially for the monitoring of SLAs: as pointed out by Koller et al. [88], an independent party can monitor and enforce SLAs more trustfully than the participating parties could do by themselves. Having an independent party also helps with the debugging of problems, as it can help pinpointing potential issues to either the consumer or the provider.

2.3.2 Related Approaches

Weissman et al. stated in [186] that “enabling proxies to assume multiple roles is key to the performance and reliability of distributed data-intensive multi-cloud applications”. In line with this thought, some multi-role distributed proxies can be found in literature, for example, to mitigate constraints of mobile devices as shown by Zhu, et al. in [192], and realizing a certificate-less re-encryption scheme, demonstrated by Wu, et al. in [187]. In practice, there are many useful proxy implementations limited to certain tasks, such as HAProxy

[170], a widely used load-balancer, Squid [166], a content cache, and Tor [130], an anonymization software.

2.3.3 Distributed Cloud Proxy: Concept

The main idea behind the distributed cloud proxy is the establishment of three instances of the same proxy at each involved party, the *cloud consumer*, *trusted party*, and the *cloud provider*. Each instance provides complementary functions, for example, TCTP client and server facilities, monitoring of stakeholder-relevant information, and backend integration (e.g., AAA systems). The communication is secured end-to-end with TCTP while the point-to-point communication uses TLS. Figure 2.2 presents an overview about this concept.

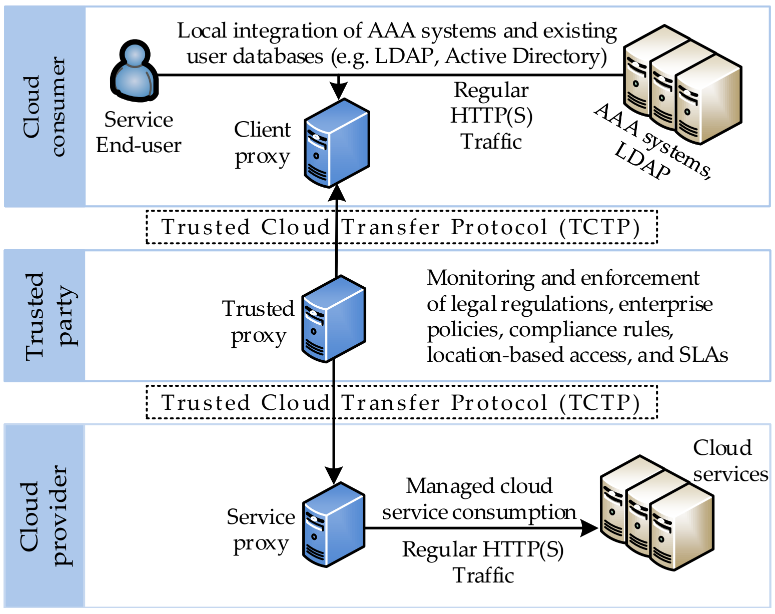

Cloud consumer
Local integration of AAA systems and existing user databases (e.g. LDAP, Active Directory)
Service End-user
Client proxy
Regular HTTP(S) Traffic
AAA systems, LDAP
Trusted Cloud Transfer Protocol (TCTP)
Trusted party
Trusted proxy
Monitoring and enforcement of legal regulations, enterprise policies, compliance rules, location-based access, and SLAs
Trusted Cloud Transfer Protocol (TCTP)
Cloud provider
Service proxy
Managed cloud service consumption
Regular HTTP(S) Traffic
Cloud services

Figure 2.2: Cloud proxy distribution, adapted from own figure in [172]

From a conceptual point of view, the distribution of functionality is the following:

- The proxy at the cloud consumer serves as an integration point to the local systems - without exposing these to the Internet. There are many potential services the could be integrated with the client proxy, for example, local AAA systems, a local policy store, and a company auditing system. This client proxy is accessed via regular HTTP(S) traffic and invokes the TCTP key exchange and encryption with the service proxy.

- The proxy at the trusted party provides centralized security functions, such as monitoring and enforcement of legal regulations, implementing enterprise policies, and supervising Service Level Agreements (SLAs). As the traffic is protected by TCTP, it can manage the traffic without having access to the confidential parts of the traffic, the HTTP entity-bodies.

- The proxy at the cloud provider is the TCTP server connection end. There are implicit guarantees given about the traffic coming to this proxy, for example, that the preceding proxies have relayed communication compliant to all legal and organizational requirements of the cloud consumers. The provider proxy decrypts TCTP traffic so that the communication encryption is transparent to the cloud services. This transparency is very beneficial for the cloud services as they do not need to be extended, for example, with TCTP server components.

2.4 Secure Management of Federated, Multi-cloud Application Deployments

Providing an architecture to securely manage federated, multi-cloud application deployments is the main challenge in this area. Such an architecture is highly relevant to realize two important characteristics of contemporary cloud services: *multi-cloud deployment*, for example, to increase resiliency and reduce latency, and *authentication and authorization using federated identities*, for example, to reuse identities provided by academia (eduGAIN) and social networks (e.g., Facebook). Within this area, three stakeholder groups are interacting: **Cloud Infrastructure Providers** who provide cloud resources to **Application Service Providers** to deploy applications for **Cloud Application End Users**.

2.4.1 Requirements for Secure Application Deployments in Cloud Ecosystems

This section presents the three most pertinent requirements for the challenge area. They originate from the CYCLONE innovation action and represent some of its goals. Having also been observed by all project partners in practice, they can therefore be considered highly relevant for contemporary cloud ecosystems.

Requirement 1: Federated authentication and authorization

Federated academic identities are quite common in cloud environments to access scientific online libraries and shared research infrastructures. Using Facebook and Google identities to access personal cloud applications is also quite popular nowadays. In fact, all stakeholder groups require their use: cloud infrastructure providers for reusing preexisting identities for administrative log-ins as well as application service providers to attract end users who can easily reuse their identities in a practical and secure manner. Especially for data sharing by end users, authorization by federated identities can be way more trusted than anonymous username/password pairs.

Requirement 2: Secure multi-cloud application deployment

Multi-cloud deployment by Application Service Providers offers many benefits, for example, lower latency through global server distribution, and higher resiliency when using more than one cloud provider.

Requirement 3: Unified logging for distributed systems

As both the cloud infrastructure as well as many cloud applications are extensively distributed, gathering diagnostic messages and performance metrics from all of these applications in a unified system is very challenging. This requirement is especially relevant for any cloud ecosystem as providing service logs is oftentimes a very important functionality for ecosystem participants. Without this functionality, debugging applications and providing audit trails for data privacy reasons becomes a very tedious endeavour.

Nonfunctional requirements: solution qualities

There are four main solution qualities that qualify as nonfunctional requirements:

- **Business relevance:** A high *relevance* for contemporary cloud environments should be achieved through a requirements analysis of concrete use cases instead of conceptual considerations.
- **Immediate instantiability:** All components of the security architecture should be published as open source to make the instantiation almost instant.
- **Comprehensibility:** To make it easy to follow the underlying ideas and take up the security architecture, there should be a large volume of comprehensible supporting material.
- **Maturity:** To build a stable and mature architecture that can be applied within production environments, it should rely on established software as well as common protocols and libraries.

2.4.2 Related Technologies

After establishing the stakeholders and their requirements, the next subsections give an overview about the technologies related to the challenge area.

OpenID, OAuth, and OpenID Connect

Accessing resources on Web 2.0 platforms on behalf of other resource owners - without handing over usernames and passwords - provided the first use case for federated web-authentication and authorization. For this purpose, OAuth, OpenID, and the recent OpenID Connect were introduced: OpenID[19] specifies how relying parties “prove that an end user controls an identifier” without disclosing credentials to relying parties.

Resource access requests can be expressed by *OAuth*[71] which enables “a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner [...] or by allowing the third-party application to obtain access on its own behalf”, most commonly through the OAuth Authorization Code Grant flow. Basically, *relying parties* use HTTP redirection to request an access token from the resource server. If users accept this request, the resource server issues a token that allows relying parties access to users’ resources (see 4.1 of [71]).

[19]http://openid.net/specs/openid-authentication-2_0.html

The most recent authentication protocol, OpenID Connect, focuses on solving security issues when using OAuth for authentication[20]. OpenID Connect uses JSON Web Tokens (JWTs)[21] for transmitting user claims. As a result, the OpenID Connect Authentication Code Flow (OIDCACF) combines JWT and OAuth to provide secure web-based single sign-on for contemporary web applications.

SAML, eduGAIN, and Compatible Implementations

The OASIS Security Assertion and Markup Language (SAML)[22] "defines the syntax and processing semantics of assertions made about a subject by a system entity". It incorporates Web Service technologies, such as XML, XML Encryption & Signature, and SOAP. Version 2.0 adds HTTP bindings to use SAML without SOAP. The GÉANT eduGAIN service "interconnects identity federations around the world" through providing a metadata aggregator for inter-federation service access between 38 participating federations, 2093 Identity-, and 1208 Service Providers.

Shibboleth[23] is an open source Discovery Service, Identity-, and Service Provider implementation, based on SAML 2.0, extensively deployed in academic institutions worldwide. The ***SimpleSAMLphp***[24] Identity- and Service Provider additionally supports OpenID and OAuth. Keycloak[25] provides "Integrated SSO and IDM for browser apps and RESTful web services" and implements all standards previously mentioned. It offers an Identity Broker, integration with Active Directory and LDAP, as well as a rich set of libraries for different implementation platforms.

PAM, XACML & Moonshot

The Linux Pluggable Authentication Modules (PAM)[26] subsystem provides a simple API to offer policy-compliant authentication, authorization, and accounting to relying software, such as Secure Shell Server (SSH) or getty processes. The Extensible Access Control Markup Language (XACML)[27] provides an XML-based policy language and a distributed access control architecture. It uses a set of *subject attributes*, for example, group membership or confidentiality level, to authorize *actions* carried out on arbitrary *resources*, which is called *attribute-based authorization*. The Moonshot project[28] "aims to enable federated access to virtually any application or service". One of its main components is a federation-enabled version of OpenSSH. However, relevant work ceased at the end of the last pilot in March 2015. Now, Moonshot-provided software packages are outdated and insecure due to a lack of patches for recent vulnerabilities and therefore unsuitable for any production environment.

[20]For a comprehensive discussion, see http://oauth.net/articles/authentication
[21]See http://jwt.io and [83]
[22]http://saml.xml.org/saml-specifications
[23]http://shibboleth.net/
[24]https://simplesamlphp.org/
[25]http://keycloak.jboss.org/
[26]http://www.linux-pam.org/
[27]https://www.oasis-open.org/committees/xacml/
[28]https://wiki.moonshot.ja.net/

ELK distributed logging stack

The ELK logging stack is one of the most widely used software stacks to implement distributed logging. It consists of three software components which form the ELK abbreviation:

- **E**lasticsearch[29]

  Elasticsearch is an open-source distributed search engine, based on Apache Lucene[30]. It operates on schema-free JSON documents and is therefore highly flexible. Within the ELK stack, Logstash uses Elasticsearch to persist log messages while Kibana uses it to retrieve these logs.

- **L**ogstash

  Logstash is a well-performing logging middleware which is used to uniformly receive, transform, and persist log messages from a diverse set of backend services. Out of the box, Logstash supports over 40 different inputs, for example, Syslog, "JSON data over a socket", Twitter Streams, and any JDBC data source. It converts each input message into a common format which can be persisted in diverse back ends, most often Elasticsearch.

- **K**ibana

  Kibana can be used to offer comprehensible and highly functional logging dashboards when provided with a configuration that fits well to the available logging data in Elasticsearch. Available dashboard components include line, area, bar, and pie charts, data tables, heatmaps, and others. These components can add dynamic filters to the displayed data, effectively providing a "drill-down" functionality.

[29]https://www.elastic.co/products/elasticsearch
[30]https://lucene.apache.org/

Chapter 3

Component Design and Development

After providing the application environments as well as the challenges, approaches, and the related work, the following subsections provide details about the components that were developed in the context of this thesis: the cloud registry architecture, the Trusted Cloud Transfer Protocol, the distributed cloud proxy, and the multi-cloud security architecture.

3.1 Cloud Service Registry Architecture

After iterating the application domain and the user requirements as well as describing and analyzing the related work in Section 2.1, this section now explicates the design of the cloud service registry as well as its implementation in the diverse use cases. Afterwards, it presents an analysis how the resulting architecture meets the stakeholder requirements by contrasting each other. The conclusion consists of providing some further remarks to help understand the performance characteristics of the service registry architecture.

The service registry architecture that is shown in Figure 3.1 includes six main components that are explained in detail in the following subsections, along with their implementation in the use cases. As the components are loosely coupled they can be scaled independently, based on the concrete performance demands of the use cases.

Figure 3.2 presents an overview of the interactions between these components in the form of a UML communication diagram. This diagram presents the two main interactions "Service Modification" (1.) and "Service Query" (2.).

To create or modify services, users instruct the client to send an HTTP POST request to the Rails backend (1.1). The backend receives the description and pushes it to the Redis Job Queue for processing (1.2). The service evaluator pops the queue (1.3) and pushes the resulting service description document to the database (1.4).

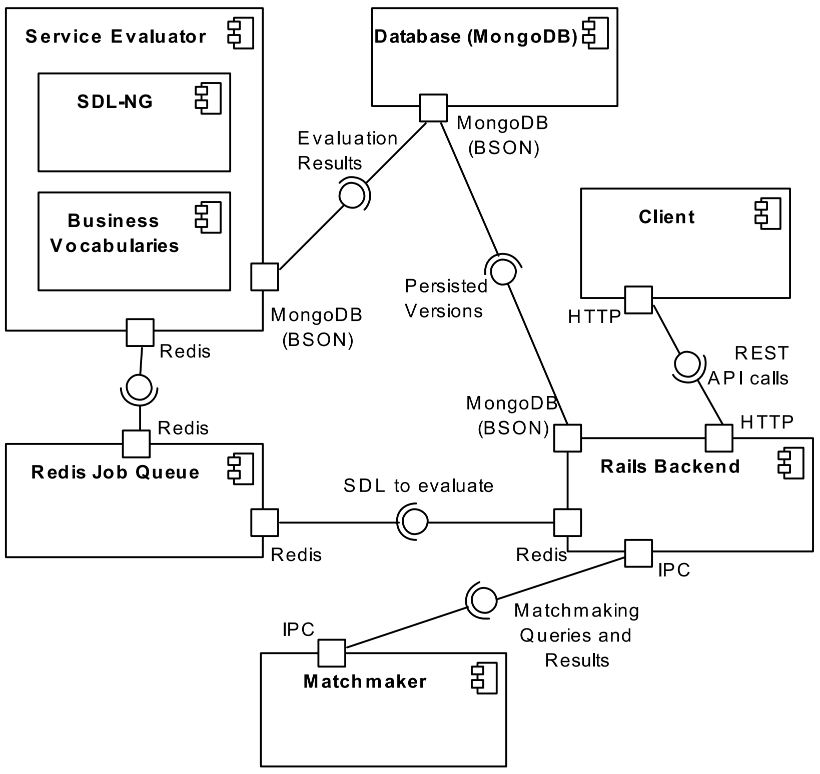


Figure 3.1: Cloud Service Registry Architecture

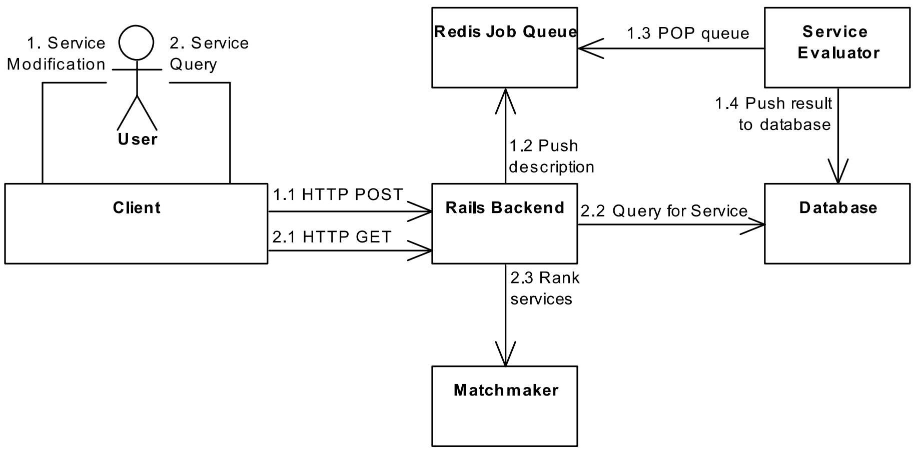


Figure 3.2: Service Registry Communication Diagram

After services are persisted in the registry, it can be queried (2.). Similar to 1., the client transmits an HTTP GET request to the Rails backend (2.1). The Rails backend transforms the GET query parameters into database filters, which it includes in the database query (2.2). As an optional step, the services can be ranked using the matchmaker (2.3) before returning them to the client.

3.1.1 SDL-NG

The SDL-NG is a Ruby-based internal service description language featuring a terse syntax, rich type system, utility functions for scraping HTML documents, and export facilities for XML, JSON, HTML, and RDF. It is used to specify service vocabularies well as using these for the description of cloud service offerings.

The following subsections provide a thorough explanation of the SDL-NG as its creation was one of the most time-consuming undertakings that were followed through this dissertation. First, the design and basic framework structure are explained, before detailing the implementation of the diverse functions and providing examples for SDL-NG use.

3.1.1.1 Design Considerations

The design of the SDL-NG framework is mainly motivated by Requirement 2, enabling simplicity and adaptability. A DSL lowers description and language definition efforts considerably, especially in relation to service description languages using semantic technologies. From past experience using Ruby as well as other programming languages comes the presumption that a Ruby-based DSL can provide a simpler and more usable tooling than the language tools of WSMO4IoS and Linked-USDL. As the service description is in fact program code, it can be augmented by libraries to implement the integration of existing sources, for example, by scraping HTML documents for feature descriptions. Another benefit of using Ruby is the low "syntactic noise", that is, the overhead characters not conveying semantic meaning (semicolons, brackets, etc.) are considerably lower in Ruby than in other programming languages, such as Java and C++, providing less distractions when reading and understanding program code.

3.1.1.2 Basic Framework Structure

The SDL-NG uses the very common RubyGems[1] package management system, included in every major Ruby distribution, for packaging and deployment. Within a `.gemspec` file, Ruby libraries define a rich set of metadata, for example, their dependencies, name, license, and version. The main source for Ruby libraries is the RubyGems community hosting service that also hosts the SDL-NG[2]. After installing Ruby, potential library users can issue a simple `gem install sdl-ng` command to download and install the SDL-NG and all required libraries.

The framework consists of five directories that are explained in the following:

[1]https://rubygems.org
[2]https://rubygems.org/gems/sdl-ng

`bin`

A common convention for RubyGems is having all executable scripts inside the `bin` directory. As the SDL-NG is rather a framework used within other programs, it does not provide any immediately executable program that would provide any usable function to the end-user. Instead, the `bin` directory contains the `process_service_descriptions` script which allows to "testdrive" the SDL-NG. It recursively loads all vocabulary files (`*.sdl.rb`) from the current directory before instantiating all service descriptions (`*.service.rb`). Afterwards, it exports the vocabulary as JSONSchema and XSD and all services as XML, JSON, RDF, and HTML. This script has shown to be quite helpful for understanding and studying the SDL-NG, for example, by students who authored theses using the SDL-NG. It was also modified to provide some performance figures about the SDL-NG in Section 3.1.10.

`examples`

This directory contains all SDL-NG artifacts that have been created over the course of the last years: service descriptions (`services`), vocabularies (`vocabulary`), translations (`translations`), and RDF mappers (`rdf_mappers`). More information about the examples can be found in Section 3.1.2 which explains the structure of the vocabularies and Section 3.1.1.15 which highlights the RDF mapping mechanism.

`spec`

The `spec` directory contains a suite of behaviour-driven tests, based on the widely used Ruby RSpec BDD library[3]. Throughout the development, having such a suite was beneficial in two aspects: first, it allowed to start development with the definition of the expected behavior of the SDL-NG which closely matched the established design ideas before working on their implementation. Second, it helped with iteratively extending the framework with new functionality while still ensuring compatibility to the services described previously.

`lib`

The `lib` directory contains the code of the framework, structured so that it is compatible to the *autoload* mechanism implemented by `ActiveSupport`[4]. Autoload eases the programming in Ruby by automatically loading unknown modules on their first use from source files that are named similarly, for example, the class `SDL::Base::Type` will be loaded on first use automatically from the file `lib/sdl/base/type.rb`.

The modules of the framework are structured according to their responsibilities:

- `SDL::Base` defines the most basic framework classes `Type` and `Propery`. Section 3.1.1.4 explains this basic relationship. The module also contains additional utility classes:
  - `ServiceCompendium` is a facade that eases interacting with the SDL-NG (see Section 3.1.1.6)

[3]http://rspec.info/

[4]http://api.rubyonrails.org/classes/ActiveSupport/Autoload.html

  - `PropertyClassification` provides a classification scheme for service properties (see Section 3.1.1.8)
  - `URIMappedResource` maps SDL classes to URIs (see Section 3.1.1.9).

- `SDL::Exporters` hosts all classes responsible for exporting SDL-NG data to HTML, XML/XSD, JSON/JSONSchema, and RDF. The mechanisms for exporting data is brought forward in Section 3.1.1.12.
- `SDL::Receivers` is designated to the `TypeInstanceReceiver` class that provides the execution context for service and vocabulary description files. Section 3.1.1.7 explains the relationship of this class to the lifecycle of service descriptions and vocabulary definitions.
- `SDL::Types` contains classes to reflect commonly used value types, e.g., boolean, numerical, and strings. Section 3.1.1.10 iterates all classes available for use in a description.

`util`

The `util` directory contains two Ruby source files that provide utility functions to the framework:

- `documentation.rb` implements all functions related to the documentation mechanisms of the SDL-NG. It is explained in more detail in Section 3.1.1.13.
- `nokogiri.rb` provides the `fetch_from_url` function that can be used to scrape HTML documents for description content. More details can be found in Section 3.1.1.14.

`translations`

The `translations` directory provides English translations for framework-defined documentation keys, e.g., comments explaining the structure of the XML Schema Definition for better comprehension by potential users.

3.1.1.3 SDL-NG Example and Overview

Listing Listing 3.1 provides an excerpt of the service description for Google Drive for Business. After a first look, the language appears to consist of space-separated key/value pairs. However, it is in fact a Ruby program that contains invocations of instance methods of the `Service` class which is provided by the SDL-NG. For example, the first line in the listing invokes the `Service#service_name` instance method and provides it with the "Google Drive for Business" parameter. This method was dynamically defined by the SDL-NG framework when it loaded the business vocabulary at an earlier stage. All these dynamic methods are responsible for setting the instance variables to the value of the given parameter, similar to "setter" methods found in other programming languages, such as Java and C#. In this example, the method sets the `@service_name` instance variable of the respective `Service` instance to "Google Drive for Business".

The SDL-NG supports different value types: simple types (such as strings and booleans), complex types (such as `ChargingUnit` or `AddOnRepository`), and lists of both types. The example listing shows all of them: a simple string for the `service_name` property, a complex type `CloudServiceModel` for the cloud_service_model property, as well as the list of compatible browsers for the compatible_browsers property.

For easier use of predefined type instances, they can be referenced by a simple identifier, for example, saas (as a CloudServiceModel instance) and firefox (as an instance of Browser). Here, saas is a method invocation that returns a predefined instance of the same name. Some type instances can be created on the fly, for example, the AddOnRepository type that is instantiated in the listing with a list of two values: a string for the repository url as well as an integer to represent the number of addons in the repository.

In addition, the lower part of the excerpt shows how scraping external resources through Ruby code is seamlessly integrated into the SDL-NG. First, the the method fetch_from_url is invoked, which fetches an HTML document from a URL and invokes an anonymous function for each element that matches a selector. The example uses 'h3' which returns a list of headings. For each heading the feature property setter is invoked, which adds a new feature to the service, taking the content of the heading as a title and the following paragraph ('\simp' matcher) as the feature description. The result of the SDL file invocation is a new Service instance that contains a wealth of static information as well as a comprehensive textual feature description that was dynamically scraped from the Internet.

Listing 3.1 Service description excerpt (truncated URLs for readability)

```ruby
service_name 'Google Drive for Business'
cloud_service_model saas

add_on_repository 'https://...', 1000
status_page 'http://...'
public_service_level_agreement 'http://...'
documentation 'http://...'
rest_interface 'https://...'

is_charged_by user_account
is_billed monthly
is_charged in_advance
payment_option credit_card

compatible_browser firefox
compatible_browser chrome
compatible_browser internet_explorer, '9'

# Fetch a list of features from the Google Apps page
fetch_from_url('https://...', 'h3')[1..4].each do |header|
  # Extract Google Apps Features
  feature header.content.strip, header.search('~p')[0]
end
```

3.1.1.4 Types and Properties

`Type` and `Property`, the foundational classes of the SDL-NG, are regular Ruby `Class` descendants. In the SDL-NG, `Type` descendants represent any structured information, for example, `Services`, `Providers`, and `Locations`. The attributes of each `Type` are represented by `Property` instances. Properties can be "multi-valued", that is, they can represent a list of values. In general, properties represent either "simple values" (e.g., a number, string, or boolean value) or "complex values", that is, `Type` instances. The syntax for type and property definition is as follows[5]:

- Adding properties to any type definition

  `type :<identifier> do <definitions> end`

- Alternative syntax for adding properties to `Service`

  `service_properties do <definitions> end`

- Adding a single property to a type

  `<type> <name>`

- Adding a multi-valued property to a type

  `list_of_<plural type name> <name>`

- Setting a property value of a type instance

  `<property> <value>`

- Setting a multi-value property of a type instance

  `<singular property name> <value 1>`

  `<singular property name> <value 2>`

  `<singular property name> <value n>`

- Setting a complex value for a property of a type instance

  `<property> do ... end`

To exemplify the definition of types and properties the next listings explain possible combinations for property types.

Listing 3.2 Single, simple value

```ruby
# Vocabulary definition
service_properties do
  string :service_name
end

# Service description
service_name 'Google Drive for Business'
```

[5]The detailed explication how the instructions are implemented follows in Section 3.1.1.7

Listing 3.2 shows the most basic case: defining a single, simple value property. In the example, Line 3 adds a `service_name` string property to the `Service` type. Line 7 shows how this property can be later set on a `Service` instance to a string value quite easily.

Listing 3.3 Single, complex value

```
1  # Vocabulary definition
2  type :provider do
3    string :provider_name
4  end
5
6  service_properties do
7    provider
8  end
9
10 # Service description
11 provider do
12   provider_name "Amazon.com, Inc."
13 end
```

Listing 3.3 highlights a more complex example. First, Lines 2-4 define a new `Provider` type and add the provider name as a simple string property to this type. Lines 6-8 add a new `provider` property with a type of `Provider` to the `Service` type. In this case, as the type `Provider` has the same name as the property `provider`, the SDL-NG allows shortening `provider :provider` to `provider`.

Listing 3.4 Multiple, simple values

```
1  # Vocabulary definition
2  service_properties do
3    list_of_strings :service_tags
4  end
5
6  # Service description
7  service_tag "great"
8  service_tag "fantastic"
9  service_tag "you'll love it"
```

Listing 3.4 shows the definition of a multi-value property. Line 3 adds the `service_tags` property, a list of strings, to the `Service` type. Then, multiple values can be added as service tags using the singular name of the property, as seen in Lines 7-9. Complex types are set using the same syntax, as seen in Listing 3.5.

Listing 3.5 is the last example, showing the definition of a multi-valued property with a complex type. Lines 2-5 define the `Initiative` type, representing social initiatives of a provider. `Initiative` has two properties: the

Listing 3.5 Multiple, complex values

```
1  # Vocabulary definition
2  initiative do
3    string :name
4    url
5  end
6
7  type :provider do
8    string :provider_name
9
10   list_of_initiatives
11 end
12
13 service_properties do
14   provider
15 end
16
17 # Service description
18 provider do
19   provider_name "Google, Inc."
20
21   initiative do
22     name "Google Digital Unlocked"
23     url "https://learndigital.withgoogle.com/digitalunlocked"
24   end
25
26   initiative do
27     name "Google Internet Saathi"
28     url "https://www.google.com/about/values-in-action/.../"
29   end
30
31   initiative do
32     name "Google for Entrepreneurs"
33     url "https://www.googleforentrepreneurs.com/"
34   end
35 end
```

`name` string property and the `url` URL property. Lines 18-35 define Google as the provider and add some information about Google's social initiatives to the service description.

3.1.1.5 Predefined instances

In contrast to defining type instances directly in the service description (as done in the listings in the preceding chapter), the SDL-NG also allows "predefining" `Type` instances in the vocabulary that can be used in multiple service descriptions. These instances can be easily referred to by their identifier, creating a straightforward enumeration mechanism. Listing 3.6 shows this facility on the example of the cloud service model of a service. The second line of the listing defines the `CloudServiceModel` type. Afterwards, lines 4-6 predefine a set of `CloudServiceModel` instances and associate them with the identifiers `saas`, `paas`, and `iaas`. Line 9 adds the `cloud_service_model` property to the `Service` type. In the service description on line 13, they can be used easily for setting service properties.

Listing 3.6 Predefined instances

```
 1  # Vocabulary definition
 2  type :cloud_service_model
 3
 4  cloud_service_model :saas
 5  cloud_service_model :paas
 6  cloud_service_model :iaas
 7
 8  service_properties do
 9    cloud_service_model
10  end
11
12  # Service description
13  cloud_service_model iaas
```

3.1.1.6 `ServiceCompendium` Class

The `ServiceCompendium` class provides a simple interface to SDL-NG functionality. There are mainly two areas of functionality where the `ServiceCompendium` class provides helpful methods:

Loading and unloading services and vocabulary
The `ServiceCompendium` class provides two utility functions for recursively loading vocabulary files (`*.sdl.rb`) and service descriptions (`*.service.rb`), named `load_vocabulary_from_path` and `load_service_from_path`. When loading, it generates URIs based on the source files and assigns them to the newly defined SDL-NG classes and instances.

"Unloading" is also supported: either using `unload(uri)` to unload specific instances, or using `clear!` to unload any previously loaded file. This unload mechanism is helpful for development and testing purposes. For example, the TRESOR Broker reloads any changed vocabulary items in development, making a restart of the whole Rails application unnecessary. At last, the RSpec tests load and unload services to test the defined behaviour of the framework.

After loading, all `Type` descendants are scoped under the `SDL::Base::Type` class in order to prevent name clashes with standard Ruby classes. If required, the `ServiceCompendium` provides the `register_classes_globally` method that can register all loaded types under `Object`. This allows referencing them in any scope, easing their use in many scenarios.

Retrieving all SDL-NG classes and instances

The `ServiceCompendium` additionally provides access to any SDL-NG class and instance that was loaded. Table 3.1 shows the four methods that can be used for this purpose:

Table 3.1: ServiceCompendium utility methods

Method	Returns
`loaded_items`	All loaded SDL-NG classes and instances
`types`	All `Type` descendants
`type_instances`	All `Type` instance and their identifier as a `Hash`
`services`	All loaded services (`Service` instances)

3.1.1.7 Implementing the Description Lifecycle

After explaining the basic functionality of the framework, the next subsections explain in detail how vocabulary definition, service description, and persistence are implemented in the SDL-NG. First, Figure 3.3 presents an overview about the general life-cycle of SDL-NG service descriptions. As already mentioned, executing vocabulary definition files create descendants of `Type` that are referenced when executing service descriptions to create `Service` instances. These service description can use any Ruby code, for example, to dynamically retrieve information from external systems. Section 3.1.2 provides more information about the employed vocabulary of the use cases. All classes can be exported to a number of formats and persisted to a database.

Implementing the vocabulary definition

This section guides through the implementation of the vocabulary definition. There are two groups of instructions within the files that define the vocabulary:

1. *Definition of a new type*

   As Ruby classes are always open for modification, the same mechanism is also used to extend an existing type, such as the `Service` type.

2. *Definition of a new predefined instance*

   They can be used to implement enumerations for restricting the possible values for a certain property.

Those two options are explicated after providing details about the steps required before:

1. `ServiceCompendium#load_vocabulary_from_path(<path_or_file-name>)`

   This method is the first step towards loading the vocabulary definition. The `<path_or_filename>` specifies either the path containing vocabulary files (`*.sdl.rb`) that should be loaded recursively, or it can specify the path to a single file. The method iterates through them, creating a URL for each item, reading the file contents, and passing any content of these files to the next method.

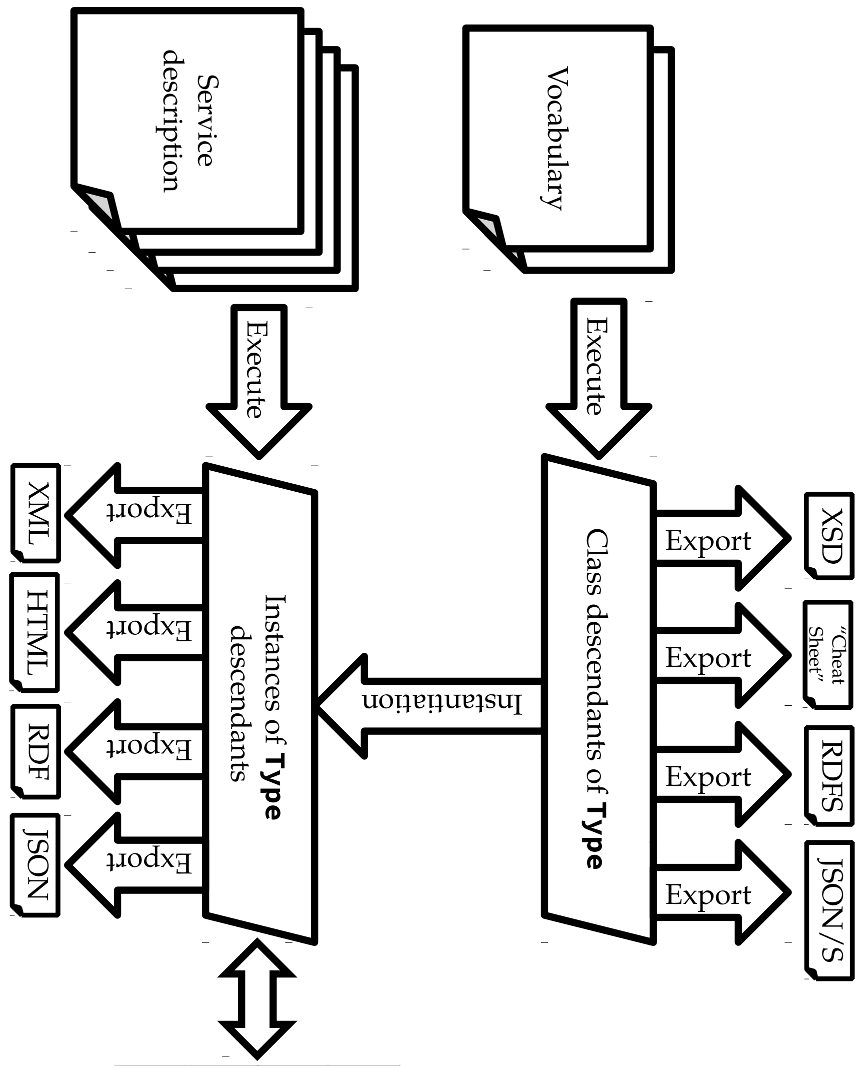


Figure 3.3: SDL-NG Description life-cycle

2. `ServiceCompendium#load_vocabulary_from_string(<definition>, <uri>, <filename>)`

   This method receives file content, the URI, and the filename and passes them to `self.instance_eval`. It also implements a kind of "transaction" mechanism: when the load fails, for example, because of syntax errors, it instructs the `ServiceCompendium` to unload all contents from this URI to prevent errors resulting from only partially loaded files. Afterwards, it raises a RuntimeError and adds the error backtrace to its own backtrace so that SDL-NG users get the specific vocabulary instruction that failed. Otherwise, users would only be instructed that `#load_vocabulary_from_string` failed, instead of the specific part of the definition that was executed at that moment.

3. `self.instance_eval(<definition>, <filename>, 1)`

   `instance_eval` executes the loaded file in the scope of the `ServiceCompendium` instance. Any unqualified instruction, for example, `type :cloud_service_model`, is therefore a call to `ServiceCompendium::type`. The next subsections explain the two possible groups of vocabulary definition instructions.

Option 1: Definition or extension of a new or existing type (for example, `Service`)

1. `ServiceCompendium#type(<symbol>, &type_definition)`

   There are two instructions that are used to define or extend an SDL-NG `Type` descendant: either calling `type(<symbol>)`, for example, `type :cloud_service_model` or using the shorthand `service_properties` for `type(:service)`. Both instructions are calls to the `ServiceCompendium` method `type` which delegates the type definition to the `subtype` method of `SDL::Base::Type`.

2. `SDL::Base::Type.subtype(<symbol>, &type_definition)`

   This method first calls `SDL::Base::Type.define_type(<symbol>, self)` which creates a descendant of the current class. This method can also be called later to define additional subtypes of SDL-NG types, as for example in the definition of the `RestInterface` which is a subtype of the `Interface` type[6].

3. `<type>.instance_eval(&type_definition)`

   If a block (`&type_definition`) was given to the initial `ServiceCompendium#type` invocation, it is evaluated in the context of the created or existing `Type` descendant. This block contains the definitions of the properties that should be added to the created or existing `Type` descendant.

4. `<type>.method_missing(<name>, <args>, &block)`

   The property definitions of the type, for example, `string :service_name` are in fact calls to class methods of the respective `Type` descendant. The

[6]See `examples/vocabulary/1_crf/interop.sdl.rb`

SDL-NG uses the `method_missing` mechanism of Ruby which enables a quite flexible property definition. For example, the method detects a `list_of_<type plural>` invocation which creates a multi-valued property. Also, it is used to implement the shorthand definition explained in Listing 3.3, for example, using `provider` instead of `provider :provider`.

5. `<type>.add_property(<symbol>, <type>, <multi>)`

   After `method_missing` determined the type, name, and multiplicity of the new property, it calls the `add_property` method of the respective `Type`. This creates a new instance of `Property` with the given attributes and adds it to the list of properties of the class.

Option 2: Definition of a new predefined type instance

A new predefined type instance is created by calling a method of `ServiceCompendium` that is named similar to the requested type of the new instance. For example, calling `cloud_service_model :saas` creates a `CloudServiceModel` instance and assign it the identifier `:saas`. This facility does not use the `method_mising` mechanism. Instead, it defines the `cloud_-service_model` method as the last activity of `ServiceCompendium#type`. After completing the other steps of Option 1, `ServiceCompendium#type` calls `regis-ter_sdltype(type)` which defines a method that is named similar to the type using `self.class.send(:define_method, type.local_name.underscore) do ... end`. This newly defined method in turn delegates to `ServiceCom-pendium#create_type_instance(<type>, <identifier>, &block)` which uses a `TypeInstanceReceiver` to create the instance and assigns it the given identifier.

Implementing the service description

The main goal of the SDL-NG service description mechanism is easing the service description in contrast to existing approaches. The conception of "easiness" here is having a syntax that is as free as possible from specifics of its implementation such as "`Class.new(...)`" to create objects to use for property values. In fact, using Ruby as the implementation language leads to a clean syntax itself and allows leaving out semicolons, brackets, and other "semantic noise" from the service description as would be required in other languages (for example, Java). Following this idea, property setting in the SDL-NG should follow a `<property name> <property value>` syntax. As service descriptions are program code, this translates to `self.send(<property name>, <property value>)`, which is an invocation of the method "`<property name>`" on the current scope object (`self`, an instance of `TypeInstanceReceiver`), providing `<property value>` as a method argument.

In detail, a service definition is implemented as follows:

1. `ServiceCompendium#load_service_from_path(<path_or_filename>)`

   This method is used to load a single or multiple service definitions. The `<path_or_filename>` specifies the path containing service files (`*.service.rb`) that should be loaded recursively or a single file. It iterates through the files and extracts the symbolic service name from the filename, for example, `google_drive_for_business` from google_drive_for_business.service.rb. It then reads the file and calls the following method.

2. ServiceCompendium#load_service_from_string(<definition>, <name>, <filename>)

   This method receives the contents of the loaded service definition, the name of the service, and the filename. It should be obvious now that a "service description" is just a predefined instance of the Service class. Therefore, it uses the same mechanism as described in vocabulary Option 2, the definition of a predefined instance, using Service as the Type. In effect, this invokes the ServiceCompendium#create_type_-instance(Service, <name>) to create a new instance of the Service type. However, it hands over the loaded service definition by calling eval(<definition>, <binding>, <filename>, 1) which causes the service description instructions to be invoked in the scope of a new TypeInstanceReceiver instance.

3. SDL::Receivers::TypeInstanceReceiver.new(Service.new)

   In order to implement the property setting syntax, the TypeInstanceReceiver instance needs to provide methods to set any property of a Type instance or its descendants (here: Service). These methods are defined at the time of object construction through iterating all properties of the instance's class. These methods take arguments representing the values to which the respective properties should be set.

4. SDL::Receivers::TypeInstanceReceiver.instance_eval(&block)

   The created TypeInstanceReceiver serves as the scope object (self) for the service description that is passed on through the &block. Calls to, for example, service_name "Google Drive for Business" in fact call those singleton methods created by the TypeInstanceReceiver constructor.

The singleton methods on TypeInstanceReceiver support many different cases of setting properties in order to achieve a high level of intuitiveness. Section 3.1.1.10 provides more information about SDL-NG value types, including how the provided Ruby objects are converted to SDLType instances.

If the property is single-valued, each invocation of <property> overwrites the properties value. If the property is multi-valued, calling the "singularized" name adds a new value to the current list of values. For example, adding to the add_on_repositories list is done through add_on_repository. This "singularization" is implemented by the verbs RubyGem that provides "English verb conjugation for Ruby (and Rails)"[7]. The following paragraphs explain the functionality of the different variants of arguments that can be provided to the singleton methods:

<value> (simple type)

This is the most basic syntax and straightforward: the property is set to the provided value. For example service_name "Google Drive for Business".

[7]https://github.com/rossmeissl/verbs

`<identifier>` (complex type)
Here, the `TypeInstanceReceiver` first resolves the identifier to the corresponding predefined type instance before setting the property, for example, `cloud_service_model saas` resolves `saas` to the predefined instance created previously by `cloud_service_model :saas` and uses this object for setting the property value. "Resolving" is implemented in two steps:

1. `saas` results in trying to invoke an unknown method, therefore calling `method_missing` on `TypeInstanceReceiver`. This returns a new instance of the internal SDL-NG class `InstanceReference` which contains the identifier.
2. This `InstanceReference` object is passed to the singleton method that now can use the property type and the identifier to select the matching predefined instance. In effect, the SDL-NG supports multiple predefined instances with the same identifier differing by their type.

`do <block of instructions> end` (complex type)
This syntax creates a new instance of the property type and uses a new `TypeInstanceReceiver` to evaluate the block to set the properties of the new object, for example, when calling `provider do provider_name "Google, Inc." end`.

`<value 1>, <value 2>, <value n>` (complex type)
Here, a new instance of the property type is created and its properties are set to the list of values in the order of the property definition. When there are only one or two properties, this can improve readability of the service description. As an example, an add-on repository (Type `AddOnRepository`) has two properties: `url` and `number_of_add_ons`. A new add-on repository can be added through `add_on_repository "http://www.example.com", 1000`.

`<name>: <value>, <name>: <value>, ...` (complex type)
This syntax provides a Ruby hash in the form `{<property name> => <property value>}` to the singleton method of `TypeInstanceReceiver` to set the properties of a newly created property type instance to the respective values. This alternative syntax can be used to make it more explicit which properties are set, or to set properties in a different order than they were defined in the vocabulary.

Implementing persistence
The result of the vocabulary definition and the service description are a number of Ruby objects that are bound to the lifetime of the Ruby VM. Interacting with the SDL-NG in such a way provides the best performance, as everything is done within the memory space of the Ruby process. SDL-NG objects are also serializable using standard Ruby persistence options, for example, `Marshal` and `YAML`. However, having services persisted using these methods is inefficient with respect to space, as the serialization formats are quite extensive, and performance, as filtering the list of services requires reading every file.

When the amount of services rise or the lifetime of service descriptions need to be longer than that of the Ruby VM, there needs to be a way of persisting services beyond `Marshal`. As all use cases are derived from the TRESOR broker, which uses MongoDB persistence, it is the only persistence option that supports dumping and loading services. Interfacing with MongoDB is implemented using the Mongoid[8] Framework. From an implementation perspective, certain methods of the SDL-NG are overridden to enable MongoDB persistence, mainly:[9]

- `SDL::Base::Type.class_definition_string`

  This method returns a string that is evaluated to define a `Type` descendant, for example, `Service`. To implement MongoDB persistence, it is overwritten to also include the `Mongoid::Document` class which provides helpful abstractions to work with `Type` instances as MongoDB documents.

- `raw_value` and `raw_value=` in `SDL::Types::SDLSimpleType`

  These methods now use the `mongoize` and `demongoize` to convert certain Ruby objects to MongoDB data types that could otherwise not be saved to the database.

- `SDL::Base::Type#add_property_setters`

  When using memory-based persistence, the values of properties correspond to the values of the instance variables. This method is responsible for adding simple property setters to the respective `Type` descendant to change these instance variables. In the MongoDB persistence, `add_property_setters` instead calls `embeds_many` and `embeds_one`[10] to instruct Mongoid to define complex setter methods that treat property values as embedded subdocuments.

- `SDL::Receivers::TypeInstanceReceiver#refer_or_copy`

  In the case of memory-based persistence, predefined instances are `dup`ed (deeply copied) before setting them as property values. The Mongoid library requires calling `clone` on them instead. This ensures correct copying and reinitialization of Mongoid internal data structures required for correct functioning of the framework.

3.1.1.8 Property Classifications

Property classification denotes the facility of the SDL-NG to define metadata on properties to classify them, for example, according to the criteria of the CRF, for example, “Characteristics”, “Charging”, “Compliance”, “Delivery”, etcetera. Depending on the concrete use case, there are two circumstances where having a property classification is beneficial: first, there could be quite a few properties for a certain `Type` descendant. The `Service` class, for example, has currently more than 30 properties defined, which makes grouping properties by their categories quite helpful for rapid understanding of the property structure and their meanings. Second, the properties could be defined in many different

[8]https://github.com/mongodb/mongoid

[9]For all, see https://github.com/TU-Berlin-SNET/open-service-compendium/tree/master/lib/sdl-ng-overrides/sdl

[10]defined in `Mongoid::Relations::Macros::ClassMethods`

vocabulary files. In this case, knowing where the property definition belongs certainly helps maintaining lucidity in the vocabulary.

Multiple classification schemes are conceivable, for example, partitioning into "essential" and "optional" properties, or "tagging" properties with arbitrary values. Currently, the SDL-NG implements the "category" scheme, implemented in `SDL::Base::PropertyClassification::PropertyCategory`. This class uses the `loaded_from` attribute to derive a category identifier from the relative path of the file where the property was defined[11]. For example, the properties defined in `1_crf/characteristics.sdl.rb` are assigned the `crf.characteristics` property.

Some SDL-NG methods were implemented to help using property categories. First of all, the category of a property can be retrieved by `SDL::Base::Property#category`. Second, the `key_category_map` class method of `PropertyCategory` provides a hash of all known keys to their `PropertyCategory` object. At last, the returned `PropertyCategory` object provides the `properties` method that returns all properties from the same category.

3.1.1.9 Generating URIs

Assigning URIs to conceptual things has been shown to be a useful approach to enable data consumers to query and interact with information systems. This is especially highlighted in the Linked Data Principles, set by Berners-Lee [23]. There are many different parts of the SDL-NG requiring URIs, for example, the identification of nodes within the RDF output. Furthermore, the way these URIs are generated differs between the "standalone" SDL-NG library and the information systems based on it. Therefore the `SDL::Base::URIMappedResource` adds a `uri` method to the classes that include this module. The specific strategy for deriving a URI for SDL-NG objects and classes is implemented using a URI mapping object which responds to the method `uri(<object>)` returning the URI. Currently, there are two URI mappers:

1. `DefaultURIMapper` is used in the SDL-NG itself, for example, to generate the URLs in the exemplary outputs of `process_service_descriptions`. It appends to the static URL `http://www.open-service-compendium.org` the identifiers for the objects and classes, for example, `/types/Service` for the `Service` Type and `/services/google_drive_for_business` for the respective service.

2. `OSBURIMapper` is used by all cloud service registry implementations. It has two additional features: first, it uses the value of the HTTP `Host` header to create the first part of the URL. Second, it includes the specific version of a service description in the URL.

3.1.1.10 SDL-NG Value-Types

The main goal of the SDL-NG value type system is providing an extensible mechanism to use Ruby objects to represent property values in the service descriptions. SDL-NG value types allows the authoring of service descriptions

[11] `filename.gsub(%r[#{path_or_filename}|.sdl.rb|\d_], '')[1..-1]`

without knowing specifically how, for example, monetary values are represented in Ruby. This is achieved by offering wrappers around Ruby classes that define conversion methods from simple literals to wrapped types, for example, from `"$ 1"` to a new instance of the `Money` class. To explain this mechanism, Listing 3.7 shows the SDL-NG value type `SDLMoney` which wraps the `Money` class provided by the `money` Rubygem.

Listing 3.7 Example SDL-NG value type 'SDLMoney'

```
 1  require 'money'
 2  require 'monetize'
 3
 4  class SDL::Types::SDLMoney < SDL::Types::SDLSimpleType
 5    include SDL::Types::SDLType
 6
 7    wraps Money
 8    codes :money
 9
10    def from_string(string_value)
11      begin
12        @value = Monetize.parse(string_value)
13      rescue ArgumentError
14        throw "Invalid Money value: #{string_value}"
15      end
16    end
17  end
```

Line 1 imports the `money` Rubygem, a "Ruby library for dealing with money and currency conversion"[12] which is used to implement all money-related functions of the SDL-NG. The following Line 2 references the `monetize` gem, a "library for converting various objects into `Money` objects"[13]. It is used for parsing the strings provided in the service descriptions and converting them to `Money` objects.

The subsequent class definition in Lines 4-17 shows how SDL-NG value types are created. First of all, any value type class inherits from `SDLSimpleType` which implements functionality shared between all instances. It mainly declares the two class attributes `value`, holding the wrapped type instance, and `raw_value`, holding the value given in the service description. The constructor of any `SDLSimpleType` descendant invokes a *conversion method* which transforms the provided value in the service description to the wrapped Ruby type. The constructor derives the name of this method from the name of the provided object class, for example, it calls `from_string` for `String` instances and `from_nokogiri_xml_element` for scraped HTML data (`NokogiriXmlElement`). In the example, Lines 10-16 contain such conversion method that takes a string and uses the `monetize` gem to parse and convert it. An `Error` is raised when this conversion fails.

[12]https://github.com/RubyMoney/money
[13]https://github.com/RubyMoney/monetize

3.1.1.11 SDL-NG Value Type Wrappers

Another part of the SDL-NG value type system is the `SDLType` module which provides two class methods. The first, `wraps`, declares which specific Ruby class is wrapped. In the example (Line 7) it is the `Money` class provided by the `money` gem. The other, `codes`, specifies the identifiers that can be used in the vocabulary definition to reference this value type. In the example (Line 8), the wrapper is registered under the `money` identifier. In effect, the vocabulary can now contain the instruction `money :last_years_revenue`, allowing easy to read statements in the service description, such as `last_years_revenue "1M €"` which correctly creates a `Money` instance representing the amount of one million Euros. The following Table 3.2 presents an overview about the existing SDL-NG value types, which Ruby class they wrap, their codes, and conversion methods (if any).

Table 3.2: SDL-NG value types

SDL Type	Wraps	Codes	Conv. Meth.
`Boolean`	`Object`[14]	`bool`, `boolean`	None
`Description`	`String`	`description`	`from_nokogiri_xml_-element`[15]
`Duration`	`Duration`[16]	`duration`	None
`Money`	`Money`	`money`	`from_string`
`Number`	`Numeric`	`number`, `int`, `integer`	None
`String`	`String`	`string`, `str`	`from_symbol`
`Time`	`Time`	`time`	None
`Url`	`URI`	`uri`, `url`	`from_string`[17]

3.1.1.12 Exporting Data

To allow other information systems to consume the vocabulary and descriptions, the SDL-NG can export service descriptions as well as the vocabulary in a multitude of formats: XML and XSD, JSON and JSON Schema, and RDF. To explain service describers how to use the SDL-NG, the registry implementations in the use cases also provide a complementary HTML "cheat sheet" which is shown in Section 3.1.4.

The implementation of the exporters is based on two base classes: `ServiceExporter` and `SchemaExporter`. They provide the self-explanatory methods `export_service_to_file(<service>, <path>)` and `export_-schema_to_file(<path>)` respectively. For each format there are independent implementations:

- Exporting to XML and XSD

[14]There is no `Boolean` class in Ruby, only `TrueClass` and `FalseClass`.

[15]Used to persist parts of HTML documents as service descriptions

[16]Part of the `active_support` gem. Contains shorthand expressions such as `3.minutes` and `2.hours`.

[17]Uses `URI.parse` to parse the String, ensuring its validity

Due to the comprehensive nature of XML and the XML Schema Definition Language [164], the combination of XML/XSD preserves most of the semantics of the service descriptions. Furthermore, the use of XSD enumerations and restrictions provides a mechanism to convey to a service description consumer which predefined instances can be expected in the XML output. However, as some XML libraries, such as Microsoft .NET used by the TRESOR Marketplace, cannot understand this fully valid construct, a new exporter, `XSDSimpleSchemaExporter`, was implemented besides the `XSDSchemaExporter` to output a simpler version of the schema. Both the schema exporters as well as the XML service exporter (`XMLServiceExporter`) make use of the `xml_mapping.rb` file. This file contains extension methods for the SDL-NG classes that derive XML/XSD specific values, for example, corresponding XSD types for SDL-NG value types, element identifiers, and values for XML nodes.

- Exporting to JSON and JSON Schema

  The implementation for these exporters follows roughly the same structure as the XML/XSD exporters. The `JSONExporter` and `JSONSchemaExporter` provide the exporter implementation while the `json_mapping.rb` contains SDL-NG extensions to derive corresponding JSON Schema types and JSON identifiers. As the functional scope of JSON Schema is more limited than XSD, the SDL-NG adds two additional fields to the schema, conveying the properties' category (`category`) as well as the set of predefined instance identifiers (`enum`).

- Exporting to RDF

  Most of the functions required for the RDF export are provided by the RDF.rb library[18]. It provides support for many common RDF serializations, such as RDF-XML, RDFa, and Turtle. As with the other formats, the `RDFExporter` class generates RDF service descriptions while the `rdf_mapping.rb` generates the RDF literal values for SDL-NG values and the type and property URLs. Generating an RDF graph using RDF.rb is quite straightforward, as the source for this graph is a simple Ruby array, containing RDF `[<subject>, <predicate>, <object>]` triples. There is no schema exporter yet, for example, to create RDF-Schema or OWL files. Section 3.1.1.15 explains how the SDL-NG can map complex types on existing RDF vocabularies.

3.1.1.13 SDL-NG Multi-language Self-documentation

The `util/documentation.rb` file provides facilities to manage descriptions for types, properties, property categories, and instances. These are used, for example, by the Open Service Compendium to generate a comprehensible service description for the potential service consumers. The required underlying functions to manage multi-language documentation are provided by the `i18n` gem, the same used by Ruby on Rails. From an implementation point of view, loading this file extends each of the documentable objects and classes with two methods: the `documentation_key` method generates an identifier to retrieve the

[18]http://ruby-rdf.github.com/

specific documentation, for example, `sdl.property.type.service.cloud_service_model` for the property `cloud_service_model` of the `Service` SDL-NG type. The other added method, `documentation`, provides this key to the `I18n` module to retrieve the respective documentation in the current language. The `i18n` gem supports different repositories, mainly databases, JSON, and the YAML format used by the SDL-NG to persist the documentation. The repositories are chained so that multiple files are sourced for the documentation. This allows each vocabulary to define its documentation separately from the documentation of the SDL-NG itself.

In effect, using the self-documentation features creates a clear correlation between SDL-NG objects and a specific documentation, preventing redundant information and ensuring consistency. Furthermore, the implementation of the Open Service Compendium is simpler, as the user-facing service output can be implemented in a generic way, relying on the self-documentation of the SDL-NG classes and objects instead of requiring modification of the view templates for each vocabulary.

3.1.1.14 HTML Parsing

One of the benefits of a dynamic DSL is the possibility to include arbitrary commands in the service descriptions, for example, retrieval of data from external sources. As cloud services are most often described using content on websites, HTML parsing presented the most obvious functionality to implement first in the SDL-NG. The file `util/nokogiri.rb` provides the `fetch_from_url(<url>, *<search>)` method which retrieves the website accessible via `<url>`, applies one or more CSS or XPath selectors contained in `<search>`, and returns a set of matching document nodes. Two additional processing steps were added: first, all relative URLs are converted to absolute URLs in order to preserve hyperlink targets. Second, any `a` element gains the `target` attribute with the assigned `_new` value. This causes the browser to not leave the current website and instead open the link in a new window.

3.1.1.15 RDF Mapping

As mentioned before, the SDL-NG supports exporting services as RDF in order to support different data consumers. Before implementing the Open Cloud Computing Map (OCCM), it did not refer to any existing schema and instead used an automatically generated, registry-internal schema. However, this prevents easy interlinking with other data repositories as the semantics of the dataset are not clearly defined. To alleviate this issue and enable interlinking, for example, between the OCCM and the Open Service Compendium, the SDL-NG was extended with mappings between established schemas, for example, the Schema.org RDF schemas, and the internal SDL-NG types and attributes. Specifically for the OCCM, it maps location-related types and properties onto the GeoNames schema[19].

The SDL-NG type definition as well as its mapping to an RDF schema is done using very concise code, as shown in the example listings in this section: Listing Listing 3.8 exemplifies the definition of the SDL-NG type `Location`. This type uses strings to represent common location attributes, such as street address

[19]http://www.geonames.org/

and postoffice number. Listing Listing 3.9 shows the mapping of this type to the Schema.org RDF type `PostalAddress`[20], for example, from the `po_number` SDL-NG property to the `postOfficeBoxNumber` Schema.org property.

Listing 3.8 SDL Location Type Definition

```
type :location do
    string :name

    string :country_code
    string :region
    string :locality
    string :po_number
    string :postal_code
    string :street_address
end
```

Listing 3.9 RDF mapping SDL Location to Schema.org

```
require 'rdf/vocab/schema'

class SDL::Base::Type::Location
    require 'sdl/exporters/rdf_mapping'

    map_rdf_type(RDF::SCHEMA.PostalAddress)

    map_rdf_property('name', RDF::SCHEMA.name)
    map_rdf_property('country_code', RDF::SCHEMA.addressCountry)
    map_rdf_property('region', RDF::SCHEMA.addressRegion)
    map_rdf_property('locality', RDF::SCHEMA.addressLocality)
    map_rdf_property('po_number', RDF::SCHEMA.postOfficeBoxNumber)
    map_rdf_property('postal_code', RDF::SCHEMA.postalCode)
    map_rdf_property('street_address', RDF::SCHEMA.streetAddress)
end
```

The SDL-NG employs both the RDF.rb library[21] as well as an accompanying set of RDF vocabularies[22] to ease the implementation of the RDF mapping. There are two central instructions that need to be invoked: First, `map_rdf_type` (Listing Listing 3.9) defines the corresponding RDF type for the SDL-NG type, here `PostalAddress` (Listing Listing 3.8). Second, `map_rdf_property` (Listing Listing 3.9) defines the mappings from the SDL-NG properties (Listing Listing 3.8) to the Geonames RDF properties (Listing Listing 3.9). Figure 3.4 illustrates the resulting Open Cloud Computing Map which uses RDF data

[20]https://schema.org/PostalAddress
[21]https://github.com/ruby-rdf/rdf
[22]https://github.com/ruby-rdf/rdf-vocab

from the service registry to display a map, presenting geospatial data about cloud services, for example, data centers and cloud provider's subsidiaries.

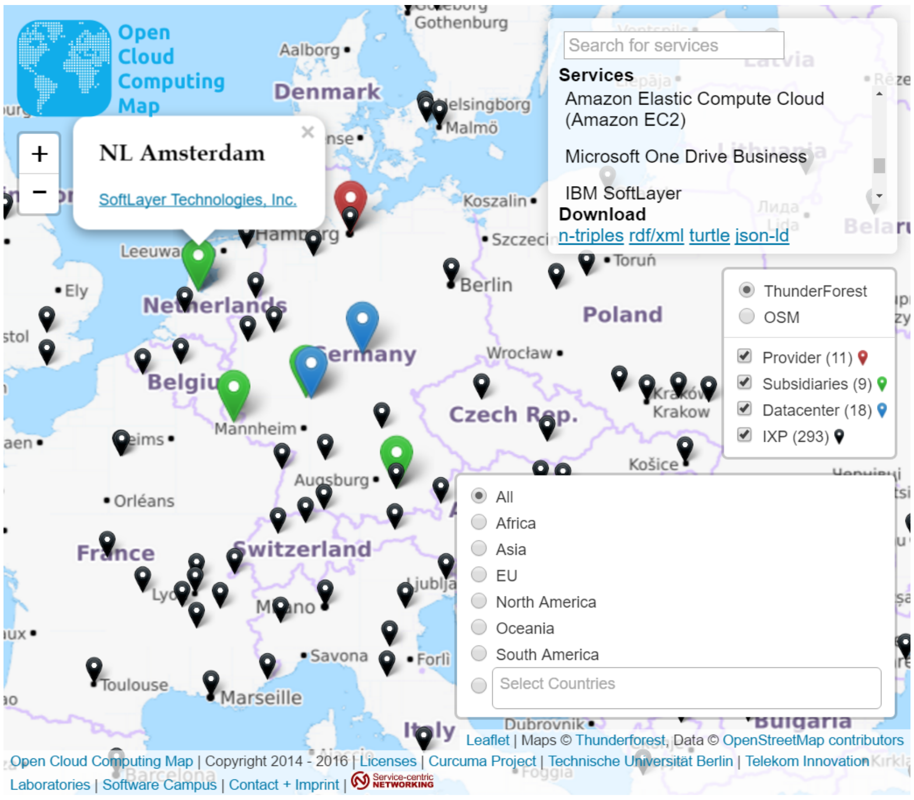

Open Cloud Computing Map
NL Amsterdam
SoftLayer Technologies, Inc.
Search for services
Services
Amazon Elastic Compute Cloud (Amazon EC2)
Microsoft One Drive Business
IBM SoftLayer
Download
n-triples rdf/xml turtle json-ld
ThunderForest
OSM
Provider (11)
Subsidiaries (9)
Datacenter (18)
IXP (293)
All
Africa
Asia
EU
North America
Oceania
South America
Select Countries
Leaflet | Maps © Thunderforest, Data © OpenStreetMap contributors
Open Cloud Computing Map | Copyright 2014 - 2016 | Licenses | Curcuma Project | Technische Universität Berlin | Telekom Innovation Laboratories | Software Campus | Contact + Imprint |

Figure 3.4: Open Cloud Computing Map

3.1.2 Business Vocabularies

The business vocabularies define the SDL-NG classes, properties, and value types that are used to describe services. There is a generic vocabulary representing common Cloud Requirement Framework (CRF) criteria as well as specific vocabularies for each of the implemented use cases, for example, the cloud storage vocabulary implemented for Use Case 2. The generic business vocabulary consists of elements that represent evaluation criteria from the CRF: 37 classes (e.g., `CloudServiceModel`, `PaymentMethod`, and `Browser`), 31 Service properties (e.g., `cloud_service_model`, `payment_methods`, and `compatible_browsers`), and 52 instances (e.g., `saas`, `credit_card`, and `firefox`). Conditions for including elements in the vocabularies are their relevance to some or all of the use cases, a meaningful and simple formal modeling, and enough information to use them.

To provide a short overview of the vocabulary, Table Table 3.3 presents the CRF requirement area and exemplary properties, for example the "billing and payment options" property in the "delivery" area. The table leaves out implementation details for brevity reasons, for example, the property types, their multiplicity, and acceptable enumeration values.

For easier maintenance, the source files of the vocabulary are structured according to the CRF hierarchy[23]. For example, to represent the first CRF criteria *cloud service model*, the vocabulary defines the `CloudServiceModel` SDL-NG type and includes four predefined instances `saas`, `paas`, `iaas`, and `haas` (Hardware-as-a-Service). As they belong to the CRF abstract requirement *characteristics*, these definitions are contained in the file `1_crf/characteris-tics.sdl.rb`.

Table 3.3: Business Vocabulary

CRF Requirement	CRF Criteria in Business Vocabulary
Characteristics	Cloud service model, service categories
Charging	Charge unit (user account, floating license)
Compliance	Data location, audit options (e.g. audit logging)
Delivery	Billing and payment options
Dynamics	Duration for provisioning an end user
Interop	Features, interfaces, and compatible browsers
Optimizing	Maintenance windows and future roadmaps
Portability	Exportable and importable data formats
Protection	Communication protection (HTTPS, VPN)
Provider mgmt.	Support availability
Reliability	Offline capabilities
Reputation	Year of service establishment
Trust	Financial statement, reference customers

The Cloud Storage Broker (Use Case 2) uses a vocabulary that reflects the defining characteristics of storage offerings in order to describe services within this market. Figure 3.5 provides a highly visual representation of the storage vocabulary to provide an overview about the covered service aspects. The details of this vocabulary can be found in [87].

3.1.3 Rails Backend

The functionality of the service registry is offered through a RESTful API, implemented using Ruby on Rails. Fundamentally, it allows to manage service descriptions through create, read, update, and delete (CRUD) operations. Creating and updating services is done by receiving SDL-NG documents from clients and putting them in a Redis Job Queue for eventual evaluation. The services persisted in the MongoDB database can be represented as HTML, XML, JSON, and RDF. The backend also allows retrieving the schema of these documents as XML Schema Definition (XSD) and JSON Schema.

While all other use cases are based on the described backend, additional Rails controllers were created to implement the TRESOR Service Broker (Use Case 1). These controllers provide services for the TRESOR marketplace and the medical PaaS platform. The TRESOR marketplace uses Service Broker facilities

[23]https://github.com/TU-Berlin-SNET/sdl-ng/tree/master/examples/vocabulary

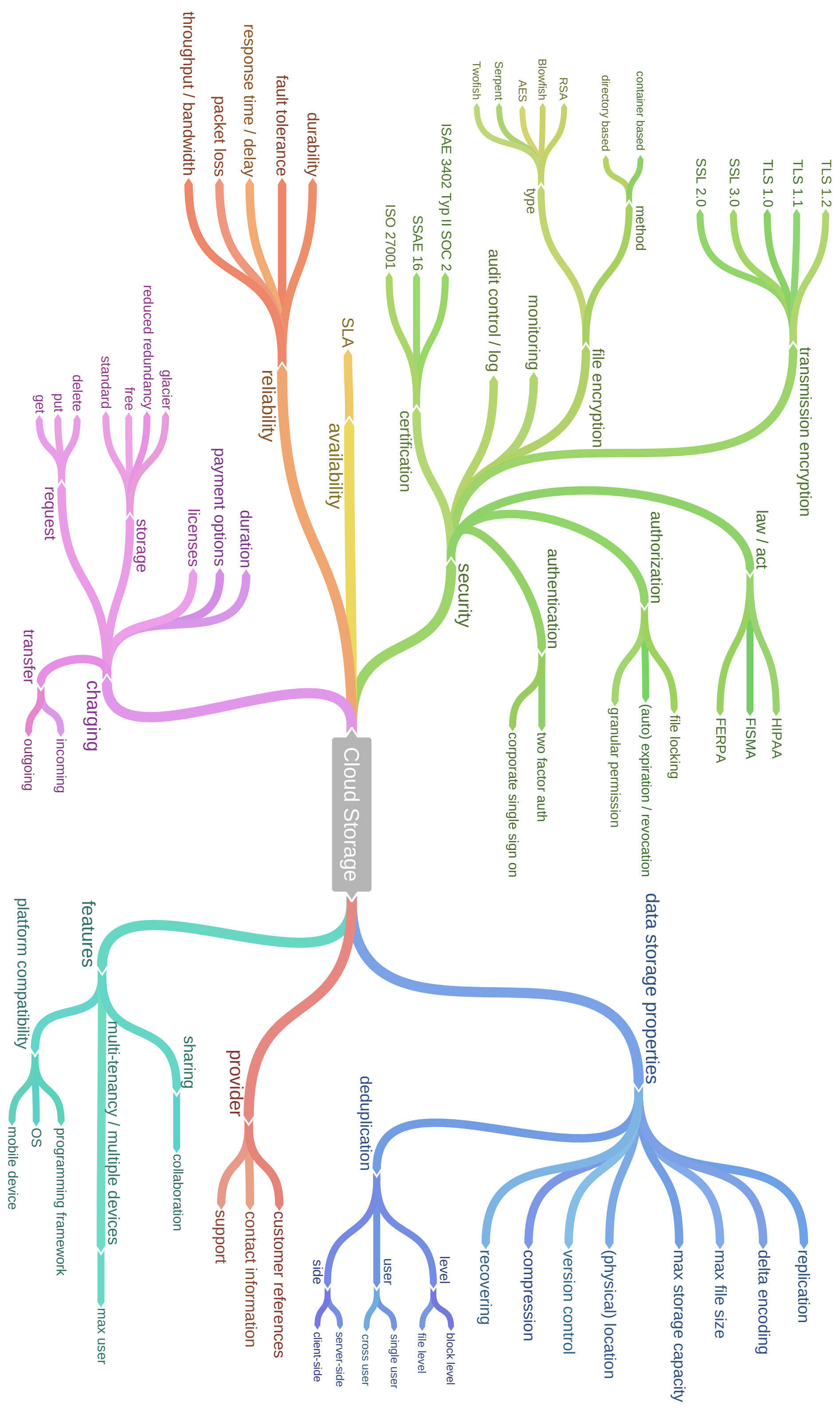


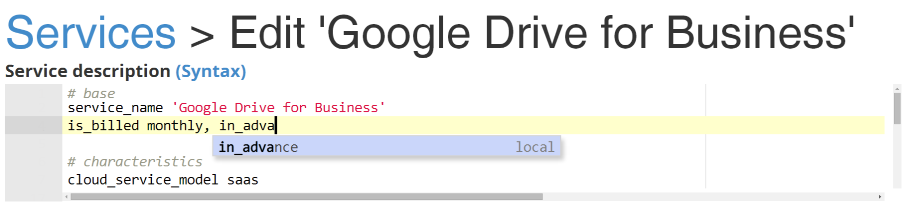
Figure 3.5: Cloud Storage vocabulary (Figure taken from [87, p. 17])

to manage information about cloud service users and providers. Furthermore, limited booking and cancellation features to be used by business users were implemented. This booking also includes instructing the PaaS platform to deploy a new instance of the respective service for those users. This functionality relies on the deployment APIs of the RedHat OpenShift PaaS platform[24]. The resulting TRESOR Service Broker implementation is available on GitHub[25]. However, it was not updated yet to reflect recent changes in the public APIs of OpenShift. More information about the integration between the TRESOR Broker and Marketplace can be found in Section 4.4.3.

The concept, requirements, architecture, and APIs of the CYCLONE IaaS Registry (Use Case 3) are defined in detail within a public project deliverable [99]. In summary, an extended version of the Rails backend serves as an additional data source for the Nuv.la Application Deployment Platform, which uses the information contained in the registry to propose alternative cloud platforms for the eventual deployment of complex applications.

3.1.4 Client

The architecture anticipates that clients consume the RESTful HTTP API of the Rails backend to invoke service registry functions. The use cases feature a variety of different clients: Within the TRESOR Service Broker (Use Case 1), the Rails ActionView view layer serves pages containing regular HTML forms to interact with the registry. There is also a service editor (see Figure 3.6) and a "cheat sheet" showing the available description elements (see Figure 3.7) to ease the management of service descriptions.



Services > Edit 'Google Drive for Business'

Service description (Syntax)

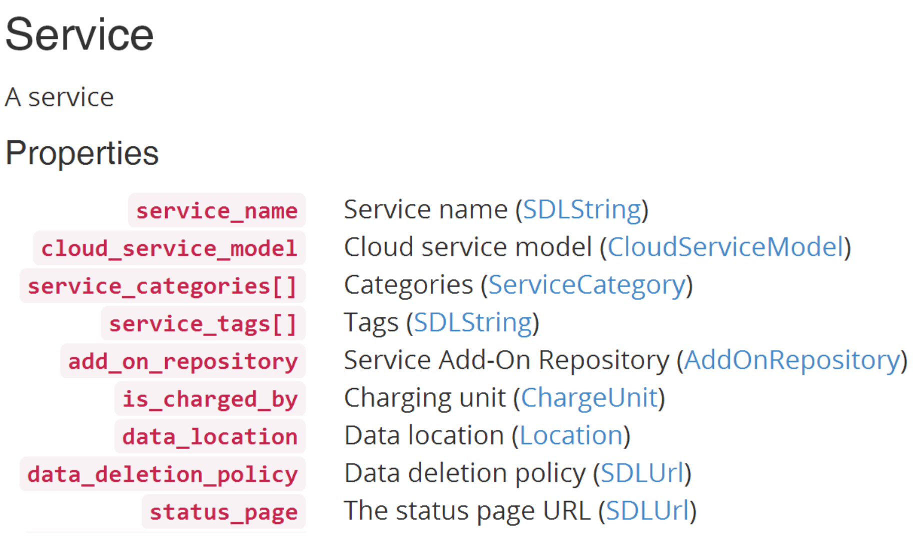
```
# base
service_name 'Google Drive for Business'
is_billed monthly, in_adva
                   in_advance                  local
# characteristics
cloud_service_model saas
```

Figure 3.6: Service editor

The Cloud Storage Broker (Use Case 2) extends the TRESOR Service Broker (Use Case 1) backend with two new Rails Controller/View modules. First, a comparison module (see Figure 3.8) that allows potential cloud customers to view two services side-by-side to compare the property values of both services. Second, a faceted search module (see Figure 3.9) that allows a fine-grained exploration of the services contained in the service registry.

The view layer of the Rails backend was replaced to realize the OSC (Use Case 5). More concretely, the Ruby ERB templates were substituted with a Single Page JavaScript Application based on AngularJS. Especially for browsing and comparing services this provides a much swifter user experience, as the browser can render the interface itself, often without a roundtrip to the server. The sources of the OSC can be found on GitHub[26].

[24]https://www.openshift.com/
[25]https://github.com/TU-Berlin-SNET/tresor-broker
[26]https://github.com/TU-Berlin-SNET/open-service-compendium



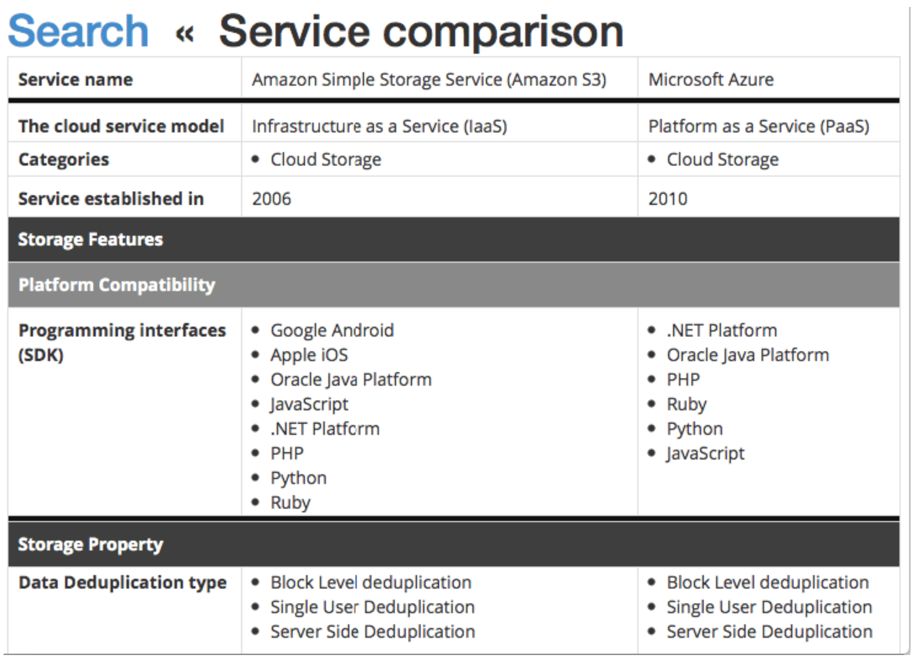
Service

A service

Properties

`service_name`	Service name (SDLString)
`cloud_service_model`	Cloud service model (CloudServiceModel)
`service_categories[]`	Categories (ServiceCategory)
`service_tags[]`	Tags (SDLString)
`add_on_repository`	Service Add-On Repository (AddOnRepository)
`is_charged_by`	Charging unit (ChargeUnit)
`data_location`	Data location (Location)
`data_deletion_policy`	Data deletion policy (SDLUrl)
`status_page`	The status page URL (SDLUrl)

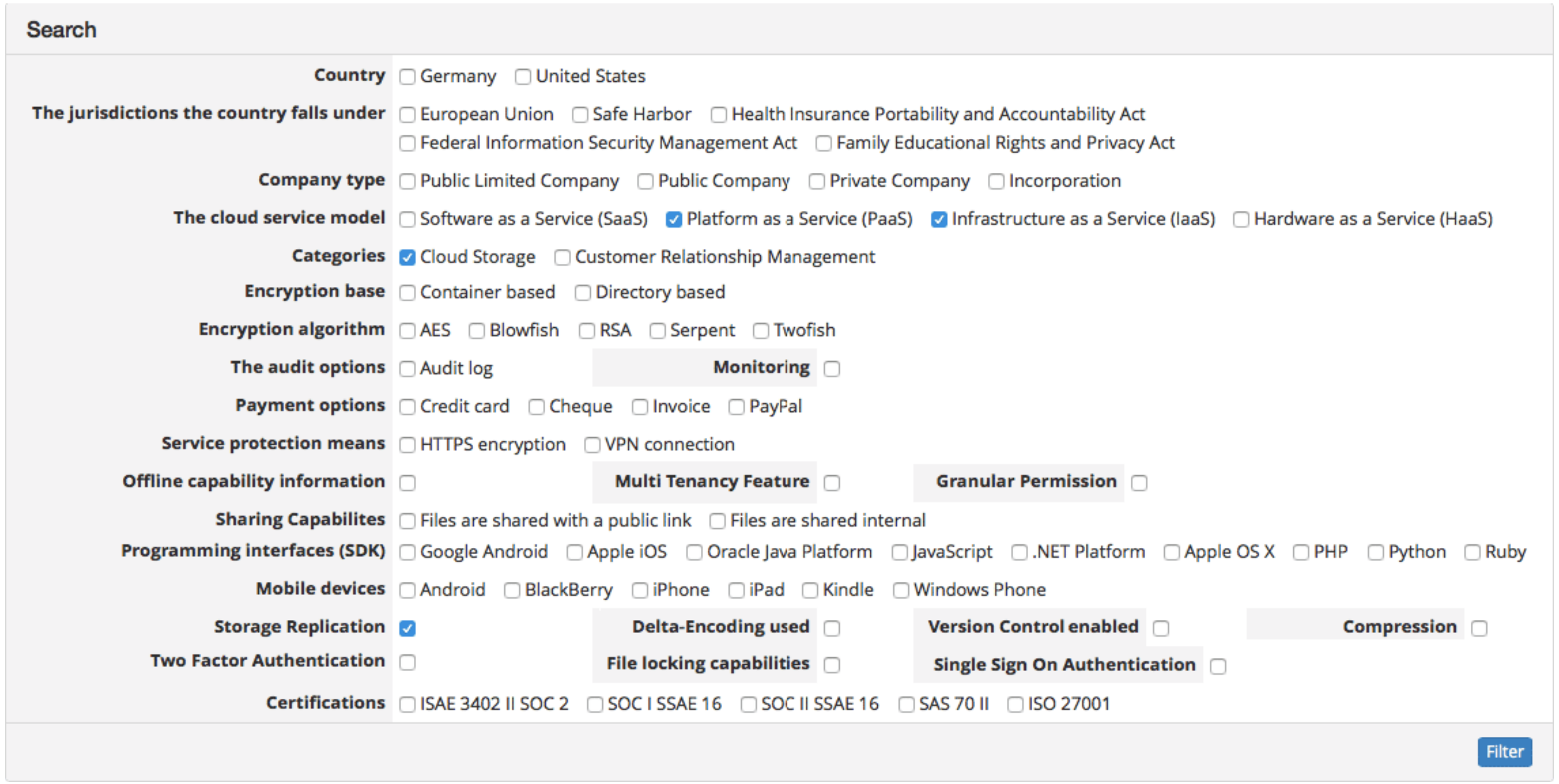
Figure 3.7: Cheat sheet excerpt



Search « Service comparison

Service name	Amazon Simple Storage Service (Amazon S3)	Microsoft Azure
The cloud service model	Infrastructure as a Service (IaaS)	Platform as a Service (PaaS)
Categories	• Cloud Storage	• Cloud Storage
Service established in	2006	2010
Storage Features		
Platform Compatibility		
Programming interfaces (SDK)	• Google Android • Apple iOS • Oracle Java Platform • JavaScript • .NET Platform • PHP • Python • Ruby	• .NET Platform • Oracle Java Platform • PHP • Ruby • Python • JavaScript
Storage Property		
Data Deduplication type	• Block Level deduplication • Single User Deduplication • Server Side Deduplication	• Block Level deduplication • Single User Deduplication • Server Side Deduplication

Figure 3.8: Comparison module



Search

Country	Germany · United States
The jurisdictions the country falls under	European Union · Safe Harbor · Health Insurance Portability and Accountability Act · Federal Information Security Management Act · Family Educational Rights and Privacy Act
Company type	Public Limited Company · Public Company · Private Company · Incorporation
The cloud service model	Software as a Service (SaaS) · [✓ Platform as a Service (PaaS)] · [✓ Infrastructure as a Service (IaaS)] · Hardware as a Service (HaaS)
Categories	[✓ Cloud Storage] · Customer Relationship Management
Encryption base	Container based · Directory based
Encryption algorithm	AES · Blowfish · RSA · Serpent · Twofish
The audit options	Audit log · **Monitoring**
Payment options	Credit card · Cheque · Invoice · PayPal
Service protection means	HTTPS encryption · VPN connection
Offline capability information	· **Multi Tenancy Feature** · **Granular Permission**
Sharing Capabilites	Files are shared with a public link · Files are shared internal
Programming interfaces (SDK)	Google Android · Apple iOS · Oracle Java Platform · JavaScript · .NET Platform · Apple OS X · PHP · Python · Ruby
Mobile devices	Android · BlackBerry · iPhone · iPad · Kindle · Windows Phone
Storage Replication	[✓] · **Delta-Encoding used** · **Version Control enabled** · **Compression**
Two Factor Authentication	· **File locking capabilities** · **Single Sign On Authentication**
Certifications	ISAE 3402 II SOC 2 · SOC I SSAE 16 · SOC II SSAE 16 · SAS 70 II · ISO 27001

Figure 3.9: Faceted search module

3.1.5 Service Evaluator

As SDL-NG service descriptions are program code, they should be invoked in a secure environment when they originate from untrustworthy sources. This is the case with the Open Service Compendium (Use Case 5), as it receives descriptions from any user on the Internet. Such a secure environment can be created using containers or virtual machines that are set up with read-only file systems and restricted host resource access. The SDL-NG as well as the Business Vocabularies should be bundled to allow standalone service evaluation in these separated environments. After SDL-NG code is executed securely, the resulting service descriptions are persisted in the MongoDB.

As all use cases were deployed in a trusted setting with selected users on disposable VMs, the service evaluator was not yet deployed within an additionally secured execution environment. For simplification, service evaluation is currently carried out within the Rails backend before persisting it in the MongoDB.

3.1.6 Redis Job Queue

The goal of the Redis Job Queue is to decouple the potentially long-running secure service execution from the other functionality of the Rails Backend. Whenever a service description needs to be evaluated, the Rails Backend pushes an evaluation job to the queue. A service evaluator regularly polls this queue and invokes a worker process to handle the job. The queue is also used for other job types, for example, to implement the TRESOR Broker service booking.

3.1.7 Constraint-based Matchmaker

The implementation of the constraint-based matchmaker was fully carried out by Zilci. We discussed how it can be reasonably integrated with the components that were created for this thesis. The following explanation was created by Zilci and paraphrased here in order to provide a better understanding of the relationship between our components. More details can be found in [195] as well as the GitHub sources[27].

The matchmaker communicates with the Rails Backend via inter-process communication (for example, a pipe) and provides clients constraint-based matchmaking facilities through the RESTful API. It uses the Java Constraint Programming API Standard JSR-331 [50] with Choco Solver 2 [115] to implement two constraint models for service matching. The first model realizes discrete value matching with hard constraints, interval matching for negative and positive tendencies, as well as feature list matching. The second model implements discrete value matching with soft constraints. Table 3.4 shows an example service matchmaking problem. Here, a service request posts constraints on the property values of three exemplary services. In order to create a satisfactory matchmaking result, both constraint models have to be applied, as there is a high diversity in the types of values and constraints.

[27] https://github.com/TU-Berlin-SNET/cloud-service-matcher

Table 3.4: An Example Service Matchmaking Problem

QoS Property	Service 1, Provider A	Service 2, Provider B	Service 3, Provider C	Service Request
Version	5.5	5.6	5.6	$= 5.6?$
Response time	$< 120ms$	$< 200ms$	$< 400ms$	$< 300ms$
Storage in Free Version	0GB	15GB	20GB	$> 5GB$
Availability	$> 99.99\%$	$> 99.95\%$	$> 99.95\%$	$> 99\%$
Establishment Year	2010	2005	2012	> 2009
Pricing	per dyno-hour	per number of requests	per hour	per hour
Compatible Browsers	Explorer, Chrome, Firefox	Explorer, Chrome, Firefox, Safari	Explorer, Safari	Explorer, Firefox, Safari

3.1.8 Database

Service records are persisted in a MongoDB database as this allows flexible evolution of the underlying schema. Each service evaluator invocation result is saved together with a timestamp, so that historic descriptions can be retrieved. Future analytical use cases can be envisioned based on this feature, for example, analyzing trends in cloud pricing over time. The resulting data could augment related research on cloud pricing, as described in the work of Rohitratana and Altmann [142] who analyze the impact of pricing schemes on a market for SaaS.

3.1.9 Meeting the Stakeholder Requirements

The following subsections iterate each requirement and discuss how the resulting service registry architecture meets them.

Business pertinence (Requirement 1)

Business pertinence is ensured by incorporating relevant empiric research on users' cloud service selection criteria, for example, by modelling the business vocabularies in accordance with the Cloud Requirement Framework (CRF) [136]. The authors of the CRF use well-grounded empirical research to compile a set of relevant service selection criteria for comparable stakeholders. Besides conceptual considerations, there have been regular checks if the resulting vocabularies can describe existing cloud services that are pertinent to the stakeholders of the use cases. This denotes going through the list of service properties and evaluate if and how they would be applicable to prominent cloud services.

Tooling simplicity and adaptability (Requirement 2)

To realize this requirement, the design strives to apply two constraints commonly associated with extreme programming: DTSTTCPW and YAGNI.[28]

[28]Beck's seminal book on extreme programming [16] provides a comprehensive summary.

DTSTTCPW stands for "Do the simplest thing that could possibly work" and reflects the best practice of only implementing the actual needed functionality, instead of creating over-engineered solution architectures trying to fit use cases not yet realized. YAGNI is the abbreviation for "you aren't gonna need it", summarizing the empiric knowledge that many people not following DTSTTCPW spend time on functionality that they only foresee, yet at the end oftentimes never need or require in another form than they initially thought.

In order to create simple tooling, the architecture employs an internal Ruby DSL for service description and vocabulary definition as the Ruby language is especially well suited for DSLs. It is very terse and the resulting service descriptions resemble ordinary text files. In comparison to RDF, JSON, and XML, such a simple text format should better support regular Internet users in authoring descriptions and registry operators in extending the vocabulary. Furthermore, a concise DSL keeps the intricacies of the internal data representation and the type system away from the users. For example, users can use literal representations, such as "`is_billed monthly`" or "`last_years_revenue '6224000000 $'`" instead of learning about a type system and how to correctly instantiate typed values. Furthermore, the use of a DSL for the vocabulary eases its gradual evolution with each new use case. It is very hard to further assess tooling simplicity objectively. One possible measure is the size of the library. At around 1.500 lines of code[29], the current size of the SDL-NG library can be regarded quite small in relation to its feature set. Following YAGNI, there is no code that is not used within any use case implementation.

To make the architecture applicable within all use cases, a RESTful API is provided that offers the complete registry functionality to any external consumers. The API supports multiple media-types (HTML, XML, JSON, RDF, and SDL-NG) as there are diverse data consumers requiring different service representations. For example, while the Open Service Compendium (Use Case 5) lets users interact directly with the service registry through a web interface, the external TRESOR Marketplace consumes the RESTful TRESOR Service Broker API to manage services using the SDL-NG and retrieve them via XML for further processing.

Versatile data retrieval (Requirement 3)

As already pointed out, an internal DSL can incorporate both static information and scraping logic, creating a versatile tool for externally retrieving service information. The effort for implementing this logic is reduced by the wealth of stable and mature Ruby libraries for needed functionality, such as HTML parsing and querying external databases and semantic datastores. The example descriptions[30] use scrapers to extract the textual description of SaaS features from the websites of Google and Salesforce and combine them with other static service knowledge in a single file.

Modeling capabilities (Requirement 4)

As plain Ruby classes and objects represent the vocabulary and services, extending the modeling capabilities of the description language is quite straightforward. This is demonstrated by implementing a simple feature model in the

[29]Counted by CLOC (https://github.com/AlDanial/cloc)

[30]https://github.com/TU-Berlin-SNET/sdl-ng/tree/master/examples/services

service description to ease the modeling of services having multiple variants. The next step are cost calculations based on a feature model-based variant modeling as a part of the CYCLONE IaaS Registry implementation in Use Case 3. It is expected that additional use cases within the application area could require extended modeling, for example, to represent commonly encountered feature matrices or to map complex pricing schemes. This leads to the anticipation that the plain model representation should help reduce the implementation effort of further extensions in the future.

Matchmaking (Requirement 5)
The matchmaker uses constraint programming (CP) methods as this is a proven method to handle the complexity of the service matching problem. With CP the problem definition stays intact in the code, which ensures that the business logic is documented as code. Due to the requirements of soft constraints and detailed rankings for services, CP is especially suitable for the problem as it has built-in mechanisms to support these features. However, the CP service matching approaches suggested so far cover only numeric QoS properties. Therefore, the registry extends constraint programming with constraint models for feature lists. The implementation of the CP-based matchmaker by Zilci implements all the relevant functionality for matchmaking in service registries, as described in [195].

3.1.10 Registry performance characteristics

It should be fairly obvious that the performance of the registry is dependent on the request volume and the sizing. As each component can be scaled independently from the others, the service registry architecture can support almost any performance requirement. For example, "read heavy" scenarios with many queries but few updates would benefit from additional web servers and caches, while "write heavy" use cases are implemented best using additional backend, evaluator, and database nodes.

The next subsections iterate all components and describe their specific performance characteristics. This should help to estimate what performance to expect from any use case implementation and how to adjust the sizing of components, if necessary.

SDL-NG and Business Vocabularies
The performance of both depends mainly on the efficiency of the SDL-NG framework implementation to load services and execute service descriptions. Both the load operation and the execution of a service description is implemented using common Ruby methods, relying on very basic language functions such as defining classes and objects and setting instance properties. Thus, the actual performance of these functions is directly correspondent to the performance of the Ruby implementation.

While Ruby has shown a substantial performance improvement over the last years, the language maintainer Matsumoto established further improvements in Ruby performance as one of the three main goals for the language evolution in the coming years [145].

The time it takes to load the vocabularies and services was measured in the current version of the Open Service Compendium to provide a rough estimation of the service evaluation performance. For this, the `process_-service_descriptions` script was amended by two calls to the `Benchmark` module of Ruby, benchmarking the `load_vocabulary_from_path` and `load_-service_from_path` methods. The benchmark was executed using the x86_-64 version of Ruby 2.4.1 running under 64-bit Archlinux on an Intel Core i7-6700 CPU using a Samsung SSD 950 PRO 512GB. On this setup, loading the vocabulary requires on average 20ms of CPU time while evaluating a single service description is performed in 2.5ms on average. Using a single thread, this machine would be capable of evaluating approximately 400 service descriptions per second, providing enough performance for even more intensive service registry use cases.

Rails Backend

There is no complex business logic implemented in the Rails Backend controllers. Instead, most HTTP requests are transformed directly into database queries to retrieve services or into push operations to the Redis Job Queue to update services. Therefore, similar performance can be expected from the Rails backend as for other typical Ruby on Rails-based applications. Beneficial for the performance of the backend is the good cacheability of the responses, especially the list of services and the service output. As both are stateless operations, their output can also be served from either a load-balanced web cluster or a globally distributed content delivery network. This would enhance the performance of the backend substantially.

Client

The view templates that are used by the TRESOR Broker are quite simple, as they consist of plain elements such as forms and tables. As they are not dependent on any user-specific state, they are also highly cacheable. However, the perceived performance of the registry is impacted by the delay that is introduced by the network round trips to the server. Their amount is dependent on the network latency between the end user and the location of the registry.

In contrast, the OSC AngularJS frontend allows the client to render the interface, leading to very quick reactions. In this case however, the performance of the user agents directly impacts the overall UI performance. Yet, as the frontend implementation is quite simple and both JavaScript engines and client PCs are getting more powerful, this is not expected to be a major performance aspect of the service registry. At last, it should be noted that the frontend is quite usable even on a modern smart phone.

Service Evaluator

There are different trade-offs between performance and security in the implementation options for the service evaluator. On the one hand, using virtual machines, for example, provided on a public IaaS platform, would be the most secure option. However, there are certain CPU overheads and memory inefficiencies which become a considerable aspect as the number of evaluators rises to meet high request demands. On the other hand, using containers or evaluating service descriptions directly in the backend provides better performance as, for example, there are no communication overheads and different evaluator instances can share memory and disk space very efficiently.

This option, however, provides very few protection measures against malicious service requests, making it only viable in fully trusted environments.

Redis Job Queue

Compared to the other components, the role of the Redis job queue in the registry architecture is only marginal. The main performance aspects to consider are memory consumption and CPU requirements. Regarding memory consumption, as Redis is an in-memory database, the maximum size of the queue is dependent on the amount of available memory on the target system. In turn, the size of a single queue element is mainly dependent on the size of the service description to be evaluated. As an example, the service descriptions in Use Case 2 (Cloud Storage Broker) are at max 5kB in size, resulting in approximately 6kB Redis key size, including metadata. Assuming 1.5GB free RAM on an Amazon EC2 "small" instance, this would allow a tremendously large queue of 262k service evaluation requests. Regarding CPU requirements, it should be noted that Redis performs very well due to its in-memory nature, achieving 100k+ transactions per second, even on older hardware.

Constraint-based Matchmaker

The performance of the constraint-based Matchmaker is defined mostly by the performance of the Choco Solver 2 [115] and the underlying Java platform. The matchmaking is performed in-memory and consists of iterating the list of services and ranking them based on how well they fit to the given constraints. This can be executed very efficiently when the list of services is first filtered externally (e.g., by selecting only services within a certain category) and then applying constraints in a Java VM that has been "warmed up" by preceding requests.

Database

There are many resources on the Internet that describe the performance characteristics of MongoDB. Furthermore, the official manual also provides information about different performance aspects in [112]. The main issue to consider here is the property of MongoDB to perform best when there is a bit more free memory than the size of the stored dataset. The dataset size of the service registry is dependent on the complexity of the service descriptions and the amount of previous service versions that are preserved. As an example, the average size of a MongoDB document in the Open Service Compendium is 15kB. In use cases featuring a similar service description complexity, this would allow persisting around 70k service versions per 1 GByte of RAM.

3.2 Trusted Cloud Transfer Protocol

After Section 2.2.4 provided a general overview of TCTP, this chapter details the protocol in detail by iterating the different parts of the protocol and showing the corresponding HTTP messages. It concludes with a short overview of its implementation. For simplification purposes, the HTTP `Content-Length` headers are left out within all listings and the messages are not chunked.

3.2.1 HTTP Application Layer Encryption Channels (HALECs).

HALECs wrap regular TLS connections and represent their transmissions as regular HTTP messages. They are created by the *TCTP handshake*, which is a regular TLS handshake wrapped in HTTP. HALECs transform the stream of HTTP entity-bodies into encrypted and authenticated TLS records which are sent instead of the plaintext entity-body and can be authenticated and decrypted by the receiving side of the communication. HALECs are identified by URLs that can be used in a RESTful manner, for example, DELETEing the channel via its URL to disconnect the wrapped TLS connection.

As HTTP does not guarantee an unvarying order of HTTP messages sent in parallel, an out-of-order TCTP response would invalidate the TLS HMAC and render the HALEC unusable. Parallel processing of TCTP traffic is therefore only possible when more than one HALEC is created and each parallel message uses a different HALEC.

HALECs are persisted by saving their TLS session state and security parameters. Therefore, they do not need to be "closed". TCTP implementations may invalidate HALECs to release resources that were required for HALEC persistence. In this case, user agents would repeat the TCTP handshake to create new HALECs.

3.2.2 TCTP Discovery

The TCTP discovery allows TCTP clients to retrieve information on how to create HALECs for accessing protected resources. Instead of defining static paths, using a discovery procedure enables every origin server to establish its own URLs. It is performed by the HTTP `OPTIONS` method to the asterisk ("`*`") URL sending the `Accept: text/prs.tctp-discovery` request-header, as shown in Listing 3.10.

Listing 3.10 TCTP Discovery

```
OPTIONS * HTTP/1.1
Host: www.example.com
Accept: text/prs.tctp-discovery
```

The TCTP discovery information consists of 2-tuples separated by colons. The first element is a regular expressions that matches URLs of one or more protected resource. The second element is a regular expression replacement string that returns the URL which should be used to establish a HALEC to communicate with the resource matched by the first elements regex. URLs that are not protected are indicated by an empty second element. The expressions are matched from top to bottom, where the first matched pattern decides the outcome of the discovery.

In Listing 3.11, the root URL, the service root URLs and all static assets are unprotected. The use of regular expression replacement strings allow a very flexible definition of recovery information, as represented by the fourth tuple. Here, different HALEC creation URIs are conveyed for different services. For example, the resource `/servicea/home` would have its HALECs created using

`/servicea/halecs` while `/serviceb/home` using `/serviceb/halecs`. These URIs could be reverse proxied and operated by different parties, each having control over their own secret TLS session states - a highly relevant scenario for cloud ecosystems.

Listing 3.11 Extended TCTP Discovery

```
HTTP/1.1 200 OK
Content-Type: text/prs.tctp-discovery

/:
/(service(.+?))?:
/(service(.+?)/)?static.*:
/(service(.+?)/)?.*:/\1/halecs
```

3.2.3 TCTP Handshake

The creation of an HALEC is performed by an HTTP `POST` request to the HALEC creation URL containing a TLS `client_hello` record. After an HALEC is created, the origin server sends the HTTP response code `201 Created` and a `Location` response-header field containing the URL of the new HALEC.

This handshake sequence closely follows the `POST` method definition in the HTTP specification [53, sec. 4.3.4]. The HTTP response contains the TLS handshake response record as an entity-body. Both handshake request and response are shown in Listing 3.12.

Listing 3.12 HALEC Creation Request and Response

```
POST /halecs HTTP/1.1
Host: www.example.com

<TLS client_hello record>

----------------------------------------------------
HTTP/1.1 201 Created
Location: http://www.example.com/halecs/1kX28fAms

<TLS server_hello, certificate, server_key_exchange>
```

The remaining handshake is executed by sending an HTTP `POST` request to the HALEC URL containing the remaining TLS handshake records, for example, `client_key_exchange`, `change_cipher_spec`, and `finished`. Listing 3.13 illustrates this behavior.

The HALEC URL can later be used to send TLS closure alerts, to perform a TLS renegotiation, and to close the HALEC by sending an HTTP `DELETE`.

Listing 3.13 TCTP Handshake Request and Response

```
POST /halecs/1kX28fAms HTTP/1.1
Host: www.example.com

<TLS client_key_exchange, change_cipher_spec, finished>

-------------------------------------------------------
HTTP/1.1 200 OK

<TLS change_cipher_spec, finished>
```

3.2.4 TCTP Entity-body Encryption

After the handshake establishes an appropriate TLS state, the HALEC is used to secure HTTP entity-bodies. The header field `Content-Encoding: encrypted` designates such encrypted messages. User agents request encrypted content through the `Accept-Encoding: encrypted` header. If an HTTP message contains a payload, the HALEC URL is sent as the first line of the payload, preceded by the encrypted entity-body. The user agent and origin server should be required to send a `Cache-Control` header of the value `no-cache`, so that no intermediary returns an encrypted server response from cache, as this would invalidate the HMAC.

Listing 3.14 TCTP-encrypted Communication

```
POST /patients/070386/details HTTP/1.1
Host: www.example.com
Accept-Encoding: encrypted
Content-Type: application/json
Content-Encoding: encrypted
Cache-Control: no-cache

http://www.example.com/halecs/1kX28fAms
<TLS application data records>

---------------------------------------
HTTP/1.1 200 OK
Content-Encoding: encrypted
Cache-Control: no-cache

http://www.example.com/halecs/1kX28fAms
<TLS application data records>
```

Listing 3.14 shows how HTTP applications can be designed in such a way that their HTTP communication shows *varying information confidentiality*: while the header signifies a general "update details of patient `070386`", the entity-body can convey sensitive medical information that is protected by TCTP.

3.2.5 TCTP Implementation

TCTP was implemented using Ruby and published as the `tctp-rack` Ruby gem. This library provides three essential features:

1. An SSL "engine" that relies on OpenSSL for the encryption primitives, providing a fast and well-tested implementation basis. It is written as a Ruby extension in C to achieve high performance. Its main responsibilities are invoking OpenSSL methods to conduct TLS communication and providing the data that would be sent over the network as regular Ruby strings. The engine implementation is based on a similar component of the stable and mature Ruby Puma web server which is the default web server for newly created Ruby on Rails applications and is used on the popular Heroku PaaS.
2. A Ruby implementation of server and client HALECs which interface with the SSL engine and offer asynchronous methods to decrypt and encrypt Ruby strings in a background thread. The server HALEC implementation can either use an existing cryptographic certificate or create a new certificate on the fly.
3. A middleware for the Ruby Rack web server interface. All popular Ruby web frameworks, for example, Ruby on Rails and Sinatra, are based on Ruby Rack so that TCTP can be added seamlessly through `tctp-rack` to any application written for these frameworks. To prevent session hijacking using intercepted cookies, the middleware contains a mapping that allows the use of HALECs only within the same sessions they have been created. This prevents the use of intercepted authentication cookies with self-created HALECs and strengthens the cookie hijacking protection of web applications.

To reduce the effort of porting this TCTP implementation to other programming environments, it only relies on common classes, and features a simple architecture. Thus, it presents a reference implementation that shows a generic solution design that is not too specific to Ruby.

3.3 Distributed Cloud Proxy

This section provides an overview about the two implemented versions of the distributed cloud proxy. The first is a Java prototype implemented in the early phase of TRESOR. The second "final" proxy was implemented using the Ruby programming language and was deployed to production.

3.3.1 Early proxy prototype

Before implementing the final version of the distributed cloud proxy, a prototype was created in order to gain experience with custom HTTP-based proxying based on non-blocking I/O. An evaluation of this implementation can be found in Section 4.3. The proxy source is available on GitHub in [153].

3.3.1.1 Technology

The proxy uses non-blocking and asynchronous functions to enable highly scalable I/O operations using the Java New I/O (NIO) API [134]. It relies on the Grizzly Framework [121] which provides abstractions for those rather "low-level" functions. Performance being one of the major concerns, Grizzly has shown impressive characteristics, for example, in [110] where it was used to implement a Network File System (NFS) server and performed better than the standard Linux kernel-based implementation. At last, it provides functions for HTTP processing as well as a customizable TLS engine that assists in the implementation of the proxy.

The industry standard OSGi [124], more specifically, the OSGi application server platform Eclipse Virgo [176] provides a modularized base for the proxy. It also lowers the deployment effort considerably, as the OSGi application modules can be independently reconfigured, updated, and replaced at runtime.

3.3.1.2 Architecture

The proof of concept implementation consists of two main OSGi bundles: the *proxy core*, containing the proxy runtime, as well as the *proxy model* which persists its configuration. The prototype supports the following functionality:

- **Authentication.** The proxy matches URI patterns to authentication rules and authenticates users through a password database.
- **Relaying identities.** After users are authenticated, the proxy relays their identities to the downstream proxies by using a special HTTP header.
- **Routing and SSL.** The proxies can encrypt traffic using SSL and route incoming messages as defined by the proxy configuration.

3.3.2 Final Proxy

The final proxy was written in Ruby to better integrate the Ruby TCTP implementation and to provide a concise and well-maintainable code base. It makes use of the `eventmachine` Ruby gem which provides an "event-driven I/O and lightweight concurrency library for Ruby" [31] using the *Reactor* pattern. The proxy combines protocol-logic written in comprehensible Ruby code with performance-critical I/O written as C++ ruby extension, creating a fast, yet easy to extend implementation.

After an introductory section, this section explains the proxy in detail, explicating the deployment and configuration options before giving an overview about the code structure. The last section iterates all functions that have been implemented and explains their use.

3.3.2.1 Introduction

As explained in the proxy concept in Section 2.3.3, there are three proxy instances, each serving a different role in enabling managed cloud service consumption. To enable significant code reuse between the instances when implementing this concept, the final proxy features a highly modular architecture

[31] https://github.com/eventmachine/eventmachine

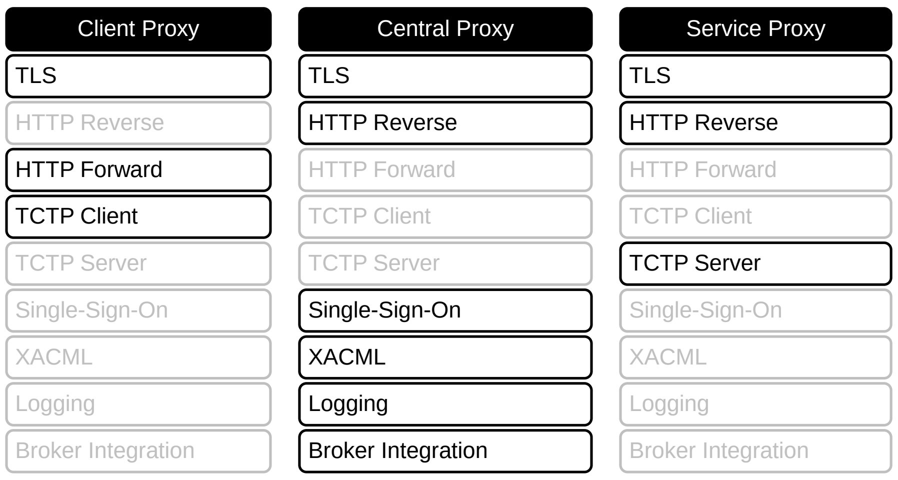


Figure 3.10: Proxy Modules and Functional Distribution

where each module can be enabled and configured independently from the others. This allows a flexible distribution of functionality between the three instances and supports future use cases that possibly have a different functional distribution. Figure 3.10 shows which modules are enabled for the three different instances. The available proxy modules are:

- TLS transport security (point-to-point)
- HTTP reverse and forward proxying
- TCTP client and server
- Web-based single sign-on
- Distributed authorization using XACML
- Remote logging
- Integration with the cloud service registry

While the common case for the proxy distribution are those three parties, other deployment scenarios could see any number of involved parties, as a modular cloud proxy architecture allows flexible configuration of the different instances.

3.3.2.2 Proxy Communication Sequence

Figure 3.11 presents the communication sequence when a cloud consumer accesses a booked service within an ecosystem featuring a central proxy. This access is presumed to be the first after booking the service, therefore requiring authentication and authorization. The *central proxy* acts as the reverse proxy for one or more *cloud services*. The proxy is integrated with a *Federation Provider* which homogenizes the user identities of the different *Identity Providers* partaking in the respective ecosystem, for example, the universities and other institutions partaking in a national research and education network. The proxy also communicates with a *service broker* that persists information about booked services, most importantly, the identifiers and the URL to the cloud consumer's

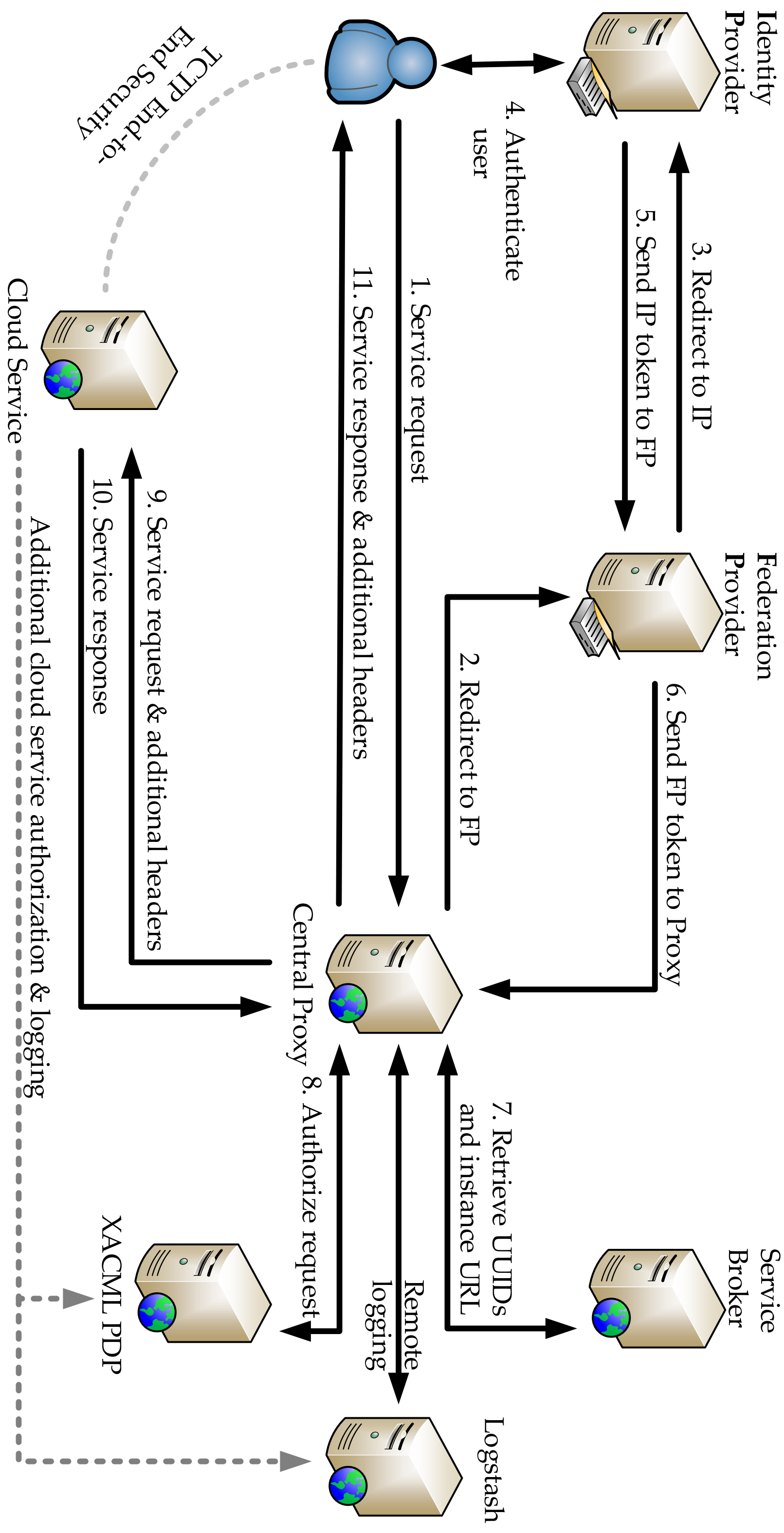


Figure 3.11: Proxy Communication Sequence

service instance. Distributed access control is provided through the integration with a XACML Policy Decision Point (PDP). At last, the proxy supports logging into an ELK distribution. The following list provides a detailed explanation of the whole sequence:

1. User request to service URL

   The sequence is initiated by the first HTTP request of service consumers to the service URL through their browsers, for example, `http://demo.service.example.com`. The DNS entry of this host name should point to the proxy, effectively making it a *reverse proxy* for the services. The dotted grey line in the figure indicates the potential TCTP key exchange and encryption that is established by other proxies.

2. Proxy redirect to the Federation Provider

   As there is no existing proxy session for this browser, the Proxy redirects the user to the Federation Provider. The proxy appends a “where do you come from” parameter to the redirect URL so that the Federation Provider can redirect the user back to the current URL after authentication.

3. Federation Provider redirect to the Identity Provider

   As the browser also does not have a session with the Federation Provider, it uses the SAML WebSSO mechanism to create an authentication request and POSTs it through a hidden form to the respective Identity Provider of the user. There are different mechanisms how the Federation Provider could decide where to POST this form, for example, using a mapping between a range of user IPs and the responsible Identity Provider.

4. User authentication with the Identity Provider

   After the Identity Provider validates the authentication request, it initiates the user authentication process. The Proxy workflow is agnostic to the concrete mechanism that is used for user authentication, for example, username/password pairs or smart card authentication.

5. Identity Provider SAML token transmission to Federation Provider

   Regardless of the authentication mechanism, its result is returned as a signed SAML token to the Federation Provider. This token contains the identity of the user as well as any number of user attributes, for example, group memberships.

6. Federation Provider SAML token transmission to the Proxy

   After validating the token, the Federation Provider transforms the specific Identity Provider attributes into a set of common ecosystem attributes. This offers a generic mechanism to consume those attributes by the cloud applications as well as the XACML components. After the mapping, the Federation Provider returns a signed SAML token with the mapped attributes to the proxy.

7. Retrieval of service instance UUID and URL from the service broker by the proxy

After a user session is established, the proxy uses the identity of the users to query the service broker for two important information: first, it uses the symbolic name of the service contained in the HTTP host name (for example, "`demo`") as well as the name of the user organization to retrieve the UUIDs of the service and the organizations. These UUIDs serve as a persistent identifier in subsequent calls to the broker, Logstash, and the XACML PDP, as both names can potentially change. As a second step, it uses those UUIDs to query the URL where the booked instance for this specific organization is deployed.

8. Authorization of the request using the XACML PDP

   The proxy asks the XACML PDP to authorize the request, based on all information available at this point: the service and organization UUIDs, the user ID, all user attributes, the HTTP method, and the HTTP path. The XACML PDP uses a set of rules defined by the organizations to arrive at an allow/deny decision.

9. Relaying the service request and additional headers to the cloud service

   Authenticated and authorized requests are relayed to the previously retrieved service instance URL. They are augmented with a set of headers conveying information about the request, for example, the user ID and the user attributes (see Table 3.7).

10. Responding to the cloud service request

    The cloud service can now respond to the request. As the request already contains a lot of information, the cloud service does not have to query external services in many cases, for example, for authentication. However, the cloud service can use backend services for additional purposes, for example, logging application messages and applying service-specific authorization rules.

11. Relaying the service response and additional headers to the user

    After receiving the response, the proxy relays it to the user, augmenting it with a set of HTTP headers (see Table 3.8). These can be used, for example, to debug the XACML and broker integration if there are any problems. As the user has authenticated, the proxy also adds a `Cookie` header to the response. This creates a proxy single sign-on session which persists the transmitted user attributes for the lifetime of this session.

Besides the augmented HTTP headers, the proxy additionally uses the logging system to persist diagnostic messages. When available, these include the UUIDs of the services and organizations which can be used to filter the logs for improved debugging facilities.

3.3.2.3 Deployment and Configuration

In order to support different target environments, there are two options how the proxy can be deployed and executed:

1. Deploying from source

The source code of the proxy contains a `Gemfile` that specifies the dependencies of the proxy on certain versions of other gems. Before the proxy can be executed, these dependencies need to be downloaded using the `bundle` command, provided by the `bundler` gem. Afterwards, there are two scripts available for executing the Proxy: `bin/proxy.rb` for running it as a standalone application, and `bin/proxy_daemon.rb` for execution as a daemon.

2. Deploying via Docker

   The source also contains a `Dockerfile` that builds an Ubuntu-based container and installs the Ruby Version Manager (RVM)[32], `bundler`, and the proxy dependencies. The Docker container entrypoint is the `bin/proxy_-docker.sh` script which loads RVM and executes `bin/proxy.rb`.

The proxy configuration is provided on the command line. When specifying the `--help` option, the proxy outputs all possible configuration options, as shown in Listing 3.15.

3.3.2.4 Source Code Structure

The classes of the proxy belong to four main modules:

1. `Proxy`

   This module contains four classes that implement common functionality: `Connection`, that delegates HTTP handling, `ConnectionPool`, accelerating HTTP processing through connection pooling of backend connections, `Request`, abstracting information about proxied requests, and `TresorProxy`, the main program.

2. `Frontend`

   The classes in this module implement the actions that should be carried out with incoming connections, for example, relaying them to backend servers, and conducting TCTP server functions.

3. `Backend`

   The module classes specify the messages that should be sent to upstream servers, for example, relayed HTTP messages, XACML authorization requests, and TCTP client functions.

4. `TCTP`

   Provides auxiliary functions to implement TCTP, for example, a HALEC registry.

Additionally, loading `lib/tresor/logging.rb` extends all other classes with the `log` and `log_remote` methods which provide a console output as well as a remote Logstash logging facility respectively.

[32]https://rvm.io

Listing 3.15 Proxy Configuration Options

```
Usage: proxy.rb [options]
    -b, --broker             The URL of the TRESOR broker
    -i, --ip                 The ip address to bind to
                             (default: all)
    -p, --port               The port number (default: 80)
    -n, --hostname           The HTTP hostname of the proxy
                             (default: proxy.local)
    -P, --threadpool         The Eventmachine thread pool size
                             (default: 20)
    -t, --trace              Enable tracing
    -l, --loglevel           Specify log level (FATAL, ERROR, WARN,
                             INFO, DEBUG - default INFO)
        --logfile            Specify log file
        --logserver          Specify remote logstash server uri,
                             e.g., tcp://example.org:12345
    -C, --tctp_client        Enable TCTP client
    -S, --tctp_server        Enable TCTP server
        --tls                Enable TLS
        --tls_key            Path to TLS key
        --tls_crt            Path to TLS server certificate
        --reverse            Load reverse proxy settings from YAML
                             file
        --raw_output         Output RAW data on console
        --sso                Perform claims based authentication
        --xacml              Perform XACML
        --pdpurl             The PDP URL
        --fpurl              The SSO federation provider URL
        --hrurl              The SSO home realm URL
    -h, --help               Display this help message.
```

3.3.2.5 Using EventMachine for the Proxy Implementation

Implementing an EventMachine server requires writing a class that inherits from `EventMachine::Connection`, in the case of the proxy, the `Proxy::Connection` class. Whenever EventMachine receives a new connection, it instantiates this class and calls instance methods on important events. The crucial methods here are `post_init`, called immediately after the network connection has been established, `receive_data`, when data is ready to be processed, and `unbind`, whenever a connection is closed. EventMachine does not enforce any message boundaries and also coalesces multiple network packets into a single string for performance reasons. Thus, there is no guarantee that the calls provide sensible data for immediate processing. In effect, this provides an efficient manner of implementing network servers while it also requires some thoughts how to cope with the increased complexity of the implementation.

The proxy uses an HTTP parser to impose message boundaries onto the stream of network data to handle it sensibly. The proxy `Connection` class passes all received data to an instance of the HTTP parser provided by the `http_parser.rb` gem[33], which is a Ruby wrapper around the Node.JS HTTP parser[34]. As this parser was written in C and is widely used, it is very efficient in terms of CPU and memory use.

There are three HTTP parser callback methods that are implemented by the proxy, which carry out all handling of network traffic:

- `on_headers_complete`

  When all headers have been received, the HTTP parser passes them as a Ruby Hash to this method. It then decides which proxy facility to involve, for example, if it relays the message to an upstream server, encrypts it through TCTP, or provides TCTP server functions. Each Proxy facility is implemented in a different class that is derived from `Frontend::FrontendHandler`. These classes provide two callback methods that are called from `Proxy::Connection`: `on_body` and `on_message_complete`. These receive the data from the HTTP parser and implement a strategy for handling it.

  The `Proxy::Connection` class delegates the decision if a certain handler should be used to the handler by calling all `can_handle?` class methods of every handler and using the first that returns `true`. For example, the `HTTPRelayFrontendHandler#can_handle?` method returns `true` only if the Proxy was configured as a forward Proxy or there is a configured mapping from the requested HTTP `Host` to a backend service. The creation of the respective handler is deferred using promises as it can take some time and would thus block the reactor thread.

- `on_body` and `on_message_complete`

  When HTTP body parts are received, the HTTP parser provides them as strings to the `on_body` callback. The callback `on_message_complete` signals the complete reception of the HTTP message. In both cases, the `Proxy::Connection` instance delegates both methods to the frontend handler by invoking similarly named methods on the handler instances. If the handler has not been constructed, these calls are also deferred, thus creating a kind of buffer for received messages. This is the case when, for example, the proxy receives requests for hosts to which it has not yet established a backend connection.

Table 3.5 provides a list of all frontend handlers, their purpose, as well as a categorization into Single-sign on (SSO), distributed authorization (XACML), TCTP, and regular HTTP handling.

[33]https://rubygems.org/gems/http_parser.rb
[34]https://github.com/nodejs/http-parser

Table 3.5: List of Proxy Frontend Handlers

Area	Name	Purpose
SSO	RedirectToSSOFrontendHandler	Redirects to the single sign-on server
SSO	ProcessSAMLResponseFrontendHandler	Processes the SAML response from the single sign-on server
XACML	DenyIfNotAuthorizedHandler	Denies the request if it cannot be authorized via XACML
TCTP	TCTPDiscoveryFrontendHandler	Returns TCTP discovery information
TCTP	TCTPHalecCreationFrontendHandler	Creates TCTP HALECs
TCTP	TCTPHandshakeFrontendHandler	Handles TCTP handshakes
TCTP	HTTPEncryptingRelayFrontendHandler	Encrypts messages using TCTP before relaying them
HTTP	HTTPRelayFrontendHandler	Relays HTTP messages to an upstream server
HTTP	NotSupportedRequestHandler	Returns an error message when receiving unsupported requests
HTTP	TresorProxyFrontendHandler	Sends a simple “Hello” message for testing purposes.

Table 3.6: List of Proxy Backend Handlers

Area	Name	Purpose
TCTP	TCTPDiscoveryBackendHandler	Queries upstream servers for their TCTP abilities
TCTP	TCTPHandshakeBackendHandler	Conducts a TCTP handshake with upstream servers
TCTP	TCTPEncryptToBackendHandler	Encrypts HTTP messages before relaying them
HTTP	RelayingBackendHandler	Relays HTTP messages to upstream servers

The frontend handlers that relay HTTP messages to upstream servers (`HTTPRelayFrontendHandler` and `HTTPEncryptingRelayFrontendHandler`) use descendants of the `Backend::BackendHandler` class to implement the different communication strategies. Table 3.6 provides an overview about the implemented backend handlers. For performance reasons, the proxy contains the connection pool class `Proxy::ConnectionPool` which reuses connections to backend servers instead of creating one upstream connection for each client. Like the creation of frontend handlers, in order to prevent blocking the reactor thread, the backend handler creation is deferred and the data to be relayed is buffered in a backend connection promise.

3.3.2.6 Proxy Facilities

The following subsections iterate the different facilities that the proxy offers, along with a description of their implementation.

HTTP Proxying: Reverse and Forward

The default operating mode for HTTP proxying is *forward proxying*, where the HTTP client is aware of the proxy and instructs it to forward requests to client-defined HTTP servers. Additionally, the proxy supports *reverse proxying*, where the proxy itself transparently forwards HTTP requests to upstream servers without client knowledge. To create a performant and reliable proxy, the handlers `HTTPRelayFrontendHandler` and `RelayingBackendHandler` make use of buffering and connection pooling.

To know which upstream server to contact, the proxy needs "reverse mappings" from incoming HTTP `Host` values to the respective upstream server URL. These are provided in a YAML file whose path is specified using the `--reverse` command line option. The structure of this file is exemplified in Listing 3.16 showing its quite simple syntax: the start of the document is signified by the `---` string. Afterwards, there can be any number of mappings, each of the form `'host': 'url'` where `host` specifies the value of the `Host` HTTP header that should be proxied and `url` the full URL, including "`http://`", of the upstream server.

Listing 3.16 Proxy reverse mappings

```
---
'<HOST 1>': '<URL 1>'
'<HOST 2>': '<URL 2>'
'<HOST n>': '<URL n>'
```

The relaying handlers provide facilities to register "request" HTTP headers that are additionally sent to the upstream servers as well as "response" headers that are returned to the clients. These can be used for a wide range of purposes, for example, authentication, authorization, or debugging. Table 3.7 and Table 3.8 present a list of all possible headers, divided into the areas HTTP proxying, Single Sign-on (SSO), distributed access control (XACML) and broker integration.

Table 3.7: Proxy Request Headers

Area	Header name	Description
HTTP	`X-Forwarded-Host`	The HTTP `Host` sent by the client
SSO	`TRESOR-Attribute`	Attributes of the authenticated subject in the form “`<URL> <value>`”
SSO	`TRESOR-Identity`	The identity of the authenticated subject, e.g., `DHZB\JStock`
SSO	`TRESOR-Organization`	The identifier of the organization of the authenticated subject, e.g., `MEDISITE`
Broker	`TRESOR-Service-UUID`	The TRESOR broker service UUID that this request tries to invoke
Broker	`TRESOR-Organization-UUID`	The TRESOR broker organization UUID that the current subject belongs to

Table 3.8: Proxy Response Headers

Area	Header name	Description
Broker	`TRESOR-Broker-Exception`	An exception in the broker integration
Broker	`TRESOR-Broker-Requested-Name`	The symbolic name of the service in the broker, e.g. `demo`
Broker	`TRESOR-Broker-Response`	The response of the broker
XACML	`TRESOR-XACML-Decision`	The XACML decision (“Permit”, “Deny”, or “Intermediate”)
XACML	`TRESOR-XACML-Error`	The XACML error description
XACML	`TRESOR-XACML-HTTP-Error`	An XACML HTTP error, e.g., if parsing the response failed
XACML	`TRESOR-XACML-Exception`	An exception in the XACML module
XACML	`TRESOR-XACML-Response`	The response from the PDP
XACML	`TRESOR-XACML-Request`	The proxy PEP request that was sent to the PDP.

TCTP Client & Server

For implementing the TCTP client and server functionality, the proxy makes use of the `rack-tctp` gem which was presented in Section 3.2.5. There are TCTP frontend and backend handlers which provide TCTP discovery, handshaking, and encryption as both client and server. At last, there are auxiliary classes contained in the `Tresor::TCTP` module which provide some minor functionality, such as an internally-used HALEC registry. Configuring the proxy for TCTP is quite straightforward: `-C` enables the TCTP client, `-S` enables the TCTP server. The debug output of the proxy provides comprehensive status information about the TCTP subsystem.

Web Single Sign-On

There are three constituents of the web single sign-on functionality:

1. Redirecting users to the Federation Provider

   The proxy uses the `RedirectToSSOFrontendHandler` to redirect users to the Federation Provider, when configured accordingly. The redirection target contains two further URLs: `wtrealm`, a URL where the authentication results should be sent as well as `whr`, an internal identifier for the proxy. The `wtrealm` URL furthermore contains a "Where do you come from?" URL that the proxy uses to redirect users to their initially requested URL.

2. Processing Federation Provider responses

   After the Federation Provider authentication workflow is complete, the proxy receives an HTTP POST request to the `wtrealm` URL and uses the `ProcessSAMLResponseFrontendHandler` to handle this request. Parsing the received SAML token is implemented through the `ClaimSSOSecurityToken` which provides a wrapper around the received XML. It allows to retrieve the SAML `NameID` assertion, the organization, as well as a hash of user attributes. After the SAML token is processed, the proxy redirects to the "Where do you come from?" URL, in effect resuming the service consumption.

3. Managing SSO sessions

   After authentication, the proxy additionally issues a `tresor_sso_id` cookie which corresponds to a key of an internal Hash persisting the attributes of the authenticated user. As long as the cookie is not expired and the proxy was not restarted, these user attributes will be relayed to the cloud services as additional request headers, sparing the users from needing to log-in again.

XACML-based Authorization

The proxy can act as an XACML Policy Enforcement Point (PEP), asking an external Policy Decision Point (PDP) if requests are authorized or not. The proxy PEP is compatible to the template-based policy generation interface for RESTful web services which was devised by Raschke and Zickau and presented in [132]. The PEP is implemented in the `DenyIfNotAuthorizedHandler` which asynchronously creates the authorization request and sends it to the PDP. To ensure good maintainability, the request template is provided as a Ruby ERB file [35] which is easy to comprehend and modify.

Logging & Monitoring

There are two methods in the proxy that provide logging functionality. The first (`log`) relies on the `Logger` class that is contained in the Ruby standard library to output debug messages to the console. The other method, `log_remote`, transmits a structured log message to a Logstash instance. This is used for auditing and diagnostic purposes.

3.4 Security Architecture for Federated Multi-cloud Applications

Computer security is an enormously varied field of computer science. It is obvious that the limited resources available for creating this dissertation prevent creating an architecture to address security in an all-embracing manner. Taking up the general direction of the associated research projects, this thesis focuses on *providing specific, missing functionality and validating it in use cases of an existing ecosystem*, especially authentication and authorization on ALL cloud layers using federated identities in academic settings, for example, to enable both web-based single sign-on as well as SSH login using the same mechanisms.

In general, the functionality that the architecture provides should benefit both end-users, as well as their corresponding organizations by meeting relevant requirements and offering a set of best-practice and ready-to-use components. This is in line with the general goal of easing the management and deployment of multi-cloud applications.

The security architecture is **not** a holistic and generic security architecture, but more of a collection of practical tools that can be adapted and applied for related uses – either singularly or in combination. The components are designed for reusability as they rely on simplicity, that is, doing one thing and this thing well, as well as production-grade tools and established industry-recognized standards, for example, Keycloak and OpenID Connect.

Following the extreme programming approach of "You ain't gonna need it" and "Do the simplest thing that could possibly work" the architecture was built upon no other sources than the requirements of the use cases that were implemented. It would be impractical if it would try to meet requirements whose specifics are not known and whose realization cannot be validated having no clear use cases for them.

One example is the substantial area of "compliance": while "compliance" plays a major role for many corporations and other entities, there have been no concrete requirements in this area in any of the use cases that were addressed. Therefore, compliance is not addressed directly by any of the developments as there are no specific rules and regulations pertinent to the use cases.

This section first explains the objectives before presenting an overview about the security architecture. Afterwards, the three main focus areas of the architecture are explained in further detail.

[35]`lib/tresor/frontend/xacml/xacml_request.erb`

3.4.1 Overview: Objectives and Architecture

There is a great diversity in the IT security infrastructures of different organizations. It is therefore a challenging task to create a security infrastructure in a generic way, as the adaptability of the infrastructure components is highly dependent on the maturity and the scope of the IT security infrastructure of the organizations adopting it. This thesis follows the method of incrementally refining requirements with the stakeholders, especially the use case owners from the Bioinformatics domain. After conducting face-to-face meetings and security workshops, a number of security objectives were defined that the architecture needs to take up, divided into *global*, *cloud application*, and *cloud deployment* objectives. These objectives provide a good indicator to assess if the security architecture would provide benefits to other potential users, that is, taking up the architecture is far more beneficial to those potential users who share the same objectives.

Global Objectives

There are two main global security objectives. First of all, it should allow **federated log-in to reuse existing identities**, for example, eduGAIN federated user identities. The main rationale is helping non-technical end-users, for example, bioinformaticians, to log in to cloud resources using the same credentials they use for logging into their workstations. The second objective is providing **distributed logging for debugging and auditing purposes**. As explained before, this should help diagnosing problems and keeping an audit trail of users' activities.

Cloud Application Objectives

First of all, the architecture should provide easy-to-use mechanisms for **federated authentication** as well as **federated authorization** in order to raise the applications' security level by integrating a secure and tested web authentication solution. It should also **secure the transmission of sensitive data**, such as the biomedical data. At last, cloud applications should **use the distributed logging** for gathering their log output.

Cloud Deployment Objectives

One of the objectives is to **allow federated log-in to the deployment manager**. It should furthermore **use the federated identities within ACLs** used for controlling authorization of deployment actions (for example, creating a new application instance). At last a deployment manager should **deploy all the security components** and **use the distributed logging**, for example, to debug deployments or audit usage of cloud resources.

Security Architecture Overview

Figure 3.12 provides an overview about the constituents of the architecture. From an operations perspective, the main interaction is between the cloud service and the cloud service user. As with all ecosystems, there are multiple usable services, for example, deployed applications and containers, the deployment manager, the logging distribution, an IaaS platform, and an optional self-service portal. There are also different cloud service users, for example,

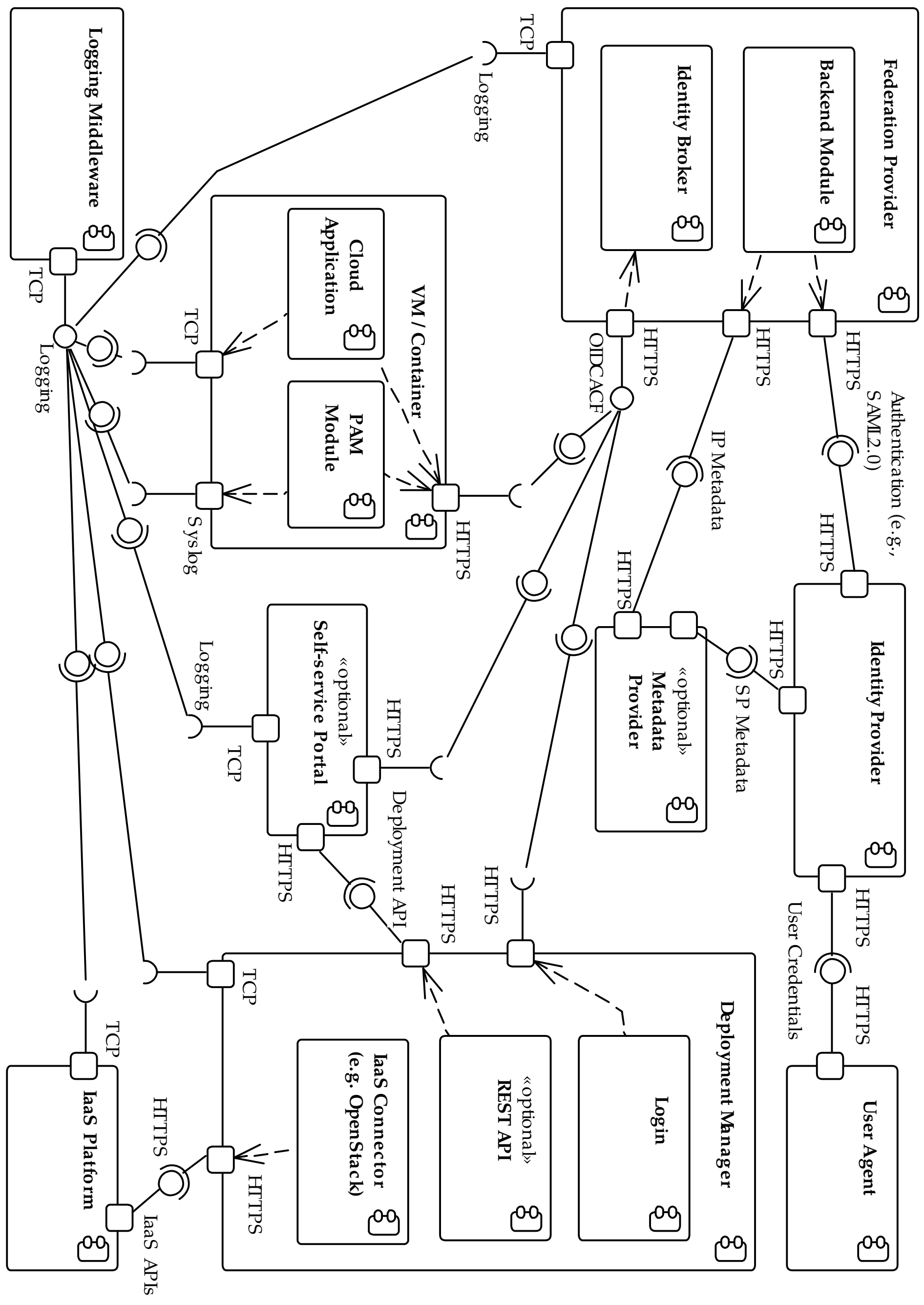


Figure 3.12: Security Architecture

end-users, VM developers, and cloud operators. As the architecture should be independent of any specific cloud model (public, private, hybrid, etc.) it dies not consider the specific role of cloud service providers.

One of the cornerstones of the architecture is the **Federation Provider**[36] which issues uniform user claims to relying applications, for example, users' identifiers, email addresses, and their home organizations. These claims are contained within JWTs retrieved using the OIDCACF. The Federation Provider contains the **Identity Broker**, letting users select the identity to be used for authentication, as well as the **Backend Modules**, implementing SAML 2.0 and OpenID Connect. The SAML 2.0 functionality is used to communicate with the **Identity Providers**, using an optional ***Metadata Provider***, for example, eduGAIN. As the end user's **User Agents** communicate directly with the identity providers, credentials are never transmitted to 3rd parties. Furthermore, end users can reuse their login sessions to achieve web-based single sign-on.

The **Logging Middleware**, for example, an ELK stack, unifies distributed log messages and should support, besides others, TCP-based and Syslog-compatible log receivers. Not shown in the figure are the **Logging Frontend**, allowing end user log consumption, as well as the **Logging Backend**, for example, a database or flat files, to persist the logs.

Deployed **Cloud Applications** rely on the OIDCACF to authenticate and authorize users - both on the application layer, through OpenID Connect libraries, as well as the OS layer, through the **PAM Module**. Depending on the concrete requirements, the PAM module maps identities either to a respective local user account, or to a shared user account. In contrast to Moonshot, the PAM module does not need a modified SSH client or server.[37]

The **Deployment Manager** supports multi-cloud application deployment, that is, it models application topologies, connects to different IaaS APIs, and offers a web- and a RESTful interface. It also allows end users to use the OIDCACF for logging in and writes its log messages to the logging middleware.

There is an optional architectural element, the **Self-service Portal**. It allows end users without technical background to use the deployment manager for instantiating VMs on the IaaS platform. This portal uses OIDCACF for authentication and authorization and logs to the network interfaces of the logging middleware. It communicates with the RESTful API of the deployment manager in order to deploy and scale applications on preconfigured clouds.

3.4.2 The Federation Provider

Figure 3.13 depicts the use of the Federation Provider for federated authentication: any system actor can perform federated user authentication: the deployed cloud applications, the deployment manager, the logging system, the IaaS platform, as well as the self-service portal. The Federation Provider relies on the local Identity Providers to authenticate users and provide assertions to the Federation Provider via the SAML 2.0 Web Authentication Workflow.

The cloud applications rely on the OpenID Connect authentication code flow (OIDCACF) to use federated identities for authentication and authorization.

[36]We use this term as "Federated Identity Provider" would be ambiguous: "provider of a federated identity" or "identity provider in a federation"?

[37]More details at https://github.com/cyclone-project/cyclone-python-pam

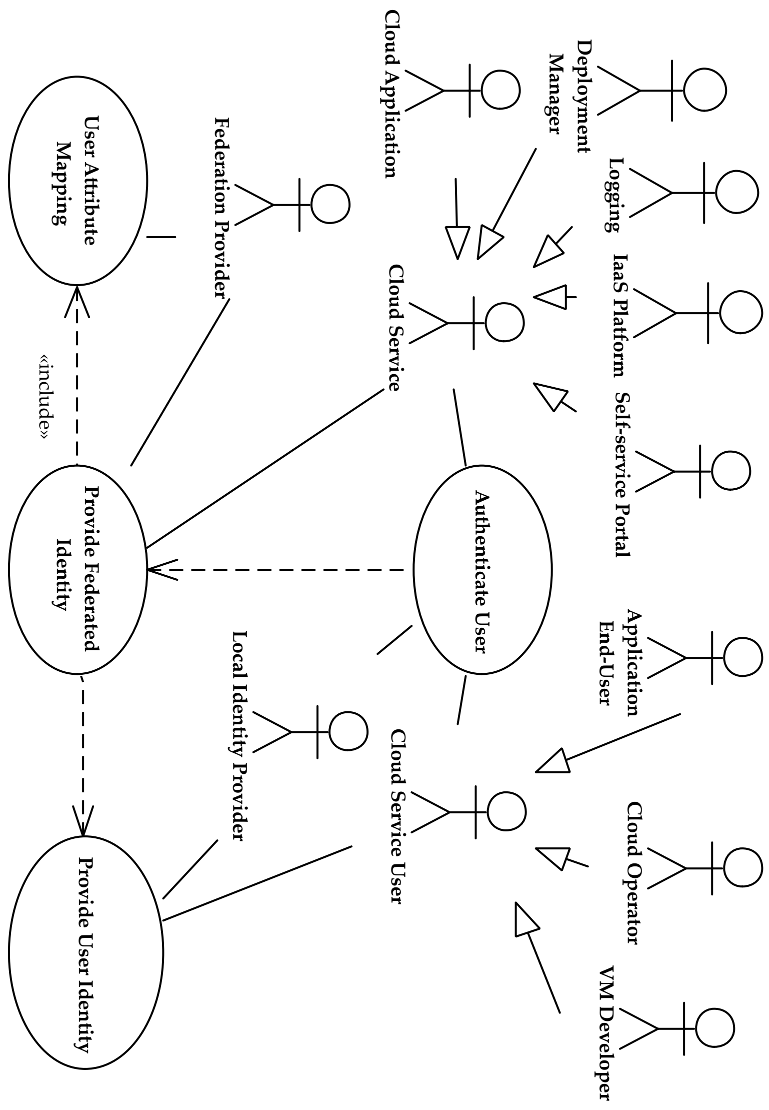


Figure 3.13: Federated Authentication

They transmit signed authentication requests and retrieve signed user identity claims to and from the *Federation Provider*. The Federation Provider uses the Identity Broker to display a list of Identity Providers for the users to chose from. The exchange of credentials is done solely between the clients and their Identity Provider, preventing disclosure of them to any system outside of the home realm.

User Attribute Mapping and OpenID Subject ID generation

Providing a federated identity includes user attribute mapping, transforming SAML user assertions into JSON Web Token claims, possibly using global, client-, or service specific configuration. One of the most relevant attributes for relying applications is the OpenID Subject ID (JWT `sub` claim) which represents a persistent user identifier. These are important for consistent mapping of federated accounts to application accounts, for example, to retain user data even when, for example, email addresses change.

3.4.3 Managing Multi-cloud Application Deployments

Figure 3.14 depicts how VM deployment is handled in the security architecture. First, deployment descriptions need to be created containing all the steps necessary to create new application instances. For example, SlipStream, the deployment manager used by CYCLONE, uses base images (e.g., "Ubuntu Linux LTS") as well as deployment scripts to describe how to install the respective application components on newly instantiated VMs. After all application modules have been prepared, the deployment manager calls the respective IaaS platform APIs to instantiate the cloud application, either for initial deployment, for subsequent scaling, or to tear down the application. This instantiation could be either initiated by cloud operators or through a self-service portal by regular end users.

Designing, creating, and integrating a multi-cloud application deployment lifecycle was not a focal point of the work done in the context of this thesis. However, this section derives a generalized life-cycle from the concrete implementation in CYCLONE which was done by an involved company[38]. The life-cycle is exemplified in the following four phases:

Phase 1: "Preparation"

In phase 1, the VM developer creates a deployable application module. This module is parameterized, that is, it can use deployment parameters to configure the instance-specifics, for example, the permitted users and groups. These parameters can be also used to configure the integration of the applications with other backend systems, for example, by providing URLs to the logging distribution or the Federation Provider.

Phase 2: Deployment and Configuration

Phase 2 can be initiated by an end-user who instructs the self-service portal to deploy the prepared application module as a new VM on the target cloud. It can also be initiated by a DevOps engineer who interacts with the deployment manager API directly. Both the self-service portal and the DevOps

[38]Details about this can be found in the CYCLONE deliverables D3.1, D6.1, D6.2, and D7.2

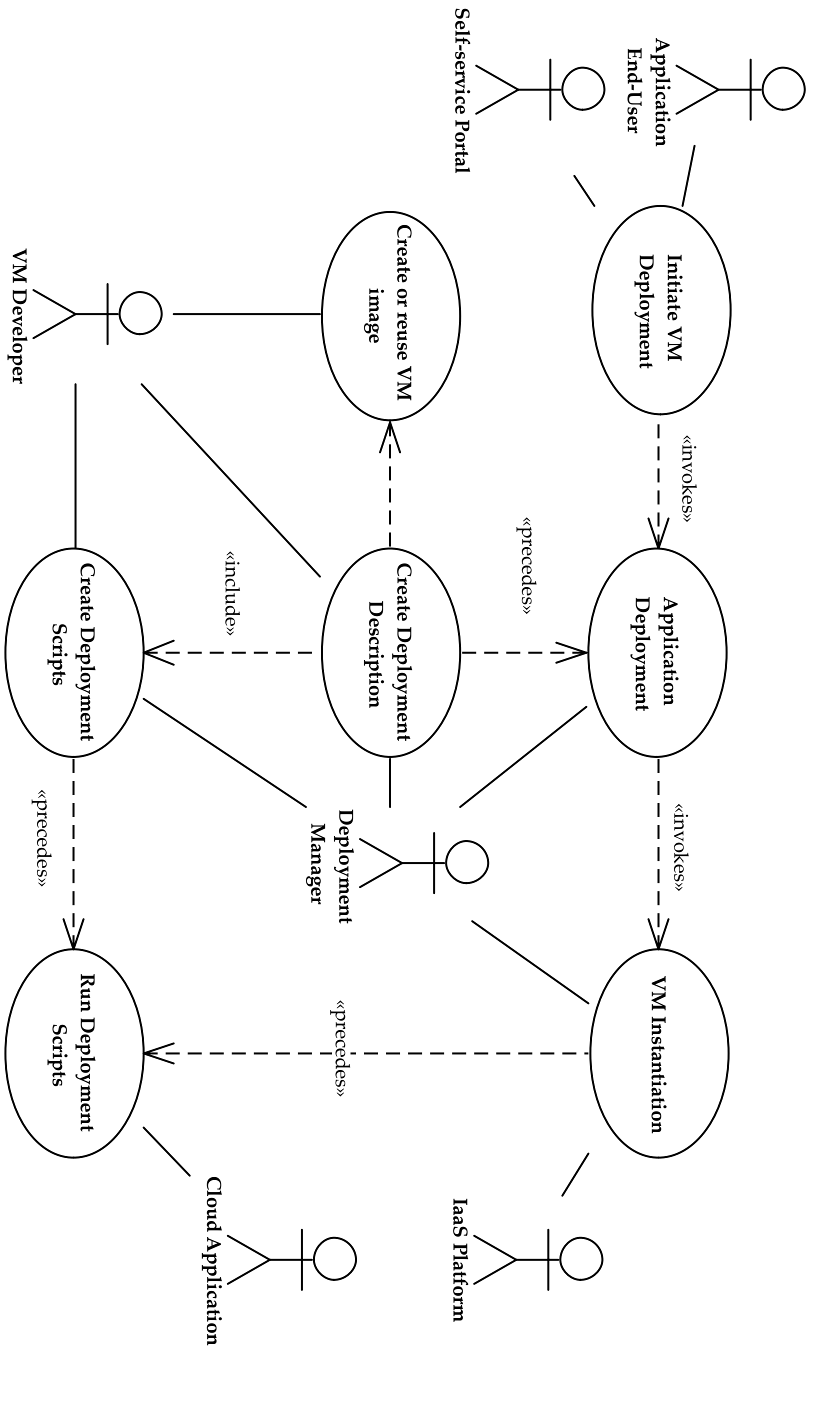


Figure 3.14: Managing multi-cloud deployments

engineer need to provide all required deployment parameters, for example, authorization-related parameters, and the target cloud platform credentials.

Phase 3: Operation, Scaling and Configuration Changes

In this phase, the deployed VM can be used by authorized end users. As long as it is running, the application can be scaled up or down by the deployment manager. Scaling scripts can also be used to update the machine configuration by issuing a scale command without changing the number of VMs (so called `null` scaling).

Phase 4: VM tear-down

Like phase 2, phase 4 can also be initiated by an end-user who instructs the self-service portal to invoke the deployment manager to tear down the VMs or a DevOps engineer interacting directly with the SlipStream API. A best practice is ensuring the complete and thorough removal of any sensitive and personal data on the target cloud when a VM is torn down.

3.4.4 Federated Authorization for Websites and SSH

Figure 3.15 highlights the interplay of actors and use cases that enable federated authorization. As a first step, VM developers have to prepare their VMs to use authorization based on federated user identities, as described in more detail in the next paragraph. The deployment scripts refer to parameters regarding federated authorization, for example, a list of users and groups that should be able to have access to the machines. Those can be either specified by DevOps engineers directly or through the self-service portal.

Figure 3.16 shows the four different methods that the security architecture designates to authorize federated identities:

PAM Module

In multicloud environments, every new cloud introduces new user accounts, increasing the number of passwords that end users must deal with. This overhead can be reduced by Single Sign On (SSO) with federated identities. However, there is no satisfying SSO implementation for Secure Shell Login that can be used web scale. While there are solutions, such as Kerberos+SSH, there are ample challenges when applying it in federated ecosystems.[39].

The challenge of implementing SSO using federated identities is the main motivation of the PAM Module. For academic end-users, employing federated eduGAIN identities for SSH login is the most obvious way of implementing SSO, since each researcher already has an EduGAIN identity, bound to their institutional email address. The PAM module was implemented by Berdonces-Bonelo who provides details in [21].

The PAM module involves two actors, the *DevOps engineer* who initiates the deployment and the *end-user* who requires access to the deployed VMs. In order to highlight the benefit of the PAM module, both workflows are contrasted: *with* and *without* use of the module.

[39]An iteration of such challenges is written down by Thatmann and Zilci in [157]

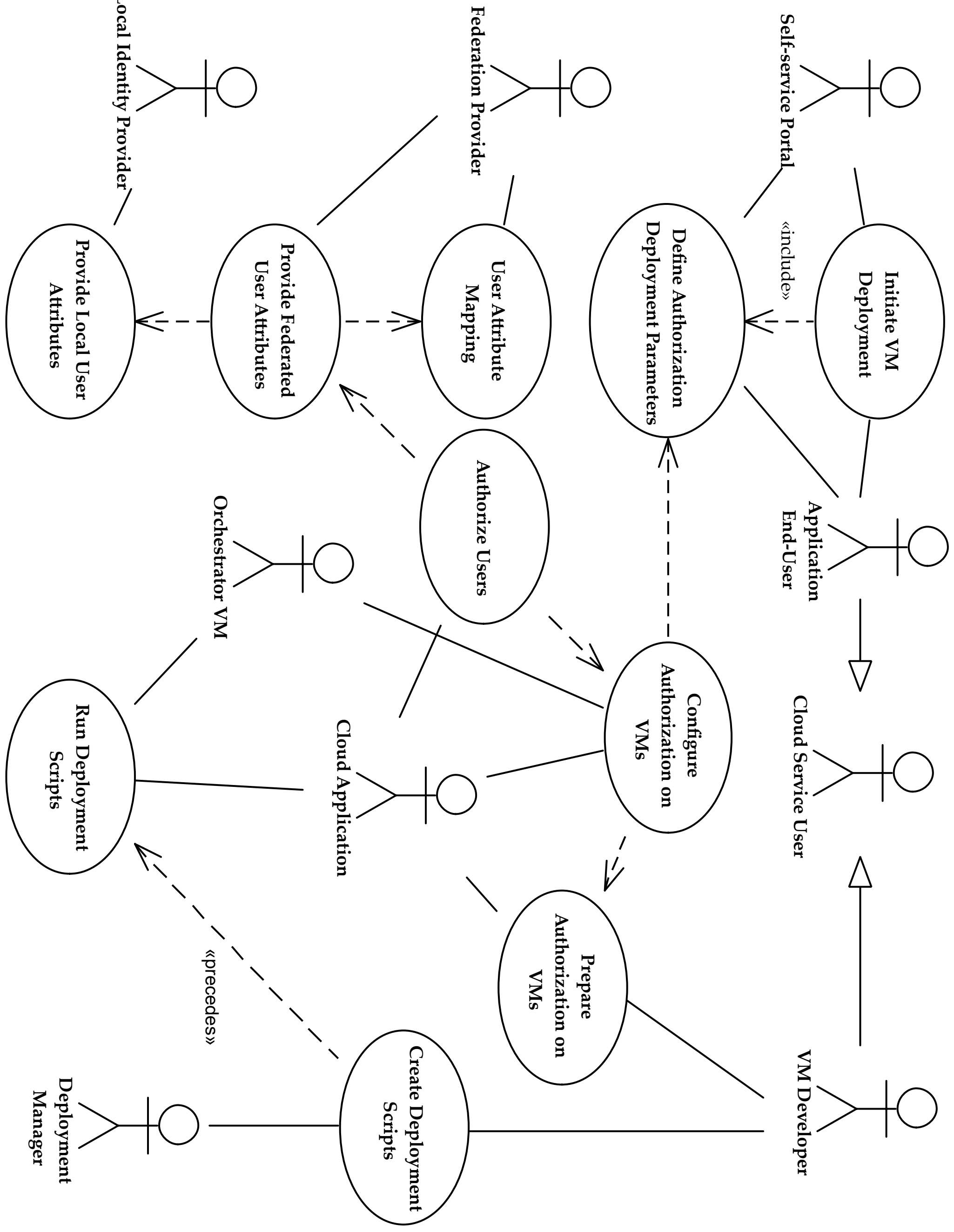


Figure 3.15: Federated Authorization

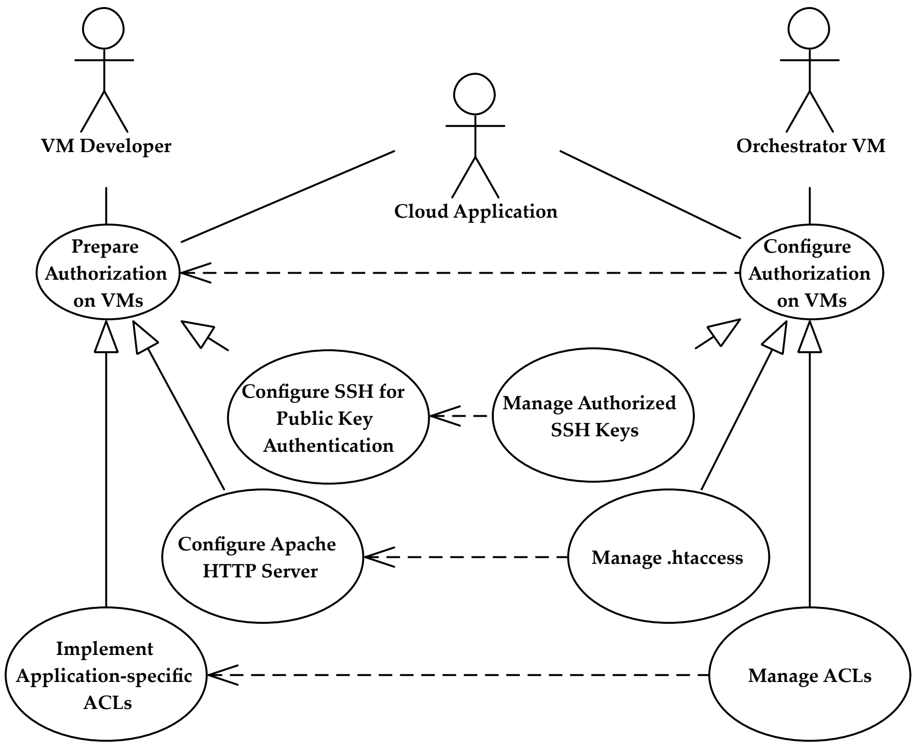

VM Developer
Cloud Application
Orchestrator VM
Prepare Authorization on VMs
Configure Authorization on VMs
Configure SSH for Public Key Authentication
Manage Authorized SSH Keys
Configure Apache HTTP Server
Manage .htaccess
Implement Application-specific ACLs
Manage ACLs

Figure 3.16: Federated Authorization Mechanisms

Conventional approach: Relying on SSH keys or passwords

First, the engineers deploy and instantiate one or multiple images. Afterwards, they log in via SSH to each VM using their public key that they previously deposited on a self-service portal and that was added to the SSH configuration by the deployment manager. After they are logged in to the machine, they add new accounts for all end-users who require access via SSH and add the public SSH keys to the `authorized_users` file. After this is done, those end-users can log in to the VM, for example, for collaboration, debugging problems, and providing support.

The scheme described above poses three main problems:

1. Many end-users prefer not to use public key authentication due to usability issues.
2. Engineers have to know and use the somewhat complex and low-level SSH configuration mechanisms to manage and map end-users to system accounts.
3. In multi-cloud deployments, manual key distribution by engineers is not feasible, especially within clustered deployments of possibly dozens of machines.

The engineers could alternatively configure password authentication for SSH. However, this has pitfalls both in terms of security and usability which makes it no viable alternative option:

1. Each end-user receives new credentials for each VM instance, which are tiresome to manage.
2. If end-users reuse the same credentials on different systems (which is quite common), it would invite security breaches with each additional system sharing the same username/password combination.
3. The engineers must securely share the credentials with the end-users, which is a challenge in itself.
4. The engineer would have to manage all the usernames and passwords of end-users who received access to VMs.

Using the PAM module

The PAM module makes use of the `keyboard-interactive` mode of SSH to interactively communicate with connecting clients. The module starts an embedded web server and transmits its URL through the yet unauthenticated SSH connection. By following the link in a web browser on their machines, they can authenticate via the Federation Provider through the regular OpenID Connect flow. The identity contained in the JWT that is returned to the embedded web server is compared to a file that contains a list of identities that are allowed to login. This list can be modified manually, for example, by engineers, or it can be provided through the application deployment parameters to the deployment manager.

In summary, using the PAM module has the following advantages:

1. The password does not leave the end-user's domain, securing it against exposure to external systems.
2. Engineers can configure rules more easily by modifying a simple text file
3. End-users can authenticate with one click if they already have an authentication session with the Federation Provider, for example, after they logged on to the self-service portal.
4. The file containing the end-user's accounts can be easily distributed by the deployment manager.

Yet, the limitation of requiring the client devices to have a browser and an SSH client installed still persists. However, as this was always the case in the analyzed use case, it can be presumed to be common in others as well.

Apache HTTP Server Authentication & Authorization

The `mod\_auth\_openidc`[40] module provides OpenID Connect authentication for the Apache HTTP Server. When configured, it allows end-users to access protected websites and applications using their federated identity with the Federation Provider. When the module is used, authorization rules are defined using the customary `Require`-Statements[41] as with any other authentication mechanism. The commonly used `.htaccess` file[42] could be used to define conditions on user attribute values that must be met for authorizing users, for example, their name, the membership in institutional groups, or the issuing organization. In effect, this mechanism allows easy AA functionality

[40]https://github.com/pingidentity/mod/_auth/_openidc
[41]https://httpd.apache.org/docs/2.4/en/mod/mod/_authz/_core.html/#require
[42]https://httpd.apache.org/docs/2.4/howto/htaccess.html

for all Apache-hosted applications and websites, for example, PHP and Python solutions such as WordPress. The evaluation section 4.6.1 presents an example configuration to secure WordPress using federated identities.

Application-specific ACLs
Any OpenID Connect-compatible application is free to implement their own ACLs. For example, the login module of the deployment manager could rely on the values of the federated user attributes to enforce deployment ACLs.

3.4.5 Providing Unified Logging

Multi-cloud environments provide challenging environments for logging, auditing, and monitoring. Especially when applications span multiple clouds, there are several requirements regarding these functions:

- **Common mapping for heterogeneous sources.** In multi-cloud environments, log messages can originate from a diverse set of systems and services. Thus, the logging system needs to provide means to map those messages onto a common set of attributes.
- **Flexible deployment for manifold topologies.** Cloud applications can be deployed in a diverse range of topologies that impact the performance of logging services. Logs should be sent to the nearest consumer to keep the logging system performing well. Therefore, the topologies of logging systems need to be flexible enough to be in line with application topologies.
- **Keeping log access control in line with application access control.** Log messages can contain sensitive and possibly personal data which always needs special protection. At best, the access control to the logging system should reflect the same access control that is applied for the service access.

The security architecture incorporates the ELK stack which is especially suited for solving multi-cloud challenges, as it features many characteristics addressing the iterated challenges, especially:

- **Flexible input and filter plugins.** These provide both a comprehensive interface to many of the potential services, for example, a Syslog interface for daemon logs, connectors for common database management systems, as well as generic JSON APIs for application specific log subsystems, for example, Log4j. Flexible processing of these logs using modifying Logstash filters such as the `mutate` filter[43] also provides an easy to use method for mapping heterogeneous data sources onto a common schema.
- **Relaying and aggregation.** As Logstash can upload its logs to another Logstash instance, creating a cascaded log system that relays log messages to upstream log servers is easily implementable using differently configured instances of the same logging distribution. Logstash can also aggregate data so that, for example, multiple data-center-local instances would keep the raw logs and at the same time a company-wide instance would collect statistics over all Logstash instances.

[43]https://www.elastic.co/guide/en/logstash/current/plugins-filters-mutate.html

- **Flexible access control.** The access control mechanisms and rules are quite flexible and can be defined and customized by the operators of the logging dashboard, Kibana. For example, an access control rule could compare the Kibana user's domain with the value of a certain field in the log data representing the domain where the log originated. If they match, access would be granted. Other access control rules comparing tags or user names are equally simple to implement.

There are however simpler approaches for scenarios requiring less functionality, for example, using a remote-capable syslogger, such as rsyslog[44]. This works quite well when applications homogeneously support syslog, run all in the same data center, and when they don't require sophisticated access control mechanisms. Another approach is using systemd's journald together with systemd-journal-remote[45] which supports collecting logs from other systems using either a simple line-based format or JSON. Journald features a compact on-disk representation as well as a good set of tools for interacting with the service. However, as the capabilities of both approaches are limited, they require a lot of effort to support multi-cloud scenarios with the same ease of use as the ELK stack. However, the approaches are complementary as both solutions can also relay their messages to Logstash.

Some alternative solutions focus on the monitoring part, such as Nagios[46], Icinga[47], and Munin[48]. The main purpose of those tools is aggregation of metrics from diverse systems and displaying it in miscellaneous graphs. Most often, SNMP is used to connect to diverse devices, such as routers and printers. However, these tools cannot be used for logging anything else than numeric values and predefined states (e.g., "OK" and "ERROR").

[44]http://www.rsyslog.com/storing-and-forwarding-remote-messages/
[45]https://www.freedesktop.org/software/systemd/man/systemd-journal-remote.html
[46]https://www.nagios.org/
[47]https://www.icinga.com/
[48]http://munin-monitoring.org/

Chapter 4

Evaluation

The following subsections describe the respective evaluation activities that were undertaken to validate the components that were described in the preceding section. Each component was evaluated using a method that should fit its intended benefits. For example, the Cloud Service Registry Architecture was discussed with its prospective users to get feedback on its qualities and practical usefulness as explained in Section 4.1. In contrast, the Trusted Cloud Transfer Protocol and the proxy prototype underwent thorough performance benchmarks that are presented in Section 4.2 and Section 4.3. Section 4.4 provides extensive information how the TRESOR components were deployed in a production environment to gain an understanding of their usefulness to create a secure cloud ecosystem. Furthermore, TCTP and the distributed cloud proxy were used to create an architecture blueprint for end-to-end secured medical SaaS offerings and were in turn also benchmarked in a realistic cloud setting, as carried out in Section 4.5. As a conclusion, Section 4.6 evaluates the security architecture by applying it to secure the CYCLONE use cases.

4.1 Evaluating the Cloud Service Registry Architecture

There were a number of evaluation activities to analyze how well the established cloud service registries and their components meet the stakeholder requirements. These activities can be divided into two categories based on their association: either they were conducted as part of the initial implementation of the TRESOR Use Case 1 (Section 4.1.2 to Section 4.1.4), or as part of further use cases (Section 4.1.5 to Section 4.1.7).

The aim of the selection of activities and focus groups was to gather a variety of feedback, to evaluate a diverse set of aspects, and to apply both qualitative and quantitative methods. After outlining the motives and purposes of these evaluation activities, the following subsections provide further details. This section is concluded by deriving follow-up questions from the evaluation results.

It should be noted that no undertaking applied the description language to services that are non-existing, as often observed in the field of Semantic Web Services. This is based on the presumption that using imaginary services for exemplary purposes does not serve the intended target of business pertinence well. Instead, the inability to apply a property to describe a service can hint at a potentially superfluous vocabulary element. That is one of the reasons to take 27 prominent cloud services and use the SDL-NG to describe them, for example, Google Drive, Salesforce Sales Cloud, and Amazon EC2. It can be observed that there were no challenges in applying the business vocabulary to create a meaningful service description. As a result, vocabulary can be considered to be well suited for applying it to describe prominent cloud services. All service descriptions can be found on GitHub[1].

At last, works within the related research area of information retrieval systems often include an assessment of the effectiveness of a specific approach using statistical methods. For example, in [22] Bergamaschi et al. solve keyword queries over relational databases and analyze the effectiveness of the proposed algorithm using precision and recall, as these are common evaluation measures for keyword-based search engines. However, there are large differences between such search engines and the service registry presented in this thesis: first of all, service registry queries are not expressed using self-determined free-text keywords. Instead, they employ a fixed structure that is predetermined by the business vocabulary. This structure consists of constraints on the permissible value of a service property, for example, it constrains the value of the `payment_option` property to `credit_card`. Furthermore, the data format of the service descriptions persisted in the registry directly corresponds to the query format utilized by the users. At the end, "service retrieval" boils down to transforming each constraint to a corresponding database selector and using it to filter the set of services persisted in the database.[2] These database selectors effectively implement the user query and guarantee that only matching services are returned and no service is missed, rendering the measures of precision and recall meaningless for evaluating the service registry.

4.1.1 Motives and Purpose

The TRESOR focus group evaluation (Section 4.1.2) allowed a better understanding of the importance of the properties for the service selection, the ideal criteria of a service registry, as well as the differences between the initial ideas and the views of the stakeholders. The focus group consisted of experts from the health sector to strengthen the applicability of the resulting registry for subsequent adoption in this area. The discussion with two lead authors of another service description language permitted learning from their past findings, contrast the different concepts of service description languages, and get design recommendations for the TRESOR broker and the SDL-NG (Section 4.1.3). A further strengthening of the work's evaluation was the participation in the "AG Standards" focus group which comprised researchers from other projects of the "Trusted Cloud" research program who were working on approaches related

[1]https://github.com/TU-Berlin-SNET/sdl-ng/tree/master/examples/services

[2]In the case of the example, the "`payment_option:$in:['credit_card]`" MongoDB query selector

to service description and brokering. Section 4.1.4 presents the received feedback on the SDL-NG and its perceived advancements in relation to previous approaches.

After concluding the work on the prime Use Case 1 and the associated TRESOR project, the evaluation of the other use cases continued. Section 4.1.5 details a quantitative evaluation with potential users of the Cloud Storage Broker (Use Case 2) through an on-line survey to gain insights into the concrete service selection process of typical cloud storage consumers. As such quantitative evaluation is limited in its depth, there was an additional topic-focused expert interview which is abridged and interpreted in Section 4.1.6. It provides in-depth knowledge about the concrete IaaS service selection process at a commercial IT provider pertaining to Use Case 4. To better contrast both types of evaluation and to cover commercial and private perspectives, the evaluation is concluded with five face-to-face interviews with both students and software developers, which were conducted in Use Case 6, in Section 4.1.7.

4.1.2 Evaluation Activities in the TRESOR Focus Group

The TRESOR focus group consisted of four IT specialists from the health care sector who were partners in the TRESOR research project: the health center CIO and three IT professionals from the same institution.

Method and Objectives

To evaluate the business vocabulary and the service registry approach, a two-hour meeting in the final phase of the project was scheduled. Participants were three researchers from the SNET chair of TU Berlin, including me, and the whole focus group. First, there was a presentation about the TRESOR broker and the business vocabulary in order to ensure the mutual understanding of the evaluation subject. Afterwards, interviews were conducted that went through preconceived questions. Another regular evaluation activity was discussing the general approach with the focus group at the project consortium meetings.

As the business vocabulary was developed on the basis of regular project workshop results with the two health centers, it was quite obvious that they could determine best if it fits their needs. However, as we had worked together for quite some time, their neutrality in answering the questions could be disputed. This was compensated by including the health center CIO in the evaluation who was only seldom involved in the project and took a leading role in answering our questions. Furthermore, the questions were formulated such as "How important is criteria X for your typical service selection?", making it meaningless for them to favorably rate the importance higher or lower.

Results and Analysis

All in all, there were no major concerns regarding the general concept of the TRESOR service broker and the business vocabulary. The focus group defined three main characteristics of an expedient service registry:

1. It should reflect their selection criteria
2. The descriptions should be easily understandable
3. The service differences should be made clear

The first characteristic was evaluated by going through the whole business vocabulary and asking the health professionals to come up with a mutual importance rating of each individual criteria on a 5-step scale from "indispensable" (1) to "irrelevant" (5). They were also asked for further feedback on each property in order to get ideas for an eventual refinement of the vocabulary. The pie chart in Figure 4.1 groups the service properties by their rated importance: 86.5% of the 52 criteria were rated important and higher, while only 13.5% were rated irrelevant or less important. These results show that the generic business vocabulary captures some of the most important selection criteria of those users.

To address the second point, the group provided detailed feedback on how to present each of the properties to the users to make the service descriptions easily understandable. The most frequent remark was that the system should explain the consequences for the service selection instead of just factually describing the properties. Regarding off-line capabilities of cloud services, for example, it is more advisable to illustrate when and why this functionality is needed and what consequences missing capabilities could have instead of potentially simply defining what the term represents. All recommendations will be considered as part of the ongoing efforts to implement the Open Service Compendium. Regarding the third point, making the differences clear was covered later by the comparison view of the Cloud Storage Broker.

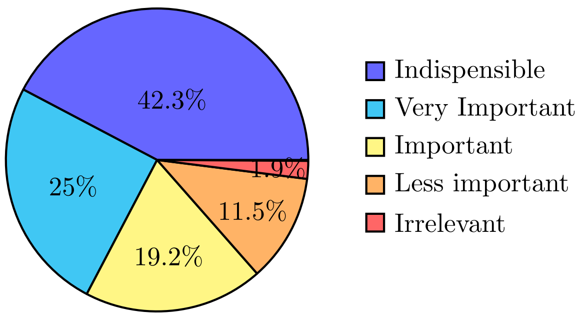


Figure 4.1: "Percentage of Selection Criteria by Importance"

There are two major topics where discussions with the focus group, especially the health care sector specialists, lead to changes in the approach from the initial ideas to their subsequent implementation. The following subsections present both topics and explain their impact on the work.

Topic 1: Reasoning about compliance

Initial conceptions of the description language included compliance information, so that in the best case a software component could reason about the suitability of a service from legal and compliance perspectives. However, this was deemed highly unrealistic by the focus group: Even if an algorithm would come up with a fitness decision, the health center would ultimately be liable so it would need to double-check the results. However, as health centers face high cost pressures they do not have the time, resources, and knowledge to check whether their provider's software fulfills all relevant constraints. Therefore, they have to rely on the provider or, in the case of TRESOR, the marketplace operator to ensure that only compliant services are offered on the marketplace. This is covered by insurances and legal provisions, so there is in practice no incentive for providers to actively become rouge and circumvent those constraints.

Second, due to the large extent of legal constraints and service aspects to consider, checking compliance is economically more sensible to do manually by a trained specialist instead of implementing an algorithm for this: both the ontology to describe services and the service descriptions themselves would have to be extremely detailed to cover all relevant aspects. Furthermore, many assessments are highly subjective and therefore unable to be implemented by a computer, at least with the technology currently available.

Topic 2: Role of Service-level Agreements

At the beginning of the project, the inclusion of an SLA formalization into the description language was envisioned. Yet, based on the discussions with the focus group, this was not followed through. First, the previous experience of the health centers showed that SLA assurances for non-critical services, such as those addressed by TRESOR, don't have a meaningful role. On the one hand it was always difficult to pinpoint whose SLA was violated, as there are always many systems involved in the service delivery. On the other hand, most SLAs are not formulated precise enough to really hold the service provider accountable. This is also supported by research: in [76] Hogben and Pannetrat show that for the same service state history (available/unavailable) the SLA definitions of Amazon EC2, Microsoft Azure, and others can lead to 0% reported availability for one provider and 100% for another. Therefore, unless SLA definitions are standardized between all cloud providers, there is low utility value in any single SLA. However, the health centers stated that information about past service failures would give them a far better estimation of expected service quality, even if inferring future results based on past measures is always associated with certain inaccuracies.

4.1.3 Discussing USDL with Leidig and Oberle

The goal of this evaluation was to present the project requirements and get design recommendations from Leidig and Oberle, two experts on service registries and the lead authors of USDL [117].

Method and Objectives

At the beginning of TRESOR, there was a telephone call with the Trusted Cloud accompanying research leader as well as Leidig and Oberle. At the time of the call, there was first an overview about TRESOR before the two experts referred about the history, the current state, and the future plans for USDL. After this, a discussion ensued about some aspects communicated to them before the meeting:

- How can USDL be applied in general and for specific project challenges?
- How will they address current issues in the implementation?
- Do they have any other remarks about the research?

Results and Analysis

Both researchers recommended to rely on Linked-USDL, the successor of USDL, as the main description language and create or reuse domain specific ontologies for all TRESOR project challenges that are not yet covered by it. They explained why USDL would not be developed further and that it also will not receive any fixes for existing implementation issues. Then, more aspects of the research were discussed on a high level without arriving at concrete conclusions.

Later in the project, an in-depth requirements analysis revealed the shortcomings of related approaches, especially the lack of business pertinence and tooling simplicity in popular works from the field of semantic web services, including the Linked-USDL and related ontologies.

4.1.4 Trusted Cloud "AG Standards" Focus Group Discussion

The "AG Standards" focus group consisted of participants from all research projects of the "Trusted Cloud" research program. After getting together multiple times over the course of the funding period, there was a closing full-day workshop in May 2014 which was focused on approaches to cloud service assessment and description.

Method and Objectives

After the focus group leader summarized the challenge area, the user-centric cloud service registry approach of this thesis was presented before other participants presented their proposals as well. A subsequent lengthy and deep discussion uncovered meaningful directions of future research.

Results and Analysis

The focus group expects the business vocabulary and the description language to be a potential basis for a common cloud offering description language in the future, making cloud offerings portable over multiple marketplaces. However, some partners objected that cloud services are not "traded" on marketplaces but are directly offered through the Internet. In this context, a potential language for tradable services would be redundant. At last, it became clear that most participants had previous experience with the USDL [117], comparing it to the results of this thesis.

In summary, three areas were identified where the presented service architecture is seen as an advancement to previous works:

Concreteness. A major advancement of the approach is limiting the descriptions to real-world services using properties pertinent to concrete users. This is in contrast to describing synthetic services with "abstract" properties deemed to be pertinent to non-existing users, as often observed by other participants in the related work. This is in line with a remark at a former focus group meeting that most of the existing USDL descriptions are not usable for any concrete use case, as they do not model any existing business service and none of the projects saw their requirements covered by the vocabulary.

External data retrieval. The need for a specialized USDL editor was presumed to be futile, as most of the service description should in principle be retrieved from external data sources, for example, the textual description of the service should just be scraped from its website. This idea was taken up by making easy scraping of external resources one of the main concerns for the description language.

Use of a textual DSL. Based on their practical experience in using a Ruby DSL for other purposes, some researchers of the focus group supported the view that using a textual DSL can achieve a higher degree of simplicity and adaptability when using it for describing services.

4.1.5 Cloud Storage Vocabulary Questionnaire

In [87], an intermediate version of the Cloud Storage Vocabulary was evaluated using an on-line questionnaire. Its "main goal was to identify the relevance, usability, and suitability of the design approach in daily life" [87, p.28].

Method and Objectives

After refining the question structure, the survey was sent to around fifty students, a university institute with roughly twenty employees, and some small local businesses. The survey first asked about the respondents' experiences and intended use of storage services. Then, the participants had to rate the importance of 27 criteria for their selection of a cloud storage service on a five point scale from irrelevant up to indispensable.

Results and Analysis

The results were "very sparse and diverse"[87, p.29] as there was only a small number of completions. Of 35 respondents that began the questionnaire, 18 (51.4%) completed it.

However, the survey provided first valuable insight into the vocabulary usefulness for generic Internet users, as most other people previously involved in the research are either professionals or academics. Figure 4.2 shows the average importance distribution of all criteria, which is grouped into $1 - 1.5$ (indispensable), $1.5 - 2.5$ (very important), and $2.5 - 3.5$ (important).

In general, security properties were almost always deemed as highly important. The other results promise a high relevance of the vocabulary for Internet users, yet the absence of less important and irrelevant criteria could also point at the inability of the respondents to differentiate the importance of their criteria. The low completion rate could also imply that people either could not understand the criteria or did not know their cloud provider selection process.

4.1.6 Open Cloud Computing Map: Expert Interview and Findings

The goal of the semi-structured expert interview was to identify the requirements of an exemplary customer and to compare it with the OCCM concept and implementation. The following paragraphs abridge the expert interview explanation [17, p.37-38].

Method and Objectives

The interview was conducted with the data protection official and software architect at a technology provider for business analytics and marketing automation. The expert had worked for 17 years in the IT industry and had ten years

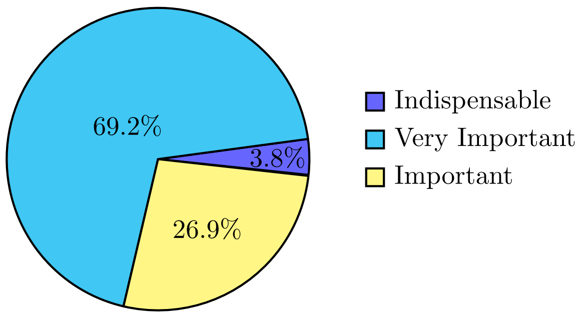


Figure 4.2: "Average Importance of Selection Criteria"

experience in evaluating and choosing infrastructure solutions, such as cloud infrastructure, storage, and platform services. His target companies expect very high availability, performance, and data privacy.

Results and Analysis

The survey was structured around three key questions that left room for discussion and follow up questions:

Question 1: How are cloud service selection processes performed? The cloud service selection process starts with a large amount of research work, the comparison of basic features and different offers, and the consideration of other service consumers' experiences. This allows obtaining an overview about the strengths and weaknesses regarding particular requirements. At a more advanced stage, providers are contacted for further information about how security and performance requirements are realized and guaranteed. The release of this information is often bound to a non-disclosure agreement.

Question 2: Which service characteristics are considered for service selection, and are data center locations crucial in this consideration? Before a cloud service is selected, it is evaluated using concrete tests, for example, parts of the software that should be moved to the cloud are deployed to the target environment. The expert stated that data center locations are an important factor in the selection of a service: first of all, privacy laws often differ between the customers' and providers' countries. Second, seismographic and political circumstances have to be also considered. At last, connections to large Internet exchanges need to be ensured.

From a legal point of view, the headquarters and subsidiary locations also play a major role, due to their jurisdictions. As an example, US companies can use their EU subsidiaries to offer contracts in the same jurisdiction as their EU customers. When companies offer this, the position of data centers is considered during the whole selection process.

Question 3: Can a selection process be improved by a tool and, if so, how? A software solution would reduce the research efforts greatly since a lot of the considered information is available on-line, yet it has to be gathered again for each selection process, due to the dynamic nature of the cloud market. In contrast, a solution such as the OCCM, providing comprehensive information and comparison facilities, should support different cloud consumers, each having different priorities and views on cloud services.

Summary The expert confirmed the main assumption about the expected usefulness of the cloud service registry approach. However, he also pointed out that the registry data will not be the only basis for selecting services. This leads to a future research question: which specific service selection processes can be supported by the registry and potential additions, for example, automated deployment of cloud components for evaluating service offers?

4.1.7 Open Service Compendium Face-to-Face Questionnaire

Five face-to-face interviews were conducted to question potential OSC users regarding their opinion on the Open Service Compendium as well as the extensions of Use Case 6. The participants were a bachelor and a master student of computer science, a master student of technical computer science, as well as a junior and a senior software developer.

Method and Objectives

At the beginning of each interview, the Open Service Compendium was presented as well as a brief introduction of its components. Afterwards, the interviewees tried out the OSC without assistance, allowing them to gain personal user experience. The following subsection abridges the explanation of these interviews [5, p.29-31].

Results and Analysis

All participants were asked three main questions and surveyed about their general remarks on the OSC.

Question 1: How well did the different components of the OSC system support you in selecting cloud services? Possible answers were: *Bad*, *Acceptable*, *Good*, and *Very good*. The majority of interviewees evaluated the support by the OSC as *good*, one person considered it as *very good*. Overall, they indicated a good user experience as well. However, all interviewed persons had complaints about insufficient data as the overall low number of services in the registry hindered their ability to test the interface using different properties.

Question 2: How helpful were both types of questionnaires? Possible answers were: *Not helpful*, *Could be better*, *Acceptable*, and *Very helpful*. One interviewee indicated that the questionnaire *could be better*: in several cases the *static* questionnaire provided questions that, regardless of the answer, selected no services at all, due to the limited registry data. For example, among all cloud services only two services provide a free trial. Due to these limitations, the *dynamic* questionnaire sometimes cannot reliably create alternate questions as the available data is not diverse enough. At last, the types of questions should be extended to also cover further data types, for example, numerical values, such as the number of service add-ons. Another interviewee considered the questionnaire *acceptable*: the questions are quite general, and the step-by-step process through the questions could potentially be time-consuming.

On the other hand, the remaining three persons considered the questionnaire *very helpful*. Those with limited knowledge about cloud services pointed out the ease of use and the questionnaire perspicuity, which helped them find cloud services matching their needs. Persons with good IT background were interested in knowing how the dynamic questionnaire works and agreed that the applied concepts of generating the questions are logical and made sense to them. One interviewee pointed out that he presumes the questionnaire to be easier to use by people with limited IT background; however, IT experts would certainly prefer filtering the list of services directly, instead of going through a questionnaire.

Question 3: What do you think of the filter options? Possible options are: *Overloaded/illegible*, *Could be less*, *Various*, and *I could filter according to all my requirements (fulfilling)*. In line with the last remark to Question 2, one interviewee with limited IT knowledge criticized the filtering as being *overloaded* with options that are unimportant and excessively in-depth for *him* as a normal user. Contrary, all other interviewees evaluated it as *various* or *fulfilling*. From the latter group, some interviewees considered filters easier to use and straight-forward as they include more specific properties than the questionnaires. Compared to other selection helpers, they also praised that the filtered services are updated immediately when selecting options, instead of requiring them to manually update the list of results. The remaining interviewees considered the filters helpful, especially that they can be reached in two ways: either as the result of the questionnaires, as well as directly from the OSC homepage.

In general, the interviews indicated a potentially highly usable system. All interviewees were willing to use it and emphasized the importance of providing more data than currently available.

Moreover, they provided three main suggestions:

1. **Service ratings.** After going through the questionnaire and filtering the services, a rating of each filtered service would support their decision-making.

2. **User entries.** The dynamic questionnaire could be extended to consider the most often answered questions when deciding which to ask the users.

3. **Data standardization and providers' data support.** To improve the system usability, all services should have a complete description with no unset property value for any service. A possible solution would be to convince the cloud providers to keep the descriptions up-to-date.

Although the number of interviewed persons is insufficient for a complete validation, the proposed approaches were seen as useful and able to address the challenges of this small group of people. Moreover, the diversity in both the professional backgrounds and the proficiency of the interviewees provided a notable insight into the usability of the approach for its potential user groups.

4.1.8 Interpretation of the Evaluation Results Regarding the Main Requirements

The previous Section 3.1.9 presents conceptual considerations how the registry architecture should meet the stakeholders' requirements. The following paragraphs extend this explanation by adopting the viewpoint of the prospective users, relating the evaluation results to their own requirements.

Requirement 1: Business Pertinence

In summary, the evaluation activities indicate that the work is mostly pertinent to the needs of businesses: The TRESOR focus group concluded that only a low 13.5% of vocabulary properties are not pertinent to their needs. On the one hand this shows the predominant pertinence of the vocabulary. On the other hand it reveals an area for future improvement: removing properties that are not important to the users. For use cases involving regular Internet users, a high relevance of the vocabulary can be also seen, as highlighted by the cloud storage questionnaire results: all properties were regarded as *indispensable*, *very important*, and *important*. At last, the "AG Standards" also applauded the concreteness of the approach: that real business services are described instead of purely abstract ones as often observed in the related work.

Requirement 2: Tooling simplicity and adaptability

The discussion with the "AG Standards" reassured that Ruby is an optimal basis for achieving tooling simplicity and adaptability of the SDL-NG. The Use Case 6 interviewees also regarded the user experience as good. Another insight of the interviews was that making the interface adapt to the different cloud computing experience levels of the prospective users should be considered, for example, hiding detailed properties that regular users infrequently use in order to prevent "information overload" of novice cloud users. On the other side, then it should be made sure that "power users" are not deprived of detailed interactions. At last, none of the use cases that were built with the service registry components showed any challenges in adapting it for specific needs.

Requirement 3: Versatile data retrieval

The need for external data retrieval was notably highlighted by the "AG Standards" group, as most of the descriptions should not be manually authored. An important issue was highlighted by the expert interviewed in [87]: the dynamic nature of the cloud market requires that at each imminent selection decision, service information has to be gathered again. This would mean that there should be an automated process to update the registry data, for example, through either automated data retrieval or establishing a manual update workflow with the providers.

Requirements 4 and 5: Modeling capabilities and service matchmaking functions

There is already a basic variants model implemented which is used in the cloud storage description. However, it was neither evaluated with more complex descriptions, nor compared to other modeling approaches for service variants. At last, cost calculation and service matchmaking were not evaluated, as the CYCLONE IaaS Registry (Use Case 3) and the integrated CP matchmaker within are not ready for deployment.

4.1.9 Discussion and Follow-up Questions

While there was quite positive feedback on the pertinence, simplicity, adaptability, and versatility of the approach, the scope and depth of the evaluation is limited. Until now, there was no large-scale evaluation to verify the findings in a broader scope. However, the implementation of the six use cases hints at a general usefulness for the challenges at hand. At the end, there are two main follow-up questions that surfaced as a result of the evaluation:

1. Which additional activities can be best supported by the registry?

   At the moment, the main functions of the registry are finding and comparing services through browsing. However, there are multiple additional activities that could also be supported, for example, benchmarking, rating, deploying, and functionally testing services. To answer these questions, those activities need to be collected and analyzed with regard to the impact of their implementation to the registry.

2. How to fill the registry with many services?

   The main hindrance for fully leveraging the service registry is the low number of services and the sparseness of property values. While, in theory, every Internet user is able to author service descriptions to fill this gap, in practice, there are two challenges: first, the adoption of the OSC does not happen automatically. In fact, there is a large effort required to attract OSC visitors, e.g., defining a clear content and marketing strategy, investing in advertisement, connecting with business partners, search-engine optimization, and more.

   At the moment, there are not enough resources to undertake these efforts. Secondly, typical cloud service consumers and providers are not motivated by altruism alone: there needs to be a clear value proposition for them to commit resources to the OSC. This could be, for example, financial compensations for providing service descriptions or generating referrals of potential users to cloud providers. All of this needs to be worked out if the OSC and any other use case should be successfully introduced to the general cloud market.

4.2 Performance Testing the TCTP Rack Middleware

This section evaluates the performance characteristics of TCTP on the basis of various TCTP implementations and provides an outlook on the further evolution of TCTP.

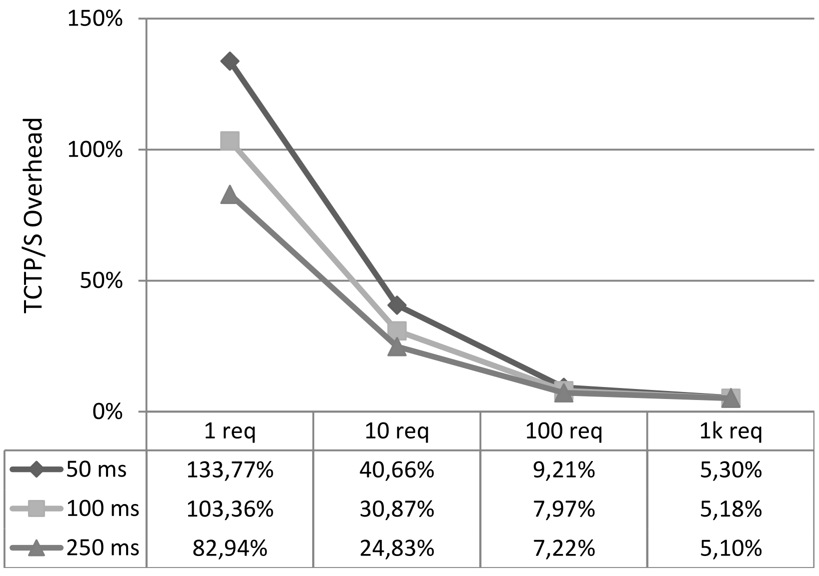
Performance Evaluation

The middleware is accessed by a custom script using the same HALEC implementation as the middleware in order to evaluate the performance overhead of TCTP. The script accesses resources varying between 1kb and 10kb in size, as those represent the average HTML transfer characteristics [163]. The HTTPS request/response processing times are compared to TCTP over HTTPS (denoted as TCTP/S in the table), averaged over 20 test runs. The benchmark client is an Intel Core i7-3520M Laptop running Windows 8.1. The average processing times and the respective TCTP/S overhead are shown in Table 4.1.

Table 4.1: TCTP/S overhead in comparison to HTTP/S

Req. Size	HTTPS	TCTP/S	Overhead
1 kB	35.49 ms	37.14 ms	4.63%
2.5 kB	34.41 ms	36.11 ms	4.94%
5 kB	35.12 ms	35.65 ms	1.50%
7.5 kB	33.81 ms	37.65 ms	11.38%
10 kB	34.72 ms	35.45 ms	2.08%

Table 4.1 shows only fixed processing times for TCTP encryption and decryption, and does not include handshake and network round-trips. To assess the relative overhead of TCTP, the additional round-trips for the TLS and TCTP handshake have to be considered, as well as the handshake processing. In the benchmark setup, the average TCTP handshake took 129 ms, while the average TLS handshake took 23 ms. At last, network latency has also be taken into account when assessing the relative overhead of an end-to-end entity-body encryption. Figure 4.3 shows the resulting relative overhead of TCTP/S with respect to different network latencies and a varying number of requests.



	1 req	10 req	100 req	1k req
◆ 50 ms	133,77%	40,66%	9,21%	5,30%
■ 100 ms	103,36%	30,87%	7,97%	5,18%
▲ 250 ms	82,94%	24,83%	7,22%	5,10%

Figure 4.3: TCTP Relative Overhead

The TCTP overhead adds up to a considerable 133,77% when requesting only one resource over a fast connection. As the number of requests made through an established TCTP/S connection rises, the relative overhead approaches the average processing overhead of 4.86%, which is not substantial. Especially the non-linear increase of the processing overhead suggests improvement opportunities within the prototype implementation.

Further Development

Next steps for further development are TCTP *user agents*, e.g., common browsers and HTTP libraries, TCTP *origin servers*, and TCTP *application frameworks*. Furthermore, *intermediaries* could also be extended by TCTP functionality. Such intermediaries could securely and transparently bridge clients from a protected company network to TCTP enabled cloud services with the additional benefit of not having to adjust any user agent on the company network:

- **Streaming optimization**

  TCTP processing overhead could be reduced by aligning the fragmentation introduced by the HTTP `chunked` transfer coding to the size of TLSPlaintext blocks, so that they are not separated into two or more HTTP chunks.

- **Peer certificate validation**

  Besides securely validating peer certificates, as presented by Georgiev, et al. [63], TCTP implementations should issue a warning, if both the HTTPS connection and the TCTP entity-body encryption use the same certificate. Cloud computing intermediaries have access to the private key of the HTTP/TLS certificate and therefore could also access the plaintext of the entity-bodies secured by TCTP.

- **TCTP discovery circumvention**

  There are some measures to mitigate the circumvention of TCTP discovery: Pre-seeded discovery information, comparable to [45], DNS TXT records containing the discovery information or their cryptographic hash, and using historic data to detect important changes of TCTP discovery information, e.g., an origin server suddenly ceasing to offer TCTP discovery information.

- **Support for HTTP/2.0**

  The next version of HTTP, HTTP/2.0, is a notable transport protocol for future RESTful Cloud Computing services. It requires the use of TLS, but does not consider any encryption for entity-bodies specifically. As the semantics of the `Content-Encoding` entity-header prevail, TCTP is equally applicable to this protocol. How TCTP would fit conceptually here is unclear and needs to be examined carefully.

4.3 Performance and Integration Efforts of the Proxy Prototype

This section evaluates the performance characteristics, the integration efforts, and the lessons learned of the early proxy prototype whose architecture was presented in Section 3.3.1. As the Grizzly-based prototype was the very first component to be implemented, the evaluation was limited to giving a rough estimation if it would be feasible to implement the final proxy according to the envisioned architecture.

Proxy performance characteristics
To analyze the performance of the proxy, a very simple Ruby on Rails web application is accessed through the proxy using the Apache JMeter [8] load test tool. The application is deployed on a Linux server (Debian 6.0) with an Intel Core i7 930 CPU, and 24 GByte of RAM. JMeter runs on an HP EliteBook 8440p notebook PC (Core i7 620M, 8GByte RAM).

In order to evaluate the performance overhead of the proxy, the application throughput (requests per minute) and the client CPU usage are measured for both direct communication as well as through the proxy. The number of JMeter threads are varied to simulate different parallel workloads. To cut out network impact, the proxy is running on the machine used to access the service. At last, SSL is deactivated to exclude encryption overhead.

The results are illustrated in Figure 4.4:

- The proxy impacts the application throughput 9% at most
- The server CPU starts to saturate when using 50 parallel threads
- The overall application throughput does not increase significantly if the number of threads is increased

Summarizing: the proxy prototype highlights that the chosen technology as well as the architecture does not impact the overall performance of the proxy in a substantial way.

Integration effort
A sample Ruby on Rails application was modified to use the relayed identity of a service user to analyze how the proxy authentication could be integrated into existing cloud services. The evaluation shows that it is very easy to modify such a contemporary RESTful web application to use the supplied proxy authentication information. If this holds true for other web frameworks, this mechanism could therefore lead to reduced implementation efforts for proxy-compatible applications.

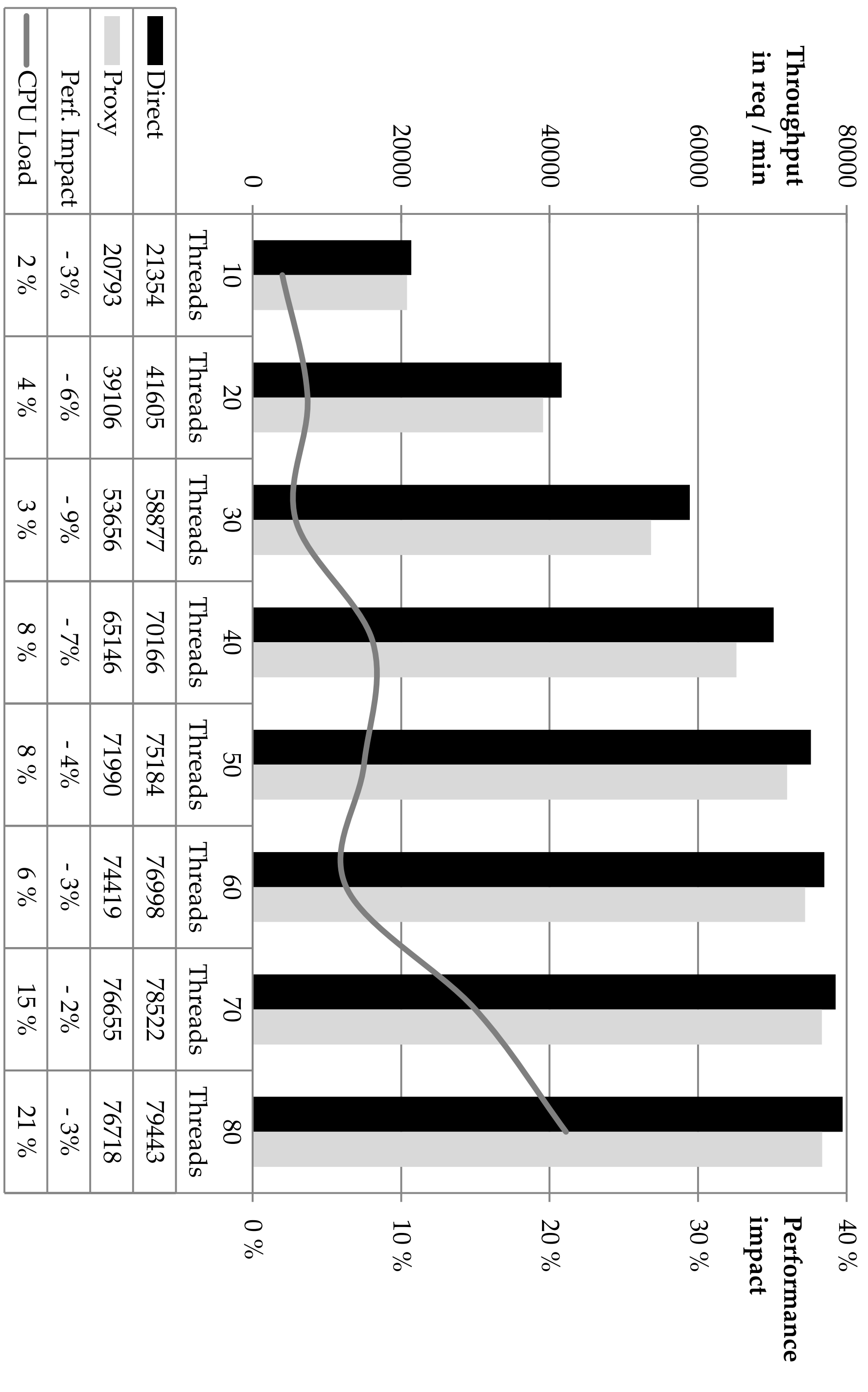
4.4 Deploying the TRESOR Components to Production

This section provides details about the production deployment of TRESOR to relate the development of the service registry and the cloud proxy to all other ecosystem components, especially in their functional integration. Figure 4.5 provides an overview about all components and their interactions in the production deployment of TRESOR. The components for whose development SNET was responsible are highlighted in red.

The production deployment included VMs that were hosted in a private VMWare vSphere cloud at TU Berlin. The different machines are listed in Table 4.2. In general, there are three distinct testbed responsibility areas:

1. **Authentication and Marketplace**

   There were a number of Windows Server 2012 VMs maintained by T-Systems MMS which are related to the TRESOR Federation Provider and Marketplace. These include the `fp-mp` VM which hosts both the Federation Provider as well as the Marketplace. Additionally,



	10 Threads	20 Threads	30 Threads	40 Threads	50 Threads	60 Threads	70 Threads	80 Threads
■ Direct	21354	41605	58877	70166	75184	76998	78522	79443
▨ Proxy	20793	39106	53656	65146	71990	74419	76655	76718
Perf. Impact	- 3%	- 6%	- 9%	- 7%	- 4%	- 3%	- 2%	- 3%
— CPU Load	2 %	4 %	3 %	8 %	8 %	6 %	15 %	21 %

Figure 4.4: Performance impact of the TRESOR proxy prototype

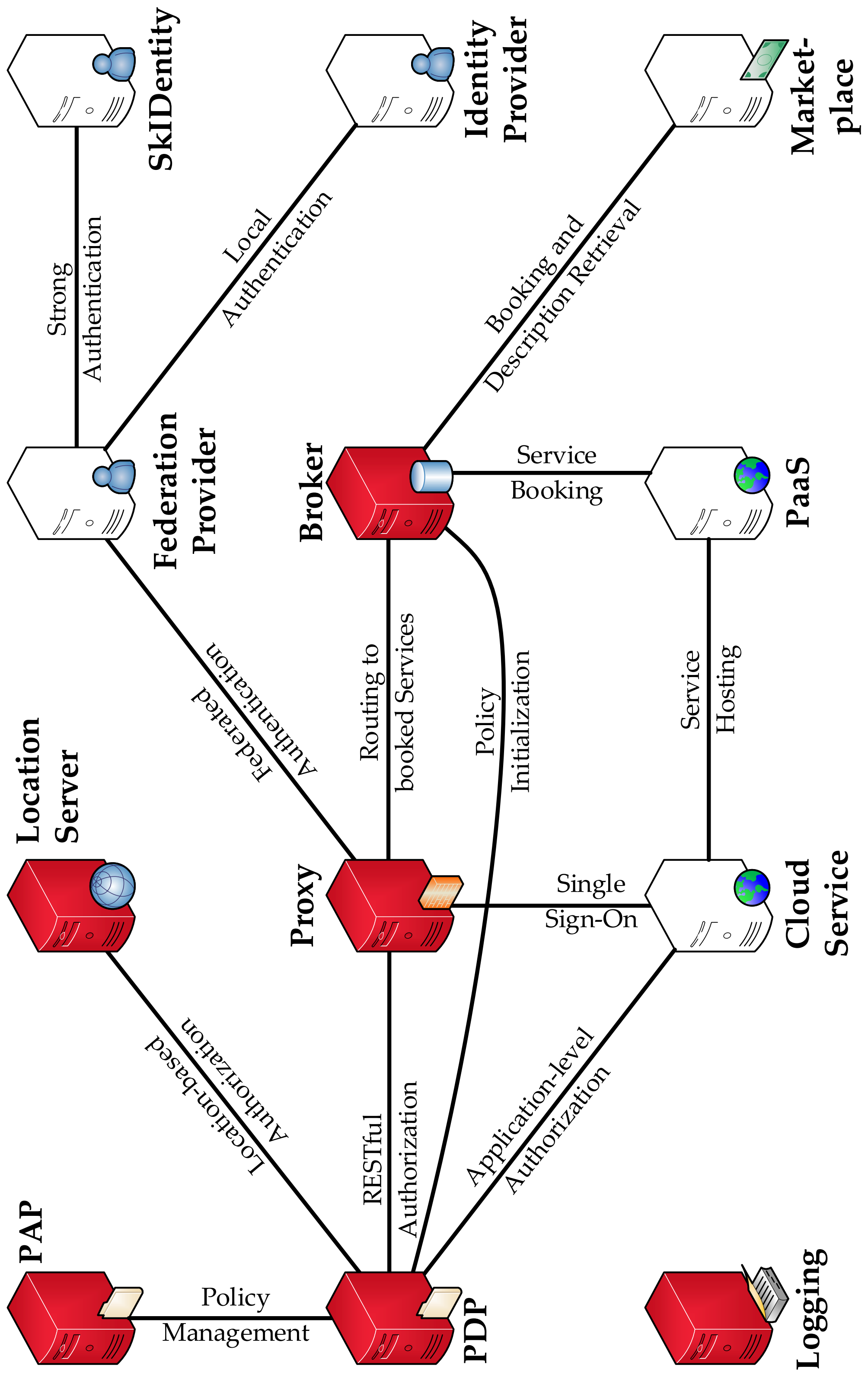


Figure 4.5: TRESOR Components and Integrations in Production Deployment

there are two combinations of ActiveDirectory server and Test Client VM for each of the involved hospitals, `ad-herz/client-herz` and `ad-pauline/client-pauline`. Using these VMs, developers could test how the systems would interact when accessed through a hypothetical client PC that would have been part of a company ActiveDirectory. These machines were also beneficial for project presentations. At last, there was the `ad` machine which hosted an ActiveDirectory for the `fp-mp` Federation Provider / Marketplace machine.

2. **TRESOR PaaS**

   The TRESOR PaaS was hosted on two VMs in the testbed. The `paas-broker` VM provided the management functions of the PaaS platform, for example, the user interface, management APIs, monitoring, distribution of applications, etcetera. The `paas-node` VM served as a PaaS computing node responsible for executing the two cloud services offered in the ecosystem.

3. **SNET Components**

   The remaining VMs provided the components that were developed by SNET: the `broker` and `proxy` VM which provide the TRESOR Broker and Proxy, as well as the `xacml` VM which provides the distributed authorization components PDP and PAP. The `xacml` VM also hosts the ELK logging stack deployment that was used by TRESOR as well as the TRESOR demo application.

Table 4.2: TU Berlin Testbed Machines

Purpose	OS	Name
TRESOR Federation Provider and Marketplace	Win 2012	`fp-mp`
ActiveDirectory for TRESOR	Win 2012	`ad`
ActiveDirectory Testserver 1	Win 2012	`ad-herz`
ActiveDirectory Testserver 2	Win 2012	`ad-pauline`
Test Client German Heart Institute	Win 2012	`client-herz`
Test Client Paulinenkrankenhaus	Win 2012	`client-pauline`
TRESOR PaaS Broker	CentOS 6.5	`paas-broker`
TRESOR PaaS Computing Node	CentOS 6.5	`paas-node`
TRESOR Broker	Ubuntu 14.04 LTS	`broker`
TRESOR Proxy	Ubuntu 14.04 LTS	`proxy`
TRESOR XACML test server	Ubuntu 14.04 LTS	`xacml`

Following the rising popularity of Docker towards the end of TRESOR, containers for the SNET components as well as an integrated Docker Compose compilation were created[3]. It includes dockerized versions of the TRESOR Broker, Proxy, PDP, PAP, and the logging distribution. This provides a helpful tool to demonstrate those components as well as reproducibly test their integration. At last, publishing both the components' code as well as instructions how to execute it supports potential users of the TRESOR results to make use of them in further use cases.

4.4.1 Distributed Authorization: PDP, PAP, and Location Server

The TRESOR production deployment features a distributed authorization architecture that consists of a Policy Administration Point (PAP), a Policy Decision Point (PDP), and a Location Server (LS). End-users can instantiate policy templates for each booked service and use the PAP to manage the access policies persisted in the PDP. This template mechanism was devised by Raschke and Zickau and presented in [132]. An example policy template was created for the TRESOR demo application[4]. It can be used to define authorization rules for each of the different exemplary data types in the demo, for example, patients and vital data.

Another component in the production deployment is the TRESOR Location Server which provides location information to the PDP so that it can evaluate geo-fences in the access policies. These geo-fences are also managed by the PAP. End-users' location information are provided by the TRESOR Location Client, an Android application that needs to be installed on the users' phones. Much more detail about the location-based access control mechanism can be found in the papers by Zickau et al. in [194, 69, 193].

4.4.2 Proxy, Federation Provider, Identity Provider and SkIDentity

The production deployment of the proxy features a configuration that enables all relevant modules, in effect leading to an implementation of the full communication sequence as explained in Section 3.3.2.2. It authenticates users through the Federation Provider, routes to booked services as retrieved from the service registry, carries out RESTful authentication with the PDP and augments HTTP traffic with additional HTTP headers.

In the production environment, there are multiple Identity Providers that can be used for authentication:

- A test provider that allows to select an identity from a list of test identities to simulate different user accounts
- An OpenLDAP installation that is used for development and testing at the German Heart Institute
- The production deployment of the ActiveDirectory of the "Paulinenkrankenhaus" health center which is not available to other participants
- Two ActiveDirectory test domains that simulate the production environments of the two health centers

[3]https://github.com/TU-Berlin-SNET/tresor-ecosystem

[4]https://github.com/TU-Berlin-SNET/tresor-pap/blob/master/xml-templates/template.xml

The sixth identity provider was created in another project of the Trusted Cloud research programme, SkIDentity[5]. SkIDentity provides a SAML-compliant identity provider for "strong" smart card authentication mechanisms. The project supports a number of German developments, such as the electronic ID ("Neuer Personalausweis") the electronic health insurance card ("Elektronische Gesundheitskarte") and the Health Professional Card ("Elektronischer Heilberufsausweis").

The PAP can designate certain application resources to require strong authentication, for example, viewing and editing patient data. When such constraints are in place, the PDP returns an `Indeterminate` authorization response that indicates missing SkIDentity user attributes. The proxy was extended to restart the authentication process in this case with a different `whr` value that indicates to the Federation Provider the requirement for SkIDentity authentication. After the users authenticate, the Federation Provider adds SkIDentity attributes to the existing set of user attributes. These include, for example, which mechanism was used for authentication, information about the smart card, and the resulting identifier. A security feature is the configurable "pseudonymization" of the user identifier. While the same user would be assigned the same ID on subsequent authentications, this ID is different for each service, preventing disclosure of personal identification to the cloud services. Strong authentication is also quite beneficial for auditing purposes as applications can log a user identity that was asserted by trusted entities outside of TRESOR.

Unlike the Federation Provider used in CYCLONE which lets the users choose their Identity Provider, the TRESOR Federation Provider always redirects users to the same host (`tresor-dev-ip.<domain>`). By creating an entry for this host name in the local DNS, customers can replace the default identity provider with their own system. In the production deployment, the DNS was mapped to the test provider.

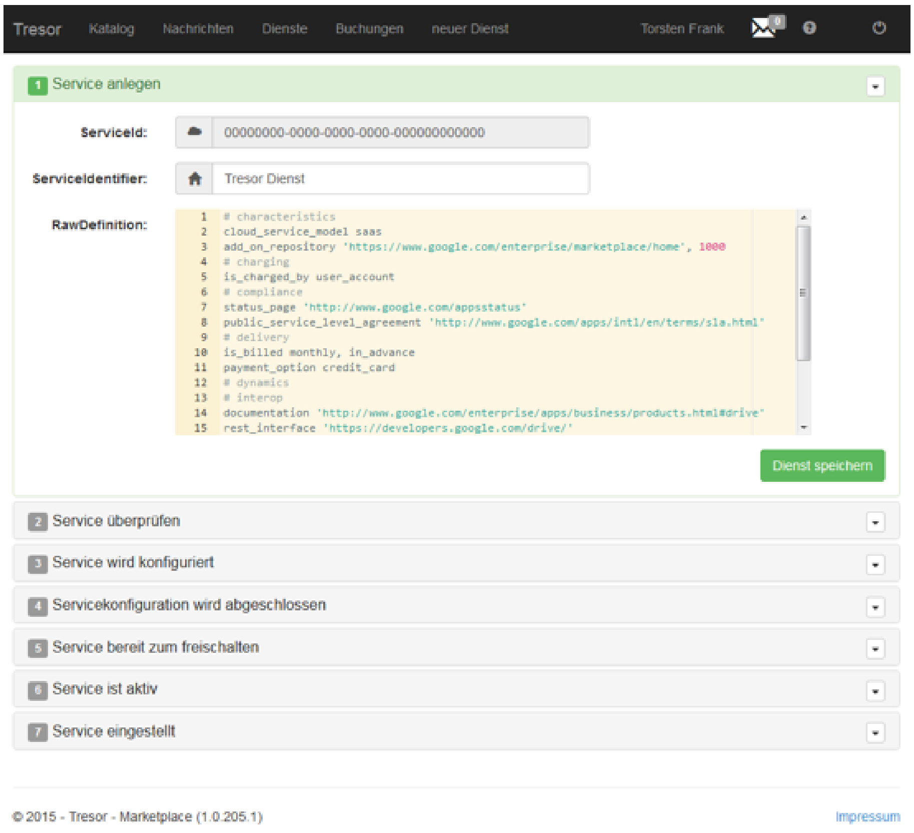
4.4.3 TRESOR Broker and Marketplace

Within the TRESOR production deployment, the TRESOR Marketplace serves as the user-facing service while the TRESOR Broker, based on the service registry architecture, provides the backend functions for managing the essential entities:

- **Services**

  The two production services ("media-break free medical history documentation" and "drug interaction check") were described by the TRESOR Marketplace developers from T-Systems MMS. Furthermore, they used the other service descriptions (Google Drive for Business and Salesforce Sales Cloud) for testing the compatibility of the Marketplace to the SDL-NG, as shown in Figure 4.6. At last, the internal TRESOR services were also described in the Broker, for example, the logging distribution and the TRESOR PAP. Figure 4.7 shows the resulting Marketplace directory. The XSD export of the service vocabulary which was provided by the Broker was quite important for the integration between both systems as it allows the automated creation of .NET classes that considerably ease the handling of retrieved data by the Marketplace backend.

[5]https://www.skidentity.de/



ServiceId: 00000000-0000-0000-0000-000000000000

ServiceIdentifier: Tresor Dienst

RawDefinition:

```
# characteristics
cloud_service_model saas
add_on_repository 'https://www.google.com/enterprise/marketplace/home', 1000
# charging
is_charged_by user_account
# compliance
status_page 'http://www.google.com/appsstatus'
public_service_level_agreement 'http://www.google.com/apps/intl/en/terms/sla.html'
# delivery
is_billed monthly, in_advance
payment_option credit_card
# dynamics
# interop
documentation 'http://www.google.com/enterprise/apps/business/products.html#drive'
rest_interface 'https://developers.google.com/drive/'
```

Figure 4.6: TRESOR Marketplace: Service Creation using the SDL-NG [38, p. 9]

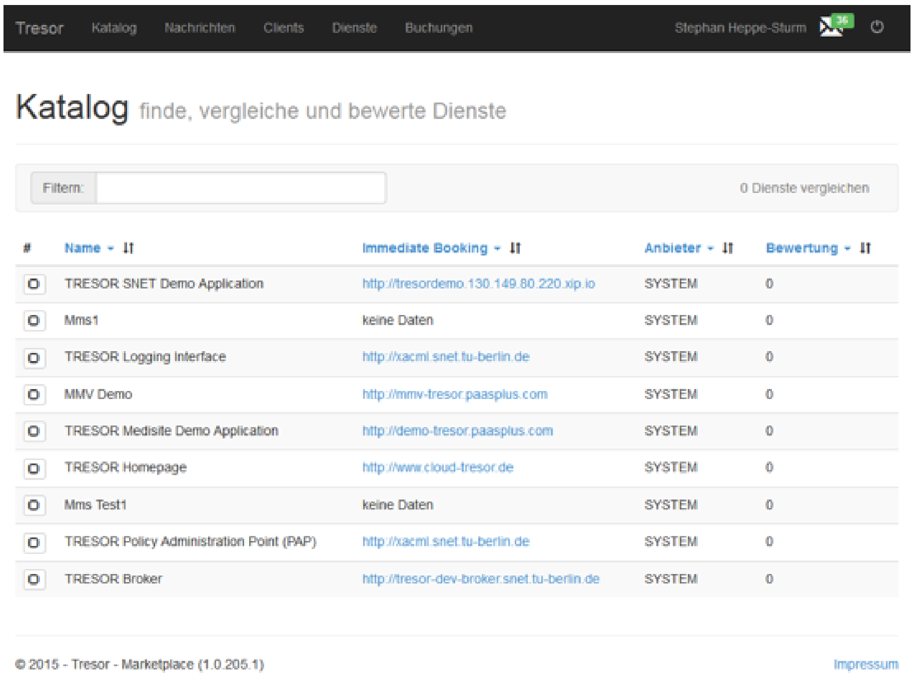
- **Service versions**

  The TRESOR Broker provides versioning capabilities for the service descriptions. This has a number of uses, for example, to make any service change explicit and open to scrutiny, to provide dependable service-level agreements, and to provide a transparent history of any service on the marketplace.

- **Service filters**

  Initially, the conception of the TRESOR Marketplace included a filter that would prevent client organizations from booking certain services that did not meet their requirements, for example, regarding data protection. That's why the TRESOR Broker includes a facility to post client-specific filters that are applied when retrieving services. However, the use case partners determined that one of the main benefits of the TRESOR Marketplace should be that it provides only services that are pre-checked if they are compliant to the common laws and regulations regarding medical cloud services. Therefore, the filters were not used in production.

- **Clients and providers**



Katalog finde, vergleiche und bewerte Dienste

Filtern:

0 Dienste vergleichen

#	Name	Immediate Booking	Anbieter	Bewertung
○	TRESOR SNET Demo Application	http://tresordemo.130.149.80.220.xip.io	SYSTEM	0
○	Mms1	keine Daten	SYSTEM	0
○	TRESOR Logging Interface	http://xacml.snet.tu-berlin.de	SYSTEM	0
○	MMV Demo	http://mmv-tresor.paasplus.com	SYSTEM	0
○	TRESOR Medisite Demo Application	http://demo-tresor.paasplus.com	SYSTEM	0
○	TRESOR Homepage	http://www.cloud-tresor.de	SYSTEM	0
○	Mms Test1	keine Daten	SYSTEM	0
○	TRESOR Policy Administration Point (PAP)	http://xacml.snet.tu-berlin.de	SYSTEM	0
○	TRESOR Broker	http://tresor-dev-broker.snet.tu-berlin.de	SYSTEM	0

© 2015 - Tresor - Marketplace (1.0.205.1)

Impressum

Figure 4.7: TRESOR Marketplace: Catalog [38, p. 9]

The TRESOR Broker also maintains the TRESOR client and provider information. Service descriptions are linked to the providers while clients are associated with service bookings. As further information about clients and providers is rather relevant to TRESOR users instead of other backend services, it is managed by the Marketplace instead of the Broker. As a result, the TRESOR Broker solemnly maintains identifiers and a limited set of metadata.

- **Service bookings**

  A booking is a relationship between a client and a specific service version. The initial conception of the Broker, Marketplace, and the PaaS designated the Broker to asynchronously invoke an API of the PaaS platform to create a new service instance for a specific client. As a result of this invocation, the PaaS platform should return the public URL to this specific instance and the Broker should persist it afterwards. However, as this API was not available in the project, the broker contains a workaround to always use the same instance URL after service booking. Still, the URL is retrieved on demand by the proxy to route traffic to the service.

4.4.4 PaaS, Cloud Services, and Logging

Figure 4.8 provides an overview about the TRESOR PaaS which was implemented by the company "Medisite". It is a highly modularized PaaS platform comprising four layers, each featuring a number of modules. The PaaS Platform

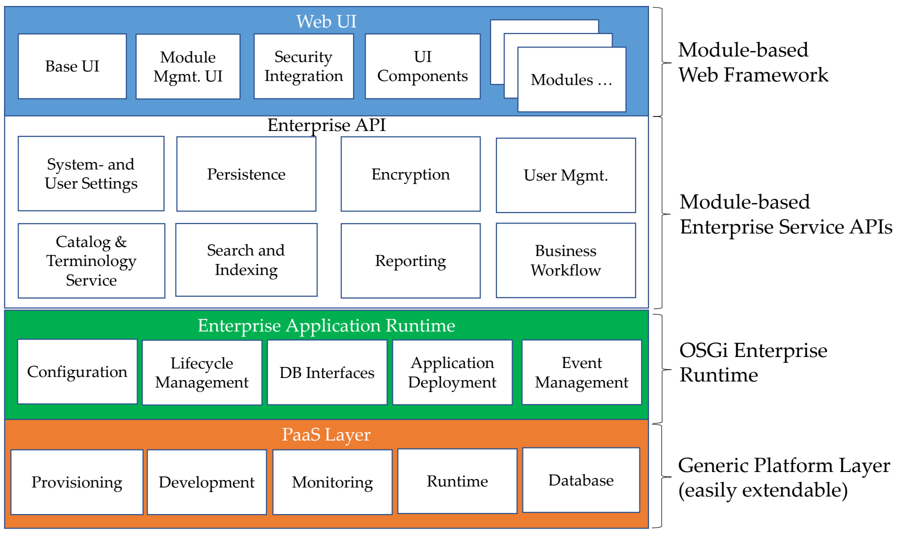


Figure 4.8: TRESOR PaaS Platform, translated from a figure by Frank on [38, p. 11]

is marketed as "PaaS+" as it does not only provide the basic platform services expected from any PaaS, but also a large number of domain-specific modules, for example, to interact with health data formats such as the widely-used standard HL7[6]. The TRESOR PaaS is based on RedHat OpenShift[7], an open-source PaaS solution. The two example services were packaged as application bundles and deployed by "Medisite".

For testing purposes, Medisite installed and configured a TCTP-enabled instance of the proxy on the PaaS node server. Creating a TCTP-secured connection from a development machine to the production deployment was successful. However, this set-up was not repeated with the production services for two reasons: first, the "media-break free medical history documentation" did not require TCTP as the transmitted data was already encrypted. Second, the "drug interaction check" client was the productive SAP installation of the "Paulinenkrankenhaus". Routing the SAP communication of the production system through a local proxy was not feasible.

TRESOR Logging

The TRESOR logging consists of a regular ELK stack that receives log data from all components. It was customized for the specific requirements of the project:

- Logstash was configured to accept JSON data via a regular TCP socket as this was the most commonly supported option by all components.
- Kibana was integrated with the Federation Provider to allow authenticated access to the logs by checking the presence of the

[6]http://www.hl7.org/
[7]https://www.openshift.com/

TRESOR-Organization-UUID header which indicates that authentication with the TRESOR proxy was performed.
- There is a static log data filter unchangeable by the users which compares the TRESOR-Organization-UUID header with the respective field in the log messages. As it restricts log viewing to only those entries that belong to the respective client organization, it makes the logging deployment multi-tenancy capable.
- The Kibana dashboard contains UI elements to filter the log data based on two additional TRESOR-specific log fields: `tresor-component`, designating the respective TRESOR component ("Proxy", "Broker", "PDP", etc.), and `subject-id` containing the user identifier (for example, "DHZB/JStock").

4.5 Architecture Blueprint for End-to-End Security in Medical SaaS Offerings

This section presents a generalization of the concrete TRESOR deployment of the components that have been developed in the context of this thesis. This architecture blueprint serves as a guidance to other researchers and practitioners how to implement trustful and secure SaaS applications in medical settings in general, without being too specific to the concepts of TRESOR.

The following subsections first outline the motivation of the blueprint as well as the application characteristics that it is especially well suited for. Afterwards, the architecture and a prototypical implementation are explained. The section concludes with an explication how the main requirements of the end-users are met as well as a performance evaluation of the implementation.

4.5.1 Motivation

Despite the currently observable advances in healthcare information technologies and electronic health (e-health), decisions about treatments and diagnostic procedures are still commonly delayed as medical records from other healthcare providers are being transferred by casual means, instead of using IT. As a consequence, healthcare centers consider adopting new IT paradigms, such Cloud Computing.

Public Cloud Software-as-a-Service (SaaS) offerings, such as Salesforce Sales Cloud and Google Drive for Business, have been adopted among a considerable number of users in their respective application domain. In healthcare, they could enjoy a high adoption as well. Especially in terms of medical records sharing, Cloud Computing promises substantial improvements as shown by Chen et al. in [36]. Nevertheless, as argued by Li et al. and Chen et al. in [97, 36] the affected parties can be concerned about patient data security and privacy. Thereupon, sensitive medical records are the subject of individuals' protection plans issued by lawmakers and implemented by healthcare centers. In particular, the regulations require the prevention of patient data access by illegitimate parties, such as Cloud Computing intermediaries which are found within most public cloud offerings. There are a number of proposed countermeasures to address these issues in the related work as summarized by Abbas and Khan in [1].

However, the end-to-end security of medical records communication with medical SaaS solutions has not been the scope of research so far. This section uses TCTP to design and implement a blueprint for e-health solution architectures that establish end-to-end security mechanism to prevent intermediary data access and therefore to ensure appropriate patient data privacy and security. The evaluation of this approach demonstrates its fulfilment of healthcare-specific security and privacy requirements, low implementation efforts, and no measurable performance overhead in a practical benchmark.

4.5.2 Application Characteristics

There are three main characteristics of medical hypermedia applications that the application blueprint fits best to: they are RESTful HTTP applications (1), are accessed through HTTP management proxies (2), and deployed as SaaS in a shared environment (3).

RESTful design (1)

Besides utilizing HTTP as a transfer protocol, the applications follow the conceptual framework laid out in the work of Fielding [54], that is, the interaction with these applications through user agents should represent operations on resourceful abstractions of health data, for example, patient records, treatment journals, or medication plans. Following these concepts allows the blueprint to make generalized assumptions about application characteristics, such as the semantics of HTTP operations. All of this leads to a generalized solution for securing almost any RESTful application, instead of potentially developing a singular solution for a specific application.

Communication through HTTP management proxies (2)

If a user agent and an origin server would be connected directly, transport layer protocols, such as Transport Layer Security (TLS) provide end-to-end security [167]. However, most Cloud Computing environments include intermediary management proxies, such as load balancers, reverse proxies, and caching cloud optimizers. Furthermore, many organizations opt to filter outgoing internet traffic to impose restrictions on their employees' internet access, introducing yet another intermediary. To carry out their tasks, proxies need to have access to communication plaintext, which is only possible if they act as TLS server connection ends. This violates the "Need-to-know-Principle", as those intermediaries do not need to have access to the HTTP entity-body to carry out their functions, yet can access it at any time.

Deployment in a shared environment (3)

Shared environments, for example, public cloud platforms, feature additional ramifications of security breaches at HTTP management proxies, in comparison to private solutions. Many have a high visibility, such as Amazon EC2 or Microsoft Azure, making these offerings and the solutions deployed on them especially worthwhile targets for security attacks. Furthermore, when security breaches happen, there is a potentially large and diverse group of affected tenants. Still, the application of end-to-end encryption through the blueprint carries security benefits for *any* solution architecture, for example, by preventing the access by intermediaries to communication plaintext.

4.5.3 Blueprint Architecture and Prototype

The blueprint consists of two parts: first, the *security technology* that enables end-to-end HTTP entity body encryption, second, the *software* on both client and server that implements this technology. The client and server security software should handle the encryption of HTTP entity bodies. Client security software can be instantiated as browser add-ons, HTTP client library extensions, and local HTTP proxies. Server software can be integrated into application frameworks, server middleware, HTTP servers, and remote HTTP proxies. The components of the blueprint are depicted in Figure 4.9, along with a short description of their roles.

The implementation of the blueprint that is described in the following is shown in Figure 4.10 and comprises: a *user agent* (e.g., a browser) on a *client* (e.g., a health center PC) that communicates with a locally installed *TCTP proxy* that was presented in Section 3.3.2. This proxy communicates with the *Redhat OpenShift platform* that hosts a *PaaS container* (an OpenShift "gear") in turn hosting a RESTful HTTP application. This *demo application* imitates a contemporary cloud-based patient data management system. It was implemented by one of the students working for TRESOR.

The demo application contains the TCTP Rack middleware which was introduced in Section 4.2 acting as the TCTP server. Every OpenShift compute node features a *cloud intermediary*, an HTTP reverse proxy, in addition to a potentially also existing company proxy at the health center. While TCTP, the TCTP proxy, and the middleware are specific instantiations of blueprint components, RedHat OpenShift and the demo application are selected for illustrative purposes of contemporary PaaS solutions and RESTful applications.

4.5.4 Meeting the Blueprint Stakeholders' Requirements

Based on the application characteristics, the following subsections iterate the main requirements for the blueprint as well as how the implementation fits to them. The requirements were gathered in multiple stakeholder workshops in the context of TRESOR and mainly reflect the statements of the use case partners from the two German health centers.

Health Data End-to-end Security

Confidentiality. Within medical RESTful applications, health data is contained in the entity bodies of HTTP requests and responses. Considering this, confidentiality of health data is achieved if unauthorized intermediaries are prevented from accessing those entities that are transferred between the workstation and the origin server. To achieve this confidentiality, entity-bodies need to be encrypted with keys exchanged between the client and the origin server. Otherwise, the confidentiality would not be "end-to-end", for example, when just encrypting hop-to-hop by establishing a TLS connection from the client to a cloud intermediary.

TCTP encapsulates the TLS handshaking protocols to set up encapsulated TLS channels (HALECs), as explicated in Section 3.2.3. This design enables the

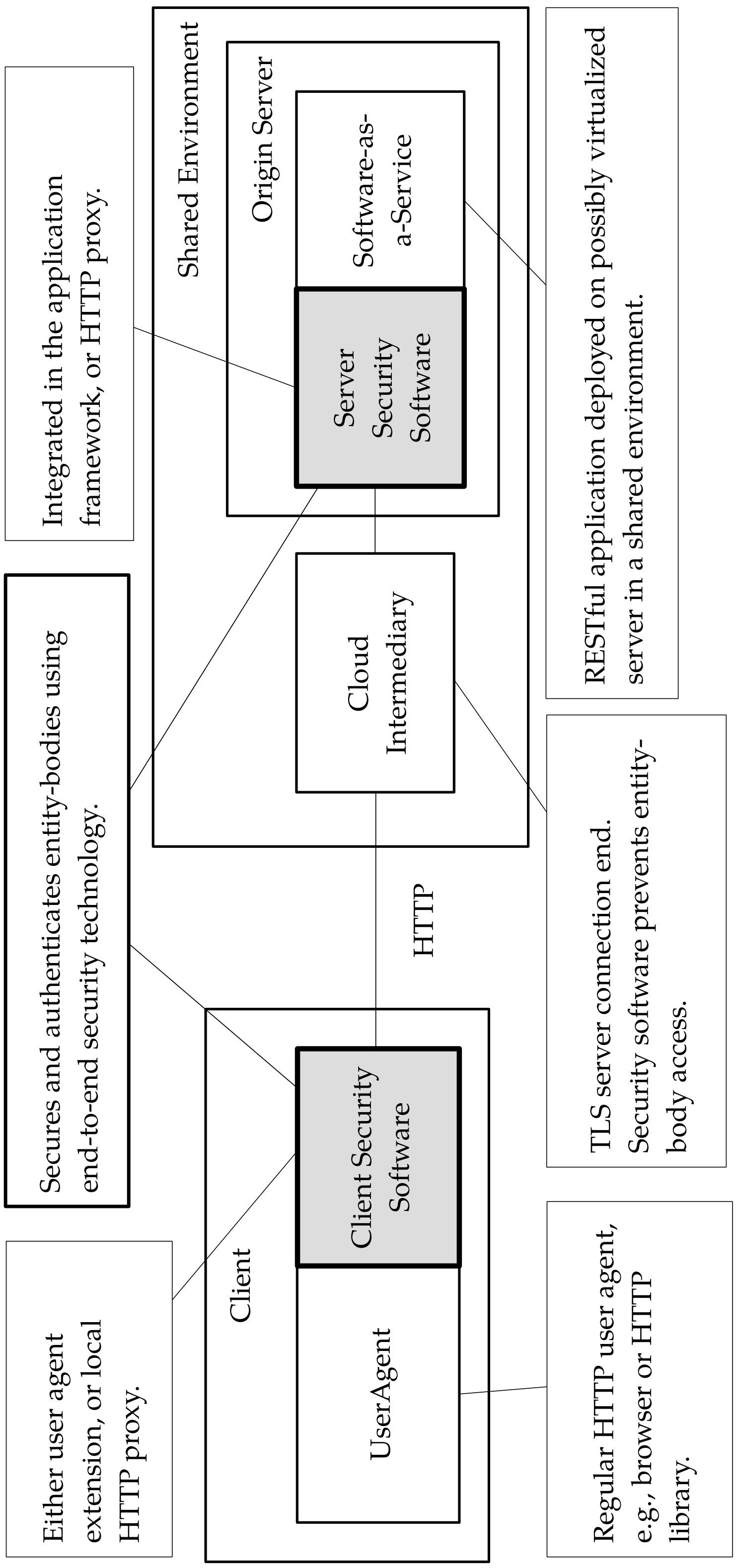

Integrated in the application framework, or HTTP proxy.
Shared Environment
Origin Server
Software-as-a-Service
Server Security Software
RESTful application deployed on possibly virtualized server in a shared environment.
Secures and authenticates entity-bodies using end-to-end security technology.
Cloud Intermediary
TLS server connection end. Security software prevents entity-body access.
HTTP
Client
Client Security Software
UserAgent
Either user agent extension, or local HTTP proxy.
Regular HTTP user agent, e.g., browser or HTTP library.

Figure 4.9: The Blueprint Components

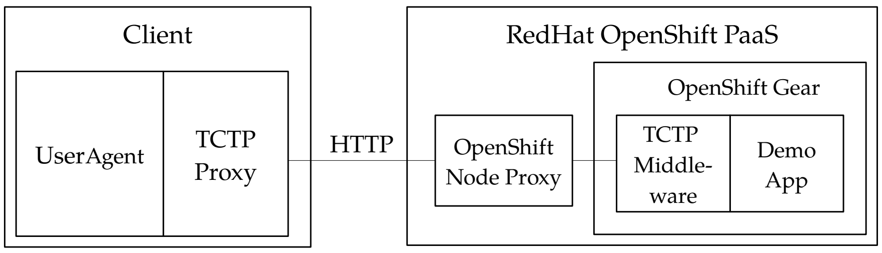


Figure 4.10: TCTP Prototype Implementation

secure on-line exchange of encryption keys as well as the negotiation of encryption algorithms. The exchange itself can rely on strong encryption algorithms, such as Elliptic Curve Diffie-Hellman [167, p.304-308], adding further strength to the security of the encryption.

Integrity. Besides the confidentiality, the integrity of the transferred health data should also be ensured by the blueprint. As it is relayed through possibly multiple HTTP management proxies, there are ample possibilities for integrity violations. These include, for example, altering the data, reordering messages, and replaying previously transmitted data.

All of these threats are mitigated by relying on the secure authentication of encrypted records included in the TLS protocol and therefore available to the medical applications via TCTP. The notion of a channel with an associated TLS session state enables the discovery of replayed and reordered HTTP messages through the TLS HMAC [167, p.372-376] mechanism.

Full HTTP Compliance

As the blueprint should be applicable to all RESTful applications, the communication between all components needs to be fully compliant to HTTP, that is, any cloud intermediary is able to process exchanged messages for maximum compatibility. Furthermore, the client and server security software should also be able to decide when to use encryption and when unencrypted communication is sufficient, for example, when transferring static HTML, CSS, and JavaScript assets.

The reliance of TCTP on standards-compliant use of the HTTP protocol, especially using the `Content-Encoding` header as well as an encapsulation of the TLS protocol in HTTP entity-bodies, ensures that the messages can be understood by any standards-compliant HTTP intermediary. Furthermore, TCTP supports the fragmentation of HTTP messages. If TCTP would only operate on complete entities, this would prevent media streaming applications such as telemedicine and HTTP comet technologies such as HTTP server push.

The decision when to use encryption is based on the TCTP discovery mechanism, presented in Section 3.2.2. This allows specifying differentiated encryption: for example, encrypting sensitive operations, such as updating patient records, while transferring interface assets, such as style sheets or icons, in plaintext. It can also discover if an origin server supports TCTP, preventing potentially unnecessary round trips for sending encrypted content to a server that lacks the server security software. In effect, the discovery mechanism can considerably lower the overall TCTP overhead.

Customer Control over Encryption Keys

The keys that are used for encrypting medical data need to be under the control of the customer to prevent attacks on the encryption by the provider and other parties who can potentially gain access. The stakeholders emphasized that other procedures, such as provider-issued encryption keys, would be unacceptable from a practical security viewpoint. Additionally, they would violate legal provisions that require certain security guarantees that could not be met if the provider could access the encryption keys. These issues prevent health centers to use cloud load balancers that act as a TLS server connection end, such as the Amazon Cloud Load Balancer [7].

Using TCTP, the key exchange is carried out between computing nodes that are under the control of the health centers: the user's workstations, as well as the origin servers. The health centers can therefore configure the TCTP server software to use certificates for key exchange that they have issued themselves, thus integrating TCTP into the PKI infrastructure of the health centers.

Low Implementation Efforts and Performance Overheads {#sec:tctp-ecis-benchmark}

To ensure the practical applicability of the blueprint, its components should impose a low overhead for encryption and authentication, possibly relying on hardware encryption functionality, such as AES-NI [167, p.139-167]. Furthermore, the effort for implementing such architectures should be reduced, for example, by reusing existing components, extensive modularization, or relying on established and mature technologies.

The TLS implementations used for implementing TCTP in the TRESOR proxy and the Rack TCTP middleware are based on the OpenSSL library [180], which is a very mature and widely used security library that uses hardware acceleration, where possible. When implementing the employed software, both DTSTTCPW and YAGNI paradigms were followed in a similar way than within the service registry implementation, as explained in Section 3.1.9. Therefore, the source code is restricted to the essential features, resulting in a quite short and concise implementation.

Implementation efforts. The implementation efforts for similar architecture instantiations include installing the TCTP proxy on the client, and adding the TCTP middleware to the Rack configuration file (in case of Ruby based applications), or installing the TCTP proxy on the server (in case of other programming environments). The experience in setting up the benchmark environment lead to the assumption that performing these actions would require an experienced programmer at most one person hour. As the TCTP proxy is implemented using the Ruby programming language, it is compatible to a wide range of operating systems: Microsoft Windows, Linux, Mac OS X, Solaris, and FreeBSD.

Performance overhead. A synthetic analysis of the performance overhead introduced by TCTP was explained earlier in Section 4.2. This analysis hints at an expected performance impact between 5% and 10%. In the following, the communication overhead is assessed in a realistic setting - the interaction of a user agent with the demo application, hosted on the OpenShift platform, secured by TCTP. The benchmark script simulates a common workflow for medical personnel: logging in, getting a list of patients, creating patient records (five in the benchmark), updating those records with medication and illness information, and logging out.

There are three communication scenarios that are compared to each other regarding the execution duration of the example workflow:

1. Direct communication between demo script and application
2. Proxy communication without TCTP
3. Proxy communication with TCTP

The goal is to evaluate how the expected variance in performance of a shared cloud environment relates to the overhead introduced by the proxy and TCTP. The measured values are averaged over ten consecutive repetitions. The benchmark is also repeated six and twelve hours later to get samples from different times of the day, and therefore different utilization of the cloud platform. The benchmark client is a desktop workstation equipped with an Intel Core 2 Duo E8400 CPU, running Debian Wheezy 64bit. The network at the SNET labs features a 1 GBit/s link to the Internet.

Table 4.3: Mean communication time of medical workflow in relation to access means.

Run	Direct	Proxy	Proxy OH	Proxy + TCTP	P+T OH
1st	9,26 s	10,31 s	+11,3%	10,42 s	+12,5%
2nd	12,53 s	9,56 s	-23,7%	11,57 s	-7,7%
3rd	14,62 s	10,97 s	-25,0%	10,45 s	-28,5%

The benchmark results are shown in Table 4.3. In this realistic benchmark setting, the anticipated performance variations of a shared cloud environment conceal any communication overhead of the proxy and TCTP. In fact, the communication time fluctuates also within the ten repetitions of the workflow, which is exemplified by Figure 4.11 showing these time variations for two exemplary workflow actions: listing patients and saving a patient record.

These observations can be used to conclude that the application blueprint can be applied in a realistic setting without impacting the performance in a way that would be distinguishable from the usual performance fluctuations of public cloud offerings.

4.5.5 Summary

The evaluation shows that the beneficial impacts of TCTP fit quite well to the requirements and constraints of medical Software-as-a-Service solutions. When implementing the blueprint, cloud-based health solutions can transmit sensitive medical records between healthcare providers securely using end-to-end encryption, thus meeting legal requirements, for example, German privacy law. In particular, the proposed blueprint guarantees the confidentiality and

[8] Use of violin plot first proposed by Ermakova for [159], who also provided the first draft of this figure.

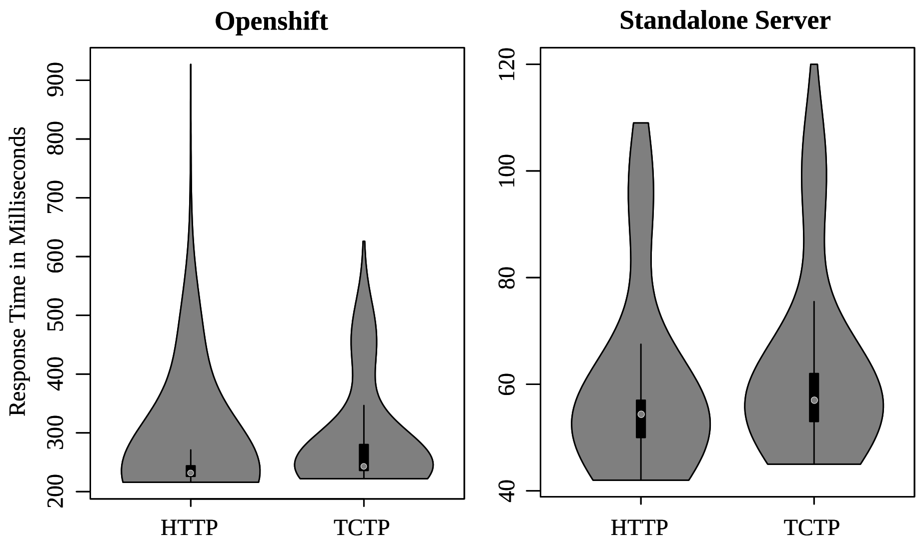


Figure 4.11: Fluctuations of completion time for two exemplary workflow actions[8]

integrity of transferred medical records between cloud services and the health centers; all of this while having very low implementation efforts and only a small performance overhead.

Still, several challenges need to be addressed in future work. While the blueprint presumes that confidential medical records are transmitted as a part of the HTTP entity-bodies, some information in the HTTP headers potentially also contain sensitive data that would be accessible by Cloud Computing intermediaries. An example are patient identifiers in the URLs of GET requests. This challenge can be mitigated by different methods, for example, encrypting those parts of the URLs, using pseudonymous identifiers that are unique for each end user, and forming encrypted POST requests for transmitting those identifiers. Furthermore, there is no TCTP-specific approach to authenticity, as it relies on the security of the certificates used for authenticating the TCTP server and potentially also the client. While this was sufficient for the TRESOR stakeholders, there was no thorough analysis conducted in this area regarding other end-users.

At last, as almost all SaaS cloud applications implement a RESTful design, there was never the goal to provide a blueprint for non-RESTful applications. Still, all communication protocols that use HTTP can be secured by TCTP, yet there are already existing end-to-end technologies for some protocols, such as WS-Security for SOAP-based solutions. Nevertheless, TCTP can provide additional functionality, for example, data-flow protection.

4.6 Securing CYCLONE

This section presents the evaluation of the security architecture that was proposed in Section 3.4 using three methods: first, the architecture is applied within CYCLONE to solve specific issues in multi-cloud deployments that manifest themselves within the CYCLONE use cases. This provides insights into the applicability and future direction of the contribution. Second, the main component of the architecture, the CYCLONE Federation Provider, is subjected to a thorough application threat analysis, resulting in a list of risks and mitigations. At last, the security architecture is analyzed with respect to its economic benefits in order to provide arguments for its take-up by the industry. The last subsection concludes this part with presenting the existing limitations in the approach.

4.6.1 Applying the Security Architecture within CYCLONE

The following subsections summarize the diverse activities that needed to be carried out to apply the security architecture within CYCLONE.

Establishing the Federation Provider in eduGAIN

As a prerequisite for being able to use federated identities in the use cases, the Federation Provider was first set up within the development environment at SNET and registered with eduGAIN through the TUBit, the IT service department of TU Berlin. Specifically, "registering" denotes importing the Federation Provider metadata[9] as a Service Provider (SP) through an internal DFN portal which only appointed people from TUBit have access to.

To allow the retrieval of further identity attributes from other eduGAIN-participating institutions, for example, the display name of the user, the Federation Provider follows the "Data Protection Code of Conduct Cookbook"[10]. However, there are two major difficulties: first of all, not every Identity Provider (IdP) is fully accustomed with all of the accompanying technologies and procedures - requiring manual coordination effort to get user attributes. Second, there are only recommendations but no requirements for the attribute release. Some IdPs chose to release all attributes, some only when SPs follow the Data Protection CoC, some implement diverse approval processes, and some release no attributes at all.

Additionally, there is a set of attributes recommended for every eduGAIN identity provider[11], for example, display name and home organization. The research institutions are free to implement any number of these attributes and can also introduce additional ones, for example, group membership. To resume testing the Federation Provider in spite of all these circumstances, the Federation Provider also supports creating local user accounts and could use an LDAP server for special cases not involving federated identities.

In contrast to earlier goals of CYCLONE, all of these circumstances provide such high barriers that it was not possible to create a single Federation Provider that would support *all* of the 2400+ Identity Providers in eduGAIN. In the end, the consortium focused its efforts on providing the best support for the institutions that were involved in the project.

Mapping eduGAIN Attributes onto JWT Claims

[9]https://technical.edugain.org/show_entity_details.php?entity_row_id=213
[10]https://wiki.edugain.org/Data_Protection_Code_of_Conduct_Cookbook
[11]https://wiki.edugain.org/IDP_Attribute_Profile:_recommended_attributes

The following essential claims are mapped from eduGAIN attributes onto JWT claims:

- A unique user identifier (`eduPersonPrincipalName`)
- The home organization's domain name (`schacHomeOrganization`)
- User's relationship(s) to their institution (`eduPersonAffiliation`)
- The preferred name when displaying entries (`displayName`)

The main challenge when generating the JWT `sub` claim, which identifies users, is the variety of ways how eduGAIN Identity Providers issue identifiers and other data that can be used to generate this claim. As the concrete manner is not known to the Federation Provider beforehand, it iteratively checks for the existence of any attribute that could be used from the most to the least reliable ID. The algorithm it uses can be found in the source code[12].

Deploying the Federation Provider to Production

The Federation Provider is deployed from a Docker-based GitHub repository [13] and configured using certificates and metadata registered with eduGAIN. For deployments in other projects, CYCLONE provides a SlipStream module as well as an example Keycloak configuration file (`keycloak-export.json`) that contains default and exemplary roles, clients, and users.

The Federation Provider relies on the PHP library SimpleSAMLphp[14] for integration with eduGAIN. This library solves two important issues of Keycloak: First, when importing eduGAIN metadata, Keycloak creates one `AssertionConsumerService` for each of the 2400+ SAML identity providers instead of one service for all of them, analogous to SimpleSAMLphp and Shibboleth. This substantially enlarges the service provider metadata of the Federation Provider, raising interoperability and management issues. Second, the identity broker user interface of SimpleSAMLphp is more sophisticated and performs better overall than the basic interface of Keycloak. For example, SimpleSAMLphp provides "search as you type" for the list of IdPs as well as bookmarking for the preferred IdPs, making IdP selection far more rapid in subsequent authentication processes, than the basic interface of Keycloak.

There are two additional areas that have been considered when deploying the Federation Provider:

Periodic Removal of Personal Data

The European Data Protection Directive specifies that computer systems processing personal data should not keep it longer than needed, for example, the SAML assertions that are issued by the eduGAIN Identity Providers. Based on the recommendations of the data protection officer of TU Berlin, the CYCLONE Federation Provider contains a removal solution that runs periodically and deletes this data from the database.[15] There are plans for the underlying Keycloak Identity Server to also provide this feature out-of-the-box in a later version. However, there is no concrete progress on this issue.

[12]Please see `components/samlbridge/config/config/config.php` in the Federation Provider source

[13]https://github.com/cyclone-project/cyclone-federation-provider-apache-oidc-demo

[14]https://simplesamlphp.org/

[15]See https://github.com/cyclone-project/cyclone-federation-provider/tree/master/components/cache-clean

Consent Screen

Many Service Providers will be registered with the CYCLONE Federation Provider, each having different people responsible for their operation as well as different data protection officers. By default, Keycloak presents users a consent screen before their attributes are relayed to the services. This default screen was extended with comprehensive information about the Service Providers so that end users can make a more informed decision, especially regarding the terms of use, data protection rules, and the people responsible for the service that users log in to.

Self-service Registration

When a new application is deployed that should use the CYCLONE Federation Provider for authentication, it needs to have a new client account for OpenID Connect created. At the beginning of the project, this was a manual process, requiring the Federation Provider operator to create a new client in the back end. Berdonces-Bonelo extended the Keycloak server with a self-service registration API which allows application DevOps to create new OpenID Connect clients by themselves [21]. While Keycloak already had internal Java classes to create clients provided by a REST API, this API did not feature user access control and was not multi-tenant capable. His extension adds this multi-tenant capability for the client registration API, assigning created clients to eduGAIN users.

Federated Authentication for the Biomedical Data Analysis VM

The Biomedical data analysis VM allows Bioinformaticians to upload data and retrieve analysis results at a later point in time, both via an HTTP interface. Extending this upload form with federated authentication proved quite straightforward: as the form was presented using the Apache HTTP server, the server module `mod_auth_openidc`[16] was used to implement OpenID Connect.

PAM-based Federated Authorization

Bioinformaticians collaboratively use the "microbial genomes analysis" as well as the "live remote cloud sequencing data processing" VMs and require simple data sharing between them. As they access the VMs using SSH and the SSH-based X2Go remote desktop, enforcing access control using Linux file system ACLs suggested itself. CYCLONE integrated the PAM module into the VMs to map federated identities to local user accounts. Now the bioinformaticians can, for example, securely assign access rights to any collaborator using their email address, creating a highly usable and simple procedure.

Extending SlipStream with Federated Login

Within CYCLONE, SlipStream is used for deployment of applications on the respective clouds, configuring federated authorization through deployment parameters, logging in users via the OIDCACF, and logging all relevant output in the distributed logging system. For reference, the whole deployment and configuration workflow as well as information about scaling applications is described in detail in the SlipStream documentation[17].

[16]https://github.com/pingidentity/mod_auth_openidc

[17]http://ssdocs.sixsq.com/en/latest/index.html

Up until the start of CYCLONE, SlipStream relied on simple username/-password combinations to authenticate users. In order to also allow federated identities to authenticate, the SlipStream login service was made compatible to the CYCLONE Federation Provider. The SlipStream UI now features a CYCLONE icon which users can click on to initiate federated authentication. The details about this integration can be found in [28, sec. 3.2.1].

Demonstrating Federation Provider integration

Within CYCLONE, two tools were developed to demonstrate the use of federated identities through the Federation Provider in a simple manner. First, the `cyclone-federation-provider-apache-oidc-demo`[18] provides a `Dockerfile` as well as further configuration to create a Docker container running an Apache HTTP server that uses the `mod_auth_openidc` to authenticate and authorize users. This represents the most straightforward use of federated identities for simple static file serving.

In addition, `cyclone-federation-provider-apache-oidc-django`[19] illustrates how this setup can be used in conjunction with a web application platform, in this case Python Django. The demonstrators are related to other CYCLONE developments, for example, as many Bioinformatics applications also rely on Apache and Django, the demos were quite helpful for the developers that also needed to include federated identities in their applications.

Securing Wordpress with the Federation Provider {#sec:wordpress}

CYCLONE provides a Docker container that demonstrates the integration of WordPress, a widely used web application, with the CYCLONE Federation Provider[20]. The Dockerfile first installs WordPress before adding and configuring the "Generic OpenID Connect Plugin"[21]. This plugin implements the OIDCACF that lets users log into the service using their federated identities. Especially in academic environments, this integration can be beneficial, for example, to provide collaborative websites for research projects or to let Bioinformaticians share their research results through a state-of-the-art content management system.

Establishing the CYCLONE Logging

CYCLONE relies on a modified version of the logging distribution that was presented in Section 4.4.4. The logging dashboard is integrated with the CYCLONE Federation Provider to let end users login with their federated identities. Instead of the `TRESOR-Organization-UUID` HTTP header which was sent by the TRESOR proxy before, it now extracts the name of the organization from the `schacHomeOrganization` attribute of the JSON web token it receives from the CYCLONE Federation Provider through the Open ID Connect flow. This preserves the mandatory filtering based on matching the organization names from the users' attributes and the log data. The main use cases for this service are diagnosis of errors and providing data for potential security audits.

[18]https://github.com/cyclone-project/cyclone-federation-provider-apache-oidc-demo

[19]https://github.com/cyclone-project/cyclone-federation-provider-apache-oidc-django-demo

[20]https://github.com/cyclone-project/cyclone-demo-wp-docker

[21]https://wordpress.org/plugins/generic-OpenID-Connect/

As the CYCLONE implementation progresses, the logging distribution will be adjusted to additional requirements of the CYCLONE use cases.

4.6.2 Federation Provider Security Modelling and Threat Analysis

This section applies the OWASP Application Threat Modelling method on the Federation Provider to establish a security threat analysis and possible mitigations. The documentation of the method [178] designates three "high level steps" for the analysis: "Decompose the application", "Determine and rank threats" and "Determine countermeasures and mitigation". The following subsections explain these steps and the result of applying them onto the Federation Provider. Unless noted otherwise, all direct quotations cite the official documentation of the method [178].

4.6.2.1 Step 1: Decomposing the Application

"The goal of this step is to gain an understanding of the application and how it interacts with external entities. This goal is achieved by information gathering and documentation. The information gathering process is carried out using a clearly defined structure, which ensures the correct information is collected. This structure also defines how the information should be documented to produce the Threat Model."[178]

Section 3.4 should be examined to better comprehend the application decomposition, as Section 3.4.1 presents a general overview about the security architecture and Section 3.4.2 explains the Federation Provider in detail. The following paragraphs explain the aspects of the application that need to be analyzed in order to determine and rank threats in the subsequent phase.

Table 4.4: External Dependencies

ID	Dependency
D1	The Federation Provider is deployed on a hardened VM with sufficient security for login and file system access.
D2	The database of the FP is an embedded H2 database with persistent disk-based tables.
D3	The Federation Provider Deployment relies on a Docker Compose deployment.
D4	A securely managed SSL reverse proxy is in front of the Wildfly Application Server.

The *external dependencies* are the first aspect that needs to be described. The OWASP threat model defines them as "items external to the code of the application that may pose a threat to the application. These items are typically still within the control of the organization, but possibly not within the control of the development team. The first area to look at when investigating external dependencies is how the application will be deployed in a production environment, and what are the requirements surrounding this. This involves looking at how the application is or is not intended to be run. For example if the application is expected to be run on a server that has been hardened to the organization's hardening standard and it is expected to sit behind a firewall, then this information should be documented in the external dependencies section.". Table 4.4 contains those of the Federation Provider, mainly referring to the way it is deployed.

Table 4.5: Trust Levels

ID	Name	Description
T1	Anonymous Web User	A non-authenticated user connected to the Federation Provider
T2	User with eduGAIN Identity	A user who has authenticated via eduGAIN
T3	Relying Party	A third party relying on the Federation Provider for authenticating users
T4	FP Admin	A Federation Provider administrator
T5	FP DevOp	A Federation Provider DevOps engineer

The *trust levels* "represent the access rights that the application will grant to external entities". They are "cross referenced with the entry points and assets", explained later in this section. Table 4.5 contains the different trust levels of the Federation Provider.

Table 4.6: Entry Points

ID	Name	Description	Trust Levels
E1	Wildfly Application Server (HTTP)	The HTTP interface to the Wildfly application server powering the FP. It is reverse proxied by a secure SSL proxy.	
E1.1	OpenID Connect API	The API used by relying parties to authenticate federated users via the FP	T3
E1.2	Administration Console	The Servlet used for administering the FP	T4
E1.3	Log-in Screen	The FP log-in screen	T1,T2,T4
E1.4	Account Manager	The Servlet used to manage users' own accounts	T2,T4
E2	Apache	The HTTP interface to an Apache HTTP server containing the SimpleSamlPHP bridge to eduGAIN	
E2.1	SimpleSamlPHP Bridge	The SimpleSamlPHP Bridge (SP) to connect to eduGAIN	T1,T2,T4

The *entry points* “define the interfaces through which potential attackers can interact with the application or supply it with data. In order for a potential attacker to attack an application, entry points must exist. Entry points in an application can be layered, for example each web page in a web application may contain multiple entry points.”. Table 4.6 contains those of the Federation Provider. In general, there are two entry points: the Wildfly HTTP Application Server, mainly executing Keycloak, and the Apache HTTP Server that provides the SimpleSamlPHP bridge to eduGAIN.

Table 4.7: Assets

ID	Name	Description	Trust Levels
A1	FP Database	Assets regarding the H2 database of the FP	
A1.1	User List	A list of all known users and their home organizations	T4
A1.2	Client List	A list of all OpenID clients, containing their certificates, redirect URIs, etc.	T4
A1.3	Keycloak Config	The general configuration of Keycloak	T5
A2	OpenID Connect API	Assets regarding the OpenID Connect API	
A2.1	Personal Data	The personal data of the logged in users, provided by their home organizations via eduGAIN	T3
A3	SimpleSamlPHP	Assets regarding the SimpleSamlPHP Bridge to eduGAIN	
A3.1	SAML configuration	The certificate and metadata for the eduGAIN integration	T5

Finally, Table 4.7 provides the *assets* within the Federation Provider that need to be protected. “The system must have something that the attacker is interested in; these items/areas of interest are defined as assets. Assets are essentially threat targets, i.e. they are the reason threats will exist. Assets can be both physical assets and abstract assets. For example, an asset of an application might be a list of clients and their personal information; this is a physical asset. An abstract asset might be the reputation of an organization.”

All of these information provide the basis for the modelling of the data flow within a data flow diagram (DFD), as is shown for the Federation Provider in Figure 4.12. The main rationale for this is to “gain a better understanding of the application by providing a visual representation of how the application processes data. The focus of the DFDs is on how data moves through the application and what happens to the data as it moves. DFDs are hierarchical in structure, so they can be used to decompose the application into subsystems and lower-level subsystems. The high level DFD will allow us to clarify the scope of the application being modeled. The lower level iterations will allow us to focus on the specific processes involved when processing specific data.”

The DFD contains five distinct Security Domains:

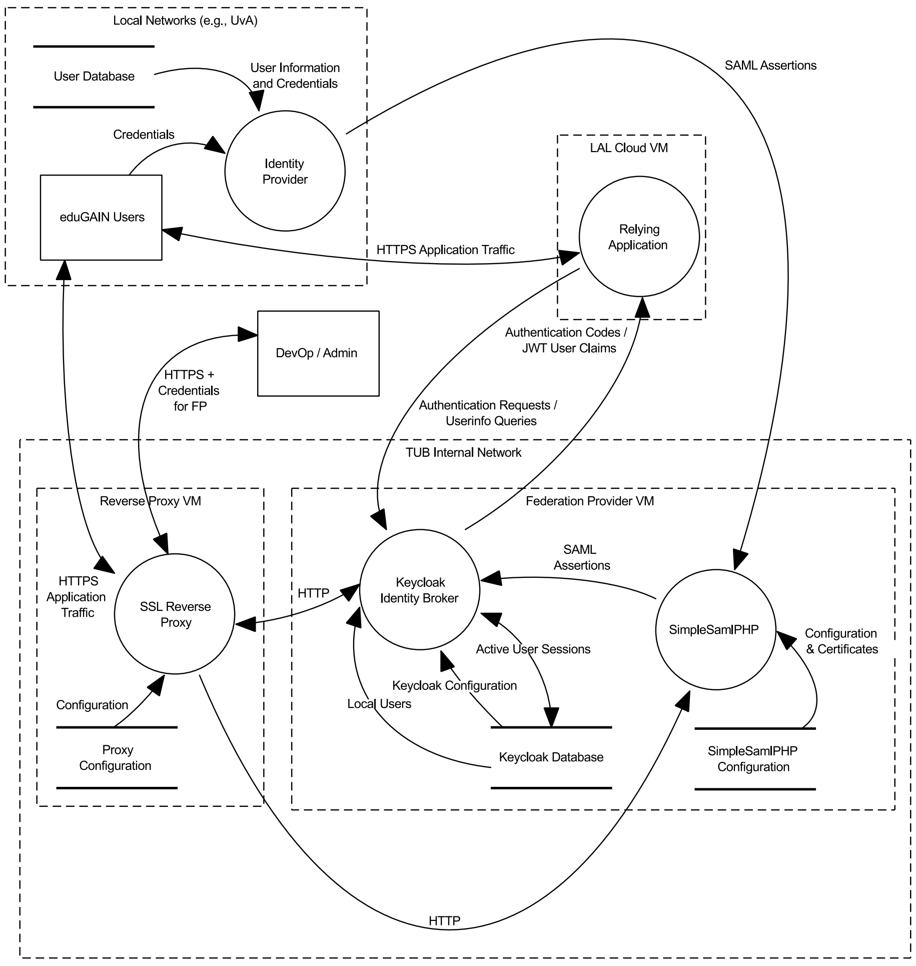


Figure 4.12: Data Flow Diagram

1. **TU Berlin Internal Network.** The network between the reverse proxy VM and the Federation Provider VM. It is exclusively owned, managed, and secured by TU Berlin. The communication path between both VMs can be considered secure, that is, there are a number of controls in place so that eavesdropping on the communication is not possible, allowing the use of HTTP for inter-VM traffic.

2. **Reverse Proxy VM.** The reverse proxy VM serves as a TLS reverse-proxy for all SNET-managed publicly accessible services.

3. **Federation Provider VM.** The Federation Provider VM hosts both the Keycloak identity broker as well as the SimpleSamlPHP bridge to eduGAIN identity providers. It is accessed via reverse-proxied HTTP requests from the reverse proxy VM.

4. **LAL Cloud.** All bioinformatics applications are deployed within VMs on the LAL cloud.

5. **Local Networks (e.g., UvA).** Most eduGAIN users access cloud applications from a local network, for example, from the University of Amsterdam (UvA).

In order to authenticate against relying applications, eduGAIN users interact with the Federation Provider through the SSL reverse proxy. The proxy retrieves its configuration (SSL certificates, reverse IPs, etc.) from the local file system and relays traffic to the Keycloak identity broker as well as the SimpleSamlPHP bridge. On the Federation Provider VM, the Keycloak identity broker handles the main functionality, for example, creating and signing JWTs, attribute mapping, as well as implementing the OpenID Connect Authentication Code Flow. It is accompanied by the SimpleSamlPHP bridge, which offers SAML-based eduGAIN federation. The identity broker persists local users (FP DevOps / Admins) as well as its configuration from a local H2 database. Active Federation Provider user sessions are also persisted there. According to OpenID Connect, relying applications initiate authentication requests, which cause eduGAIN users to authenticate against their local identity providers. Those providers return SAML assertions to the SimpleSamlPHP bridge, which relays them to the identity broker. It returns authentication codes in response to the authentication requests that allow the relying applications to query the identity broker for user information.

4.6.2.2 Steps 2 and 3: Determining Threats and Countermeasures

Tables 4.13, 4.14, and 4.15 contain a list of threats, their possible causes as well as a mitigation strategy. These were modeled systematically based on the information derived from the system in Step 1. Furthermore, the tables take up the OWASP Treat Risk Modelling recommendations from [179] to include the level of attacker to defend against.

The threat analysis uses the DREAD classification scheme for "quantifying, comparing and prioritizing the amount of risk presented by each evaluated threat". DREAD was first proposed by Microsoft in [106] and proposes to classify security threats into five categories:

1. **D**amage potential: "How great is the damage if the vulnerability is exploited?"
2. **R**eproducibility: "How easy is it to reproduce the attack?"
3. **E**xploitability: "How easy is it to launch an attack?"
4. **A**ffected users: "As a rough percentage, how many users are affected?"
5. **D**iscoverability: "How easy is it to find the vulnerability?"

The designated ratings 1, 2, and 3 for each of the categories follow the definition laid out in the Threat Rating Table [106, fig. 3.6]. These ratings are summed up to a 1-15 rating allowing to categorize threats in low (1-9), medium (10-12), and high (13-15) priority. As the main security facilities of the Federation Provider rely on the OpenID Connect protocol and its flawless implementation, it is hard to differentiate between Federation Provider-specific threats and threats originating from the protocol. Therefore, the last column in the threat table provides a distinction into threats that are either more implementation-related (I) or more design-related (D).

Threat	Cause	Mitigation	Accidential	Malware	Script Kiddie	Curious A.	Motivated A.	Criminals	Damage Pot.	Reproducibilit	Exploitability	Nr. Affected	Discoverabilit	DREAD	Hi/Me/Lo	Des. / Impl.
Impersonation of any user to relying applications	Stolen private key for Keycloak JWT certificate.	Deployment on hardened VM.		X	X	X	X	X	3	3	2	3	2	13	H	D
	Broken encryption through too little entropy.	Use of high-quality entropy sources.					X	X	3	1	1	3	1	9	L	D
Impersonation of eduGAIN user to relying applications	Applications accept tokens, without verifying them with the FP. Tokens could be manufactured, or swapped with different issuers or subjects.	In OIDCACF, user agents do not supply tokens, but codes, which only the application can transform into tokens. Therefore, applications should never allow user agents to send them tokens, but only use (possibly encrypted) sessions.					X	X	3	3	3	3	3	15	H	D
	Server impersonation to client	Every client should validate the SSL connection to the FP and also received tokens using a shared secret or PKI certificates.					X	X	3	3	2	3	3	14	H	D

Figure 4.13: Threat List 1 of 3

Threat	Cause	Mitigation	Accidental	Malware	Script Kiddie	Curious A.	Motivated A.	Criminals	Damage Pot.	Reproducibilit	Exploitability	Nr. Affected	Discoverabilit	DREAD	Hi/Me/Lo	Des. / Impl.
Impersonation of DevOps and Admins	Password guessing by brute force	Strong password policy, Fail2Ban integration, Increasing Hash Iterations, and possibly enforcing Keycloak two-factor authentication with time-based one-time pass.				X	X	X	3	3	3	3	3	15	H	D
	Clickjacking on Keycloak login page	Usage of X-Frame-Options and Content-Security-Policy by Keycloak to protect against clickjacking.				X	X	X	3	3	2	3	2	13	H	I
Leaking user information (user name, email address, user organisation, user affiliation [staff, student, ...])	Access Token Redirect	Access Token should be audience and scope restricted, Keycloak has to validate audience and scope.	X	X	X	X	X	X	2	3	2	3	1	11	M	I
	Stolen access token by Man-in-the-Middle attack	Enforce SSL on all connections, use trusted server certificates, use secure networks if possible.					X	X	2	3	1	3	1	10	M	I
	Compromised / replayed keycloak access code	Cryptographically strong random value prevents guessing. Access codes cannot be reused, preventing replay attacks. Also, the lifetime of access codes is very short.					X	X	2	1	1	3	1	8	L	I

Figure 4.14: Threat List 2 of 3

Threat	Cause	Mitigation	Accidential	Malware	Script Kiddie	Curious A.	Motivated A.	Criminals	Damage Pot.	Reproducibilit	Exploitability	Nr. Affected	Discoverabilit	DREAD	Hi/Me/Lo	Des. / Impl.
Revealing of sensitive information (e.g., what service is accessed, which client is used, ...)	Plaintext OpenID Connect requests	Encryption of OpenID Connect requests	X	X	X	X	X	X	2	3	3	3	3	14	H	D
Misuse of Keycloak as Open Redirector	No redirect URL validation	Only allow specific redirect URLs for each client.	X	X	X	X	X	X	1	3	3	1	3	11	M	D
Client abuse	Server response repudiation, e.g., by not signing the response.	Response signing with non-repudiatable key by server. Enforced signature validation on client.					X	X	1	3	3	3	3	13	H	D
Server abuse	Client request repudiation, e.g., when clients do not sign their request	Request signing with non-repudiatable key by client. Enforced signature validation on server.					X	X	1	3	3	3	3	13	H	D
Further exposure to security and privacy threats	Request disclosure	Encryption of OpenID Connect request in an encrypted JWT within the request and request_uri parameter	X	X	X	X	X	X	1 - 3	3	3	3	3	13 - 15	H	D
	CSRF attack on Keycloak and OpenID Connect	Oauth 2.0 and Keycloak state cookies are matched against transmitted state parameter.					X	X	1 - 3	2	2	1	1	7 - 9	L	I

Figure 4.15: Threat List 3 of 3

Besides all the other threats, two can be seen as most important. The first is a brute-force attack on administrators' passwords, allowing an attacker to gain quite extensive control over the Federation Provider. The second threat is the lack of correct JSON Web Token validation by cloud applications. This threat was highlighted by McLean in his famous blog article [105] which revealed critical vulnerabilities in many JWT libraries allowing attackers to bypass the verification step. In order to better evaluate the security of popular libraries, Auth0 provides jwt.io, a website containing an overview about the supported JWT verification steps in different programming libraries and platforms.

4.6.3 Economic Benefits of the Security Architecture

This subsection discusses the economic benefits of the security architecture, contrasting them with the required upfront efforts. These efforts can be reduced by preparing ready-to-deploy modules, practical demos, as well as comprehensive documentation. However, not every user will realize all of the benefits as there are a number of impacting factors, for example, how many relying applications there are and how well the users are versed in the technologies.

Significantly Reduced Client Registration Efforts
Once the Federation Provider is initially set-up and registered in eduGAIN, further applications are registered in minutes instead of weeks. Before introducing the Federation Provider, registering every cloud application instance with eduGAIN was simply not feasible for a large number of applications: first, the process itself is manual and can take days to complete. In fact, registering the first Federation Provider instance took weeks, a duration deemed typical by other project partners that have completed such an undertaking before. Second, eduGAIN requires publishing every Service Provider's metadata[22]. Adding every cloud application instance would enlarge this document considerably, raising memory and processing requirements for all eduGAIN participants. In contrast, registration of new OpenID Connect clients at the Federation Provider is a straightforward process: logging into the administrative interface and entering the details of the new client. This OpenID Connect client registration effort are further reduced by relying on the self-service registration capabilities implemented by Berdonces-Bonelo [21].

Simplified Technology Stack
Using OpenID Connect libraries and handling JWTs proves far more easier than using the Shibboleth SP and SAML; of course, after investing effort in learning the technologies. This observation is based on experience as well as the observations of the use case partners. Reasons include: the token format is simpler, the documentation is more abundant and comprehensive, there is a larger number of libraries available for a wider range of platforms, and the protocols and data formats are less complex.

Easier Integration of Additional Identity Sources
After setting up the Federation Provider and integrating all relying applications through OpenID Connect, the extensive identity brokering available at the

[22]Currently 1206 SPs, see https://technical.edugain.org/entities

Federation Provider saves the effort of integrating another identity source into each application. For example, Keycloak supports LDAP, Google, Facebook, Twitter, GitHub, LinkedIn, Microsoft, and Stack Overflow as identity sources. The economic benefit of using the Federation Provider as a kind of authentication proxy are most extensive if there are a large number of applications in need of this functionality, as it needs to be integrated just once into the Federation Provider instead of *every* application.

Enhanced User Experience for both End-Users as well as Administrators
After the PAM module is installed and set-up, end users reuse their existing identities instead of learning about SSH, or remembering yet another credential. Furthermore, federated authorization management is very simple for administrators as they only need to modify a simple JSON file. Before CYCLONE, Bioinformaticians were required to either learn how to manage SSH keys or memorize yet another local account in order to access their VMs. Now, they can just reuse their existing federated identities, thus reducing identity management overheads and simplifying account management on the cloud provider's side. This effect is magnified when there are a large number of machines where the end users have access to. Permitting access to a VM is also very simple: VM owners just need to add the mail address of the other account to a certain file on the VM.

Easier Debugging of Multi-cloud Applications
Debugging distributed applications becomes far more easier after setting up the distributed logging and changing the configuration of relying applications. Additionally, merging application- as well as infrastructure log messages eases the debugging process considerably. As the logging middleware supports a large number of input sources, integrating applications is oftentimes as easy as changing some lines of a configuration file, for example, when using the popular Log4j Java library.[23]

4.6.4 Limitations

Even with all of the benefits of the security architecture, there are still limitations observed while implementing multi-cloud security. These are iterated in the following subsections.

Using Federated Identities in Non-Browser Scenarios.
OpenID Connect is best used in browser-based scenarios, for example, web single sign-on. For command line usage, the OpenID Connect Direct Access Grant was designed to query for a token directly, specifying the account name and password in the request. However, implementing this would require support by the EduGAIN Identity Providers that is not existing yet. For SAML 2.0 there is the "Enhanced Client Profile" (ECP) which was designed for enabling federated identities to be used on the command line, for example, for SSH login. However, while Shibboleth supports SAML 2.0 ECP, none of the 1,446 Identity Providers in EduGAIN support the "Reverse SOAP (PAOS) Binding" (urn:oasis:names:tc:SAML:2.0:bindings:PAOS) required for ECP.

[23]https://www.elastic.co/guide/en/logstash/current/plugins-inputs-log4j.html

Multi-cloud Account Management

SlipStream currently manages clouds on behalf of the users, persisting their credentials for later use, such as instantiating applications on different clouds. SlipStream could implement yet more management functions, for example, updating credit card details and analysing cloud invoices. However, it would be very challenging to implement this in a reliable and secure manner as no public cloud exposes those functions via APIs.

Federated Access Delegation

Another limitation is highlighted by the use of the IFB Bioinformatics Cloud self-service portal: as the resource quotas on the underlying IaaS solution are enforced based on user accounts, all portal end-users need to have their own account on the IFB cloud. This IaaS account is created on the fly by the portal whenever the end-users initiate their first deployment using a random username and password. However, the IaaS credentials are neither disclosed to the end-users nor linked to their federated identity. Ideally, there would be an auto-registration process at the IaaS cloud that end-users could use with their federated identities to create accounts and delegate access to their accounts to the portal. Complicating the scenario, SlipStream needs to be also integrated in this workflow.

PAM Module Deployment Issues

The deployment in the CYCLONE testbed at CNRS LAL highlights some of the issues that could possibly be observed in other clouds as well. First of all, the firewall at the LAL cloud restricts TCP/UDP access to only two ports at maximum. This required modification of the module to replace the random allocation of ports for the internal web server with a configuration option to specify a singular port that is always used. While not required by the use case, this issue prevents login by multiple users concurrently.

The second issue is caused by the very low use of the underlying `python_-pam` module by the Internet community. The module is included in the Ubuntu and Debian Linux repositories, however it is not provided for CentOS which is used at IFB. CentOS requires both a small source code patch as well as further modification of certain configuration files, for example, the SSH daemon configuration.

Chapter 5

Summary and Outlook

This thesis presents the analysis of real-world stakeholder requirements to provide a clear design rationale for the subsequent implementation of a number of cloud components. Their thorough validation reveals considerable advantages when they are used to tackle contemporary cloud challenges in production settings. They are integrated well with contributions of other project partners, providing the foundation for the two cloud ecosystems TRESOR and CYCLONE. At last, the dissertation not only observes interworking contemporary technologies. It also provides readily instantiable software under an open source license so that practitioners and researchers can directly benefit from their use and examination.

This section first provides answers to the research questions and summarizes key takeaways for cloud stakeholders and researchers. As a conclusion, it summarizes current limitations and provides starting points for future work.

5.1 Answers to the Research Questions

The preceding thesis' content contains detailed elucidations how the research questions were addressed that were raised in the introduction. The following paragraphs contain a summary of the thesis' answers to the research questions:

1. Using the proposed service registry architecture, cloud services can be discovered, assessed, and selected easily while following a *user-centric philosophy*.

Without relying on any previous approach, the thesis presents a novel service registry architecture whose design centers around three essential features, setting it apart from the related work: first, instead of proposing a generic solution for a range of use cases, the registry architecture was built from scratch to meet real-world requirements of concrete stakeholders in practical use cases, leaving out features that were neither required nor essential for the target environment.

Secondly, instead of relying on the technology with the highest sophistication or the largest feature set, the approach focuses on simplicity, adaptability, and usability by the stakeholders. For example, the service descriptions in the registry build upon a text-based "next generation" domain specific language (SDL-NG) as well as structured vocabularies that capture the essential properties of service offerings most pertinent to the specific use case stakeholders. The architecture does not rely on semantic technologies, such as ontology tools and automated reasoners, as they are little-known to the stakeholders, who additionally favor a more simple tooling.

At last, the service registry implementations are deployed to near-production environments and are evaluated in practice by prospective users. This evaluation provides feedback to optimize the developments. These three aspects form a *user-centric philosophy*, which means that the business needs of concrete users are set at the center of the work when designing, implementing, and validating the architecture and its constituents.

2. Using TCTP, HTTP entity-bodies can be secured end-to-end in a dependable and well-performing manner, even through HTTP intermediaries.

This thesis introduces the novel TCTP protocol which provides an HTTP-compatible wrapper around TLS to enable end-to-end security for HTTP entity-bodies. As the HTTP headers are unencrypted, HTTP intermediaries, such as the distributed cloud proxy, can use their plaintext information for management functions, such as routing, load balancing, and application-layer firewall protection. The evaluation of the TCTP implementations shows beneficial performance characteristics, as the underlying TLS libraries have been highly optimized for efficiency. Furthermore, the design of TCTP addresses many existing shortcomings in other approaches, for example, message-flow vulnerabilities and the prevention of HTTP streaming applications.

3. When cloud proxies are distributed between stakeholders and integrated with other ecosystem components, they provide secure and compliant cloud service consumption.

The concepts brought forward in this thesis envision cloud proxies that are distributed between the users, ecosystem operators, and service providers. They should be integrated with other cloud ecosystem components and also leverage the capabilities of TCTP. Through the implementation of such a proxy and its integration with TRESOR, this thesis demonstrates wholistic cloud consumption management that is secure and compliant to security policies and company regulations.

4. The multi-cloud security architecture supports application deployment, management, and the utilization of federated identities.

This dissertation establishes an economical and comprehensive security architecture that is readily instantiable, pertinent to concrete users' requirements, and relying upon up-to-date protocols and software. The feasibility of the architecture is highlighted by applying it within the CYCLONE ecosystem, deploying federated Bioinformatics applications within a cloud production environment. At last, special emphasis is put on the reduced management efforts in order to highlight the economic benefit of the architecture.

5.2 Key Takeaways for Cloud Stakeholders and Researchers

As the guiding idea of this thesis is addressing significant cloud challenges, there are key takeaways for the involved stakeholders as well as other researchers:

Enhanced Cloud Service Description, Discovery, Assessment, and Selection

The thesis provides a set of interacting software components that should help diverse cloud stakeholders in their quest for easy and usable cloud service description, discovery, assessment, and selection.

First of all, the provided information systems, for example, the TRESOR Service Broker and the Open Service Compendium, offer many helpful functions to *cloud service users* for better service discovery, assessment, and selection. Additionally, the customization of the TRESOR Service Broker allows *cloud ecosystem operators* to globally coordinate the requirements of the ecosystem users with the capabilities of the ecosystem services by offering a consistent service description language and comprehensive brokering functions. As all developments are provided as immediately usable open source components, new service registries are rapidly instantiable.

Service providers can use the business vocabularies to streamline the public descriptions of their services by focussing on the most important aspects for service selection and thus provide highly appropriate information to potential customers. *Cloud researchers* are able to contrast the practical knowledge gained through this thesis with rather theoretical research considerations to better understand how their work can be applied in practical settings. This should improve the business pertinence of proposals related to the topics of this thesis.

Improved Cloud Communication Security

TCTP offers transparent end-to-end encryption for HTTP communication. Within TRESOR, it has demonstrated considerably raising the security level of cloud service consumption. Especially in highly sensitive areas, such as the health sector, *Cloud service users* benefit greatly from an additional protection layer for their data. *Cloud ecosystem operators* and *cloud service providers* can use the heightened security to protect ecosystems and services against data leaks and also raise the trustfulness of their offerings.

The transparent manner in which TCTP operates helps *DevOps Engineers* to provide simpler solution architectures that rely on HTTP proxies encrypting transmissions instead of rather complex alternative technologies, such as SOAP with XML Encryption & Signature. It would be quite helpful if *cloud security researchers* would analyze the protocol to thoroughly model its security characteristics and also reveal potentially hidden security flaws.

Reliable and Policy-compliant Cloud Consumption

Through its diverse modules, the distributed cloud proxy offers a wide range of functionality to manage the consumption of cloud services in a reliable and policy-compliant manner, benefiting all ecosystem stakeholders. *Cloud service users* profit notably from the single sign-on module which simplifies the authentication within cloud ecosystems, especially when consuming many different services within a single login session. They also gain reliable policy enforcement through the XACML-based distributed authorization module, which ensures service consumption that is compliant to their company regulations and other legal frameworks.

Cloud ecosystem operators mostly benefit from the integration of the proxy with their backend systems, possibly through custom modules. In TRESOR, for example, the integration of the proxy with the service broker provided coordination of service consumption with booking and billing workflows. Furthermore, a combination of the remote logging module with a logging system is useful for many stakeholders, for example, *cloud service users* obtain a comprehensive audit trail while *DevOps engineers* of operators and providers can use the combined log output to better troubleshoot their services.

Having such a management proxy also supports diverse research endeavours. For example, Zickau et al. show how the distributed cloud proxy is used in TRESOR to enable location-based policies in [194].

Simplified and Secure Multi-cloud Application Deployment

The multi-cloud security architecture depicted in this dissertation simplifies the required activities for the secure deployment of multi-cloud applications as well as their management. In this, it materializes advantages for all involved stakeholders.

First of all, the management of accounts and credentials of *cloud service users* and *DevOps engineers* is eased, as the CYCLONE Federation Provider and the PAM module allow using a sole identity for many different purposes, for example, logging into websites or administering systems through SSH. Provisioning and deprovisioning workflows at *cloud ecosystem operators* also condense considerably with each saved account - even more so when new customers reuse existing company accounts for their ecosystem access, as demonstrated in TRESOR. A reduced number of provided accounts furthermore reduces support overheads at *cloud service providers*.

By following the application deployment procedure that is implemented by Nuv.la, many tasks of *DevOps engineers* are supported well, for example, scaling applications over multiple clouds. This potentially increases the resiliency and lowers the latency of cloud services, thus increasing their perceived quality of experience for potential customers.

The usefulness of capabilities for unified logging of distributed applications has already been observed in TRESOR. It is further amplified through the multi-cloud security architecture, especially when considering large topologies that span multiple clouds. There, *service providers* benefit from the reduced efforts for managing and troubleshooting cloud services.

At last, *researchers* also profit from taking up the architecture. The bioinformatics use case of CYCLONE exhibits advantages when relying on the architecture, for example, the simplified collaboration between researchers and the improved utilization of research infrastructures.

5.3 Limitations

The work presented in this thesis is mainly limited by the conditions, resources, and stakeholders of the research projects that served as its application area. For example, the cloud service registry architecture has proven quite successful to implement the requirements of TRESOR and the other use cases. However, all of them are still quite limited in depth, as typical for research projects. Realizing the far-reaching vision of the Open Service Compendium requires much more time and effort than available at the moment. Limited resources and changed circumstances were also the reason for the lack of progress on the CYCLONE IaaS registry which could not be realized within the duration of CYCLONE.

While benchmarks for the TRESOR proxy and TCTP are quite good, production environments not only require high performance, but also mature software with possibly larger functional scope. As the YAGNI principle was followed while implementing the components, functionality is missing that was not required in the current use cases. This includes, for example, OpenID Connect integration of the distributed cloud proxy or the implementation of TCTP as a JavaScript library to be run in the web browser. However, as there is currently no project supporting the extension and maintenance of the components, potentially required functions for a practical use can only be implemented by the future users of the software.

It is obvious that the design constraints of a software dictate its eventual capabilities. Thus, each potential future use case needs to separately evaluate if it shares similar constraints than those used in the design of each component. Therefore, potential benefits of using the created components can only be realized in scenarios where the stakeholder requirements are similar to those recognized by TRESOR and CYCLONE.

5.4 Future Work

Three main areas provide ample opportunity to perpetuate the work of this thesis:

General Data Protection Regulation (GDPR)

May 2018 marks the beginning of new EU-wide regulations regarding data protection. Key changes are increased territorial scope, far higher penalties, and the requirement for clear and always withdrawable consent.[1] The results of this thesis can help adhering to these regulations. For example, the end-to-end security provided by TCTP contributes towards “privacy by design”, a requirement of the GDPR. The XACML policies that are enforced by the cloud proxy allow querying external information systems when consuming services. One of such Policy Information Points (PIPs) could persist the users’ consent or opposition for processing of their personal data for specific purposes. Such a GDPR PIP would prevent any personal data processing in a cloud ecosystem that is not in accordance with the declared intention of the affected users. At last, some GDPR-related demands have already been implemented in the multi-cloud security architecture, for example, removing personal data after the end of the login session. Nevertheless, a more detailed analysis of the impacts of the GDPR on the deployment of the thesis’ developments constitutes future work.

[1]https://www.eugdpr.org/key-changes.html

Decentralization: Fog / Edge Computing, Distributed Ledger (Blockchain)

Besides centralized cloud computing, there is an ongoing drive towards decentralized architectures. The concept of Fog / Edge Computing, for example, seeks to move computing to the edges of the network, forming a highly distributed and resilient computing platform. One example is the Cloudy community cloud distribution, presented in [13], which is used within the guifi.net community network, providing computing resources at the network edge. In fact, a potentially beneficial deployment of the user-centric cloud service registry has already been discussed with one of the co-authors of the article. Additionally, distributing the proxy at the network edges could form an extra management layer that would unify certain functions, for example, edge services login, monitoring, and logging.

Peer-to-peer distributed ledger technologies, such as the Blockchain powering Bitcoin, prove to be quite disruptive for computing in general. It would be quite useful to transfer the benefits that have been achieved for multi-cloud application deployment to the area of peer-to-peer application deployment and management. Also, user-centric Blockchain registries would be able to capture the specific characteristics of each chain and help distributed application developers to make the right technology choice.

Intercloud Computing

The cloud computing paradigm is continually evolving towards the Intercloud, which is envisioned to be an interconnected "cloud of clouds". This evolution commences new research addressing the challenges in Intercloud scenarios. As a starting point, together with Demchenko et al., [41] extends the multi-cloud security architecture to form an "Intercloud Security Framework".

The other challenges also require extended solutions. For example, Intercloud service brokers could be even more utilized than cloud brokers as the targeted challenge is amplified as, for example, discovering services in the Intercloud could pose far more difficult. Intercloud communication topologies are potentially also far more complex, highlighting the need for secure end-to-end communication, possibly provided by an "Trusted Intercloud Communication Protocol" (TICP). At last, addressing the challenges related to the management of Intercloud service consumption and the secure deployment of Intercloud applications constitutes a consequential continuation of the activities of this thesis.

Chapter 6

Bibliography

[1] Assad Abbas and Samee U Khan. "A review on the state-of-the-art privacy-preserving approaches in the e-health clouds". In: *IEEE Journal of Biomedical and Health Informatics* 18.4 (2014), pp. 1431–1441.

[2] Mathieu Acher et al. "FAMILIAR: A domain-specific language for large scale management of feature models". In: *Special section: The Programming Languages track at the 26th ACM Symposium on Applied Computing (SAC 2011) & Special section on Agent-oriented Design Methods and Programming Techniques for Distributed Computing in Dynamic and Complex Environments* 78.6 (2013), pp. 657–681. ISSN: 0167-6423. DOI: 10.1016/j.scico.2012.12.004.

[3] R. Akkiraju et al. *Web Service Semantics - WSDL-S: W3C Member Submission*. 2005. URL: http://www.w3.org/Submission/WSDL-S.

[4] R. Akolkar et al. "The Future of Service Marketplaces in the Cloud". In: *2012 IEEE 8th World Congress on Services*. 2012, pp. 262–269. DOI: 10.1109/SERVICES.2012.59.

[5] Luna Alrawas. "Simple Cloud Service Selection in the Open Service Compendium Based on Dynamic Questionnaires and Property Statistics". Bachelor thesis. Berlin: Technische Universität Berlin, 2016. URL: http://www.snet.tu-berlin.de/fileadmin/fg220/theses/archive/Alrawas_2016_-_Simple_Cloud_Service_Selection.pdf.

[6] Jörn Altmann and Mohammad Mahdi Kashef. "Cost model based service placement in federated hybrid clouds". In: *Future Generation Computer Systems* 41 (2014), pp. 79–90. ISSN: 0167-739X.

[7] Amazon Web Services, Inc. *Elastic Load Balancing: Cloud-Load Balancer*. 2016. URL: http://aws.amazon.com/elasticloadbalancing/ (visited on 11/15/2016).

[8] Apache Software Foundation. *Apache JMeter*. 2016. URL: http://jmeter.apache.org/ (visited on 11/10/2016).

[9] Michael Armbrust et al. “A view of cloud computing”. In: *Commun. ACM* 53.4 (2010), pp. 50–58. ISSN: 0001-0782. DOI: 10.1145/1721654.1721672.

[10] J. Aznar et al. “CNSMO: A Network Services Manager/Orchestrator tool for cloud federated environments”. In: *2016 Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*. 2016, pp. 1–5. DOI: 10.1109/MedHocNet.2016.7528422.

[11] J. Aznar et al. *Specification of network management and service abstraction. CYCLONE Deliverable D5.2*. 2016. URL: http://www.cyclone-project.eu/assets/images/deliverables/Specification%20of%20network%20management%20and%20service%20abstraction.pdf (visited on 01/06/2017).

[12] R. Baig et al. “Cloud-based community services in community networks”. In: *International Conference on Computing, Networking and Communications (ICNC 2016)*. 2016, pp. 1–5. DOI: 10.1109/ICCNC.2016.7440621.

[13] Roger Baig, Felix Freitag, and Leandro Navarro. “Cloudy in guifi.net: Establishing and sustaining a community cloud as open commons”. In: *Future Generation Computer Systems* (2018). ISSN: 0167-739X. DOI: 10.1016/j.future.2017.12.017.

[14] Mark Bartel et al. *XML-Signature Syntax and Processing*. 2001. URL: http://www.w3.org/TR/2001/PR-xmldsig-core-20010820/.

[15] Steve Battle et al. *Semantic Web Services Framework (SWSF)*. 2005. URL: http://www.w3.org/Submission/SWSF/.

[16] Kent Beck. *Extreme programming explained: embrace change*. Addison-Wesley Longman Publishing Co., Inc, 1999. ISBN: 0-201-61641-6.

[17] Alexander Becker. “Linked and Semantically Enriched Crowdsourced Cloud Computing Location Data”. Bachelor thesis. Berlin: Technische Universität Berlin, 2015. URL: http://www.snet.tu-berlin.de/fileadmin/fg220/theses/archive/Becker_2015_-_Linked_and_Semantically_Enriched_Crowdsourced.pdf.

[18] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. “Keying Hash Functions for Message Authentication”. In: *Advances in Cryptology CRYPTO ’96*. Ed. by Neal Koblitz. Vol. 1109. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1996, pp. 1–15. ISBN: 978-3-540-61512-5. DOI: 10.1007/3-540-68697-5_1.

[19] E. Berdonces Bonelo. *xpra-electron-client*. 2017. URL: https://github.com/cyclone-project/xpra-electron-client (visited on 01/11/2017).

[20] E. Berdonces Bonelo and B. Brancotte. *pam_openid_connect module*. 2017. URL: https://github.com/cyclone-project/cyclone-python-pam (visited on 01/11/2017).

[21] Erik Berdonces-Bonelo. “OpenID Connect Client Registration API for Federated Cloud Platforms”. Master thesis. Berlin: Technische Universität Berlin, 2016.

[22] Sonia Bergamaschi et al. “Combining user and database perspective for solving keyword queries over relational databases”. In: *Inf. Syst.* 55 (2016), pp. 1–19.

[23] Tim Berners-Lee. *Linked Data - Design Issues*. 2006. URL: http://www.w3.org/DesignIssues/LinkedData.html.

[24] Tim Berners-Lee. *W3 future directions*. 1994. URL: https://www.w3.org/Talks/WWW94Tim/.

[25] David Bernstein et al. “Blueprint for the Intercloud - Protocols and Formats for Cloud Computing Interoperability”. In: *2009 Fourth International Conference on Internet and Web Applications and Services (ICIW 2009)*. Ed. by Institute of Electrical and Electronics Engineers. 2009, pp. 328–336. ISBN: 9781424438518. DOI: 10.1109/ICIW.2009.55.

[26] BITKOM. *Cloud Computing - Evolution in der Technik, Revolution im Business. BITKOM-Leitfaden*. 2009. URL: http://www.bitkom.org/files/documents/BITKOM-Leitfaden-CloudComputing_Web.pdf.

[27] S. Blank. *Perfection By Subtraction – The Minimum Feature Set*. 2010. URL: https://steveblank.com/2010/03/04/perfection-by-subtraction-the-minimum-feature-set (visited on 01/10/2017).

[28] Charles Branchat Roberta dn Loomis et al. *CYCLONE Deliverable D4.3: CYCLONE Secure Action and Resource Models*. Ed. by CYCLONE Project. 2016. URL: http://www.cyclone-project.eu/deliverables.html.

[29] I. Brandic et al. “Compliant Cloud Computing (C3): Architecture and Language Support for User-Driven Compliance Management in Clouds”. In: *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. 2010, pp. 244–251. DOI: 10.1109/CLOUD.2010.42.

[30] Ivan Breskovic, Jörn Altmann, and Ivona Brandic. “Creating standardized products for electronic markets”. In: *Future Generation Computer Systems* 29.4 (2013), pp. 1000–1011. ISSN: 0167-739X. DOI: 10.1016/j.future.2012.06.007.

[31] Wolfgang Bröcker and Joseph Walenta. *Experteninterview zum Thema "Motivation Cloud Computing im Gesundheitswesen: Krankenhausperspektive" mit Wolfgang Bröcker (Paulinenkrankenhaus Berlin, Leiter EDV) und Joseph Walenta (Deutsches Herzzentrum Berlin, IT-Projektleiter)*. Apr. 2012. URL: http://www.cloud-tresor.de/2012/05/14/experteninterview.

[32] Liliana Cabral et al. “IRS-III: A Broker for Semantic Web Services Based Applications”. In: *The semantic Web*. Ed. by Isabel Cruz. Vol. 4273. Lecture Notes in Computer Science. Berlin: Springer, 2006, pp. 201–214. ISBN: 978-3-540-49055-5. DOI: 10.1007/11926078_15.

[33] Carnegie Mellon University. *CSMIC: Cloud Service Measurement Initiative Consortium*. 2012. URL: http://csmic.org/.

[34] Daniele Catteddu and Giles Hogben. *Cloud Computing Benefits, risks and recommendations for information security*. ENISA Europeam Network and Information Security Agency, 2009. URL: http://www.enisa.europa.eu/.

[35] Lingfeng Chen and D. B. Hoang. “Novel Data Protection Model in Healthcare Cloud”. In: *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*. 2011, pp. 550–555. DOI: 10.1109/HPCC.2011.148.

[36] Tzer-Shyong Chen et al. "Secure Dynamic Access Control Scheme of PHR in Cloud Computing". In: *Journal of Medical Systems* 36.6 (2012), pp. 4005–4020.

[37] SUPER Consortium. *SUPER: Semantics Utilised for Process management within and between EnterPrises*. 2009. URL: http://cordis.europa.eu/project/rcn/105285_en.html.

[38] TRESOR project consortium. *TRESOR - Sichere Cloud-Infrastruktur für das Gesundheitswesen*. Presentation at the closing event of the Trusted Cloud Programme. 2015. URL: http://www.aktionsprogramm-cloud-computing.de/media/content/02-16-Frank.pdf.

[39] Y. Demchenko et al. "Federated Access Control in Heterogeneous Intercloud Environment: Basic Models and Architecture Patterns". In: *Proceedings of IC2E 2014*. 2014.

[40] Y. Demchenko et al. "Intercloud Architecture Framework for Heterogeneous Cloud Based Infrastructure Services Provisioning On-Demand". In: *2013 27th International Conference on Advanced Information Networking and Applications Workshops*. 2013, pp. 777–784. DOI: 10.1109/WAINA.2013.237.

[41] Yuri Demchenko et al. "Defining Intercloud Security Framework and Architecture Components for Multi-Cloud Data Intensive Applications". In: *2017 6th Intercloud 2017 Workshop as part of CCGrid 2017*. 2017.

[42] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246 (Proposed Standard). Updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685, 7905, 7919. Internet Engineering Task Force, Aug. 2008. URL: http://www.ietf.org/rfc/rfc5246.txt.

[43] Derek Du Preez. *A CIO's worst nightmare: When your cloud provider goes bankrupt*. 2015. URL: http://diginomica.com/2015/01/06/cios-worst-nightmare-cloud-provider-goes-bankrupt/.

[44] Nicole Dufft. *Reality Check Cloud Computing 2012: Wirklichkeit oder Wolkenkuckucksheim?* 2012. URL: http://www.cloud-practice.de/sites/default/files/downloads/live/07_1000_dufft_pac.pdf.

[45] Electronic Frontier Foundation. *HTTPS Everywhere*. 2013. URL: https://www.eff.org/https-everywhere.

[46] Daniel Elenius et al. "The OWL-S Editor: A Development Tool for Semantic Web Services". In: *Proceedings of the Second Semantic Web Conference, ESWC 2005*. Ed. by Asunción Gómez-Pérez and Jérôme Euzenat. Vol. 3532. Lecture Notes in Computer Science. Heidelberg: Springer, 2005, pp. 78–92. ISBN: 3-540-26124-9. URL: http://www.csl.sri.com/papers/owlseditor-eswc05/.

[47] Amna Eleyan and Liping Zhao. "Service Selection Using Quality Matchmaking". In: *International Conference on Communications and Information Technology (ICCIT), 2011*. IEEE. 2011, pp. 107–115.

[48] Eric Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2003. ISBN: 978-0-321-12521-7.

[49] Joel Farrell and Holger Lausen. *Semantic Annotations for WSDL and XML Schema: W3C Recommendation*. 2007. URL: https://www.w3.org/TR/2007/REC-sawsdl-20070828.

[50] Jacob Feldman. *JSR 331: Constraint Programming API, Version: 1.0.0*. 2012. URL: https://jcp.org/en/jsr/detail?id=331.

[51] Dieter Fensel et al. *Semantic Web Services*. Heidelberg and New York: Springer, 2011. ISBN: 3642191932.

[52] R. Fielding and J. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. RFC 7230 (Proposed Standard). Internet Engineering Task Force, June 2014. URL: http://www.ietf.org/rfc/rfc7230.txt.

[53] R. Fielding and J. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. RFC 7231 (Proposed Standard). Internet Engineering Task Force, June 2014. URL: http://www.ietf.org/rfc/rfc7231.txt.

[54] Roy Thomas Fielding. "Architectural Styles and the Design of Network-based Software Architectures". Dissertation. Irvine, California: University of California, 2000. URL: http://www.ics.uci.edu/%7Efielding/pubs/dissertation/top.htm (visited on 11/09/2016).

[55] FIWARE. *Apps and Services Ecosystem goes Open Source*. 2012. URL: https://www.fiware.org/2012/11/23/apps-and-services-ecosystem-goes-open-source/ (visited on 03/09/2016).

[56] Paul Ford. *A Response to Clay Shirky's "The Semantic Web, Syllogism, and Worldview" (Ftrain.com)*. 2003. URL: http://www.ftrain.com/ContraShirky.html (visited on 08/19/2014).

[57] fortiss GmbH. *CloudServiceCheck*. 2014. URL: http://www.value4cloud.de/de/cloudservicecheck.

[58] fortiss GmbH. *Value4Cloud*. 2014. URL: http://www.value4cloud.de.

[59] Martin Fowler. *Domain-Specific Languages*. Addison-Wesley, 2011. ISBN: 0-321-71294-3.

[60] Stephan Fowler. *HTTPsec: Public key authentication for HTTP*. 2006. URL: https://web.archive.org/web/20100926084623/http://www.httpsec.org/ (visited on 11/15/2016).

[61] A. Freier, P. Karlton, and P. Kocher. *The Secure Sockets Layer (SSL) Protocol Version 3.0*. RFC 6101 (Historic). Internet Engineering Task Force, Aug. 2011. URL: http://www.ietf.org/rfc/rfc6101.txt.

[62] GÉANT. *eduGAIN*. 2017. URL: http://www.geant.org/Services/Trust_identity_and_security/eduGAIN (visited on 01/11/2017).

[63] Martin Georgiev et al. "The most dangerous code in the world: validating SSL certificates in non-browser software". In: *Proceedings of the 2012 ACM conference on Computer and communications security*. CCS '12. New York, NY, USA: ACM, 2012, pp. 38–49. ISBN: 978-1-4503-1651-4. DOI: 10.1145/2382196.2382204.

[64] German Federal Ministry of Justice. *StGB § 203: Verletzung von Privatgeheimnissen*. 2013. URL: http://www.gesetze-im-internet.de/stgb/__203.html.

[65] GitHub. *Electron - Build cross platform desktop apps with JavaScript, HTML, and CSS*. 2018. URL: http://electron.atom.io/ (visited on 01/18/2017).

[66] D. Gmach, J. Rolia, and L. Cherkasova. "Comparing efficiency and costs of cloud computing models". In: *Network Operations and Management Symposium (NOMS), 2012 IEEE*. 2012, pp. 647–650. DOI: 10.1109/NOMS.2012.6211977.

[67] Reyes Gonzalez, Jose Gasco, and Juan Llopis. "Information Systems Outsourcing Reasons and Risks: An Empirical Study". In: *Industrial Management and Data Systems* 110.2 (2009), pp. 284–303.

[68] Google. *Google Apps Marketplace*. 2014. URL: https://www.google.com/enterprise/marketplace/home/apps/?pli=1 (visited on 06/10/2014).

[69] T. Graf, S. Zickau, and A. Küpper. "Enabling Location-based Services on Stationary Devices using Smartphone Capabilities". In: *Mobile Web Information Systems*. Vol. 8093. Lecture Notes in Computer Science. Paphos, Cyprus: Springer, Aug. 2013, pp. 49–63. DOI: 10.1007/978-3-642-40276-0_5.

[70] Martin Gudgin et al. *SOAP Version 1.2 Part 1: Messaging Framework*. 2007. URL: http://www.w3.org/TR/soap12-part1 (visited on 11/15/2016).

[71] D. Hardt. *The OAuth 2.0 Authorization Framework*. Internet Engineering Task Force. Oct. 2012. URL: http://www.ietf.org/rfc/rfc6749.txt.

[72] Martin Hepp. *GoodRelations Language Reference*. 2011. URL: http://www.heppnetz.de/ontologies/goodrelations/v1.html#references.

[73] Martin Hepp. "GoodRelations: An Ontology for Describing Products and Services Offers on the Web". In: *Knowledge engineering: practice and patterns*. Ed. by Aldo Gangemi and Jérôme Euzenat. Vol. 5268. Lecture notes in computer science Lecture notes in artificial intelligence. Berlin: Springer, 2008, pp. 329–346. ISBN: 978-3-540-87696-0. DOI: 10.1007/978-3-540-87696-0_29.

[74] Martin Hepp and Roman Dumitru. "An Ontology Framework for Semantic Business Process Management". In: *Wirtschaftsinformatik Proceedings 2007*. ais, 2007, Paper 27. URL: http://aisel.aisnet.org/wi2007/27/.

[75] Alan R. Hevner et al. "Design science in information systems research". In: *MIS quarterly* 28.1 (2004), pp. 75–105.

[76] Giles Hogben and Alain Pannetrat. "Mutant Apples: A Critical Examination of Cloud SLA Availability Definitions". In: *5th International Conference on Cloud Computing Technology and Science (CloudCom)*. Ed. by IEEE. Vol. 1. IEEE, 2013, pp. 379–386. ISBN: 9781479915484. DOI: 10.1109/CloudCom.2013.56.

[77] IBM. *Watson*. 2014. URL: http://www.ibm.com/smarterplanet/us/en/ibmwatson.

[78] IBM Research. *The DeepQA Research Team*. 2013. URL: http://www.research.ibm.com/deepqa.

[79] Takeshi Imamura, Blair Dillaway, and Ed Simon. *XML Encryption Syntax and Processing*. 2002.

[80] Bala Iyer and John C. Henderson. “Preparing for the future: Understanding the seven capabilities of Cloud Computing”. In: *MIS Quarterly Executive* 9.2 (2010).

[81] jboss.org. *RESTEasy: Distributed peace of mind*. 2016. URL: http://resteasy.jboss.org (visited on 11/25/2016).

[82] Lim Yuan Jie, Rajaraman Kanagasabai, et al. “Dynamic Discovery of Complex Constraint-Based Semantic Web Services”. In: *Fifth IEEE International Conference on Semantic Computing (ICSC), 2011*. IEEE. 2011, pp. 51–58.

[83] M. Jones, J. Bradley, and N. Sakimura. *JSON Web Token (JWT)*. Updated by RFC 7797. Internet Engineering Task Force. May 2015. URL: http://www.ietf.org/rfc/rfc7519.txt.

[84] Kyo C. Kang et al. *Feature-Oriented Domain Analysis (FODA): Technical Report*. Ed. by Software Engineering Institute. Pittsburgh, Pennsylvania, USA, 1990.

[85] Mohammad Mahdi Kashef and Jörn Altmann. “A Cost Model for Hybrid Clouds”. In: *Economics of Grids, Clouds, Systems, and Services*. Ed. by Kurt Vanmechelen, Jörn Altmann, and Omer F. Rana. Vol. 7150. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 46–60. ISBN: 978-3-642-28674-2.

[86] Friederike Klan. “Effective Decision Support for Semantic Web Service Selection”. PhD thesis. Jena: Friedrich-Schiller-Universität, 2012.

[87] Fabian Knaack. “Towards an Open Service Compendium: Evaluation and Extension of Brokering Technologies”. Bachelor Thesis. Berlin: Technische Universität Berlin, 2015. URL: http://www.snet.tu-berlin.de/fileadmin/fg220/theses/archive/Knaack_2015_-_Towards_an_Open_Service_Compendium.pdf.

[88] Bastian Koller and Lutz Schubert. “Towards autonomous SLA management using a proxy-like approach”. In: *Multiagent Grid Syst* 3.3 (2007), pp. 313–325. ISSN: 1574-1702.

[89] Koordinierungsstelle für IT-Standards. *OSCI: Startseite*. 2013. URL: http://www.osci.de (visited on 11/15/2016).

[90] KPMG and BITKOM. *Cloud Monitor 2012*. Mar. 2012.

[91] Kyriakos Kritikos and Dimitris Plexousakis. “Mixed-Integer Programming for QoS-Based Web Service Matchmaking”. In: *IEEE Trans. Serv. Comput.* 2.2 (Apr. 2009), pp. 122–139. ISSN: 1939-1374. DOI: 10.1109/TSC.2009.10.

[92] Mary Lacity and Peter Reynolds. “Cloud Services Practices for Small and Medium-sized Enterprises”. In: *MIS Quarterly Executive*. Vol. 13:1. Minneapolis: Management Information Systems Research Center, 2014, pp. 31–44.

[93] Thomas Lacroix1 et al. “Insyght: navigating amongst abundant homologues, syntenies and gene functional annotations in bacteria, it’s that symbol!” In: *Nucleic Acids Research* (2014). DOI: 10.1093/nar/gku867.

[94] Christine Legner. “Is There a Market for Web Services?” In: *Service-Oriented Computing - ICSOC 2007 Workshops*. Ed. by Elisabetta Di Nitto and Matei Ripeanu. Vol. 4907. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 29–42. isbn: 978-3-540-93850-7. doi: 10.1007/978-3-540-93851-4_4.

[95] Torsten Leidig. *Simple editor for Linked USDL descriptions*. 2013. url: https://github.com/linked-usdl/usdl-editor.

[96] Stefanie Leimeister et al. “The Business Perspective of Cloud Computing: Actors, Roles and Value Networks”. In: *The European Conference on Information Systems (ECIS)*. 2010.

[97] Ming Li et al. “Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption”. In: *IEEE transactions on parallel and distributed systems* 24.1 (2013), pp. 131–143.

[98] Min Liu et al. “An Weighted Ontology-Based Semantic Similarity Algorithm for Web Service”. In: *Expert Syst. Appl.* 36.10 (Dec. 2009), pp. 12480–12490. issn: 0957-4174. doi: 10.1016/j.eswa.2009.04.034.

[99] Charles Loomis et al. *CYCLONE Deliverable D6.2: Specification of Interfaces for Brokering, Deployment, and Management*. Ed. by CYCLONE Project. 2015. url: http://www.cyclone-project.eu/deliverables.html.

[100] Luciano Floridi. “Web 2.0 vs. the Semantic Web: A Philosophical Assessment”. In: *Episteme* 6.01 (2009), pp. 25–37. issn: 1750-0117. doi: 10.3366/E174236000800052X.

[101] Ari Luotonen. *Tunneling TCP based protocols through Web proxy servers*. 1998. url: http://tools.ietf.org/html/draft-luotonen-web-proxy-tunneling-01.

[102] Robert P. Mahowald et al. *IDC FutureScape: Worldwide Cloud 2017 Predictions*. study. International Data Corporation, 2017.

[103] David Martin et al. “Bringing Semantics to Web Services: The OWL-S Approach”. In: *Semantic Web Services and Web Process Composition*. Ed. by Jorge Cardoso and Amit Sheth. Vol. 3387. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2005, pp. 26–42. isbn: 978-3-540-24328-1.

[104] Eyhab Al-Masri and Qusay H Mahmoud. *The QWS Dataset*. Oct. 2014. url: http://www.uoguelph.ca/%7Eqmahmoud/qws/ (visited on 10/10/2014).

[105] Tim McLean. *Critical vulnerabilities in JSON Web Token libraries. Which libraries are vulnerable to attacks and how to prevent them.* 2015. url: https://auth0.com/blog/critical-vulnerabilities-in-json-web-token-libraries/ (visited on 05/04/2017).

[106] J.D. Meier et al. *Threat Modeling - Rate the Threats*. 2003. url: https://msdn.microsoft.com/en-us/library/ff648644.aspx (visited on 05/03/2017).

[107] Peter Mell and Timothy Grance. *The NIST Definition of Cloud Computing*. Ed. by National Institute of Standards and Technology. 2011.

[108] Friedrich Merz. *Wachstumsmotor Gesundheit: Die Zukunft unseres Gesundheitswesens*. München: Carl Hanser Verlag, 2008.

[109] Microsoft Inc. *Transport Layer Security Protocol (Windows)*. 2016. URL: http://msdn.microsoft.com/en-us/library/aa380516(v=vs.85).aspx (visited on 11/25/2016).

[110] Tigran Mkrtchyan. “dCache: implementing a high-end NFSv4.1 service using a Java NIO framework”. In: *Computing in High Energy and Nuclear Physics (CHEP)* (2012).

[111] Delnavaz Mobedpour and Chen Ding. “User-centered Design of a QoS-based Web Service Selection System”. In: *Service Oriented Computing and Applications* (2013), pp. 1–11.

[112] MongoDB. *MongoDB Performance*. 2017. URL: https://docs.mongodb.com/manual/administration/analyzing-mongodb-performance/ (visited on 08/21/2017).

[113] Günter Müller et al. “Sustainable Cloud Computing”. In: *Business & Information Systems Engineering (BISE)* 5 (2011).

[114] Anthony Nadalin et al. *Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)*. 2006. URL: https://www.oasis-open.org/committees/download.php/21257/wss-v1.1-spec-errata-os-SOAPMessageSecurity.htm (visited on 11/15/2016).

[115] Mines Nantes. *Choco Solver*. Oct. 2014. URL: http://www.emn.fr/z-info/choco-solver/ (visited on 10/10/2014).

[116] Nordic APIs AB. *SOAP vs. REST: A NordicAPIs infographic*. 2016. URL: http://nordicapis.com/rest-vs-soap-nordic-apis-infographic-comparison (visited on 11/09/2016).

[117] Daniel Oberle et al. “A unified description language for human to automated services”. In: *Information Systems* 38.1 (2013), pp. 155–181. ISSN: 0306-4379.

[118] OpenID. *OpenID Connect*. 2016. URL: http://openid.net/connect (visited on 11/08/2016).

[119] OpenNaaS. *OpenNaaS CNSMO*. 2016. URL: http://opennaas.org/opennaas-cnsmo/ (visited on 01/06/2017).

[120] Oracle. *JSSE Reference Guide*. 2016. URL: http://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html.

[121] Oracle Technology Network. *Project Grizzly: NIO Event Development Simplified*. 2016. URL: https://grizzly.java.net/ (visited on 11/09/2016).

[122] Organization for the Advancement of Structured Information Standards. *OASIS eXtensible Access Control Markup Language (XACML)*. 2017. URL: https://www.oasis-open.org/committees/xacml.

[123] Sixto Jr. Ortiz. “The Problem with Cloud-Computing Standardization”. In: *Computer* 44.7 (2011), pp. 13–16.

[124] OSGi Alliance. *OSGi Alliance: The Dynamic Module System for Java*. 2016. URL: http://www.osgi.org (visited on 11/08/2016).

[125] Carlos Pedrinaci, Jorge Cardoso, and Torsten Leidig. “Linked USDL: A Vocabulary for Web-Scale Service Trading”. In: *The Semantic Web: Trends and Challenges*. Ed. by Valentina Presutti et al. Vol. 8465. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 68–82. ISBN: 978-3-319-07442-9. DOI: 10.1007/978-3-319-07443-6_6.

[126] Carlos Pedrinaci, Jorge Cardoso, and Torsten Leidig. *Presentation: Linked USDL: a Vocabulary for Web-scale Service Trading*. 2014. URL: http://slideshare.net/cpedrinaci/linked-usdl-a-vocabulary-for-webscale-service-trading.

[127] Carlos Pedrinaci and John Domingue. "Toward the Next Wave of Services: Linked Services for the Web of Data". In: *Journal of Universal Computer Science* 16.13 (2010). Ed. by J.UCS Consortium, pp. 1694–1719. DOI: 10.3217/jucs-016-13-1694.

[128] Olli-Pekka Pohjola and Kalevi Kilkki. "Value-based methodology to analyze communication services". In: *NETNOMICS: Economic Research and Electronic Networking* 8.1-2 (Oct. 1, 2007), p. 135. ISSN: 1573-7071. DOI: 10.1007/s11066-008-9013-2.

[129] Programmable Web. *Protocol usage by APIs*. 2012. URL: http://www.programmableweb.com/images/charts/TopProtocolsAlltime.png (visited on 11/09/2016).

[130] TOR Project. *Tor Project: Anonymity Online*. 2016. URL: https://www.torproject.org/ (visited on 11/11/2016).

[131] B. Ramsdell and S. Turner. *Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification*. RFC 5751 (Proposed Standard). Internet Engineering Task Force, Jan. 2010. URL: http://www.ietf.org/rfc/rfc5751.txt.

[132] P. Raschke and S. Zickau. "A Template-Based Policy Generation Interface for RESTful Web Services". In: *On the Move to Meaningful Internet Systems: OTM 2014 Workshops*. Vol. 8842. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, pp. 137–153. ISBN: 978-3-662-45549-4. DOI: 10.1007/978-3-662-45550-0_17.

[133] RedHat. *Keycloak Open Source Identity and Access Management*. 2018. URL: http://www.keycloak.org/ (visited on 01/09/2017).

[134] Mark Reinhold. *JSR-51: New I/O APIs for the Java Platform*. 2002. URL: http://www.jcp.org/en/jsr/detail?id=51 (visited on 11/09/2016).

[135] J. Repschläger et al. "Selection Criteria for Software as a Service: An Explorative Analysis of Provider Requirements". In: *Proceedings of the AMCIS 2012*. 2012, n/a.

[136] Jonas Repschläger et al. "Cloud Requirement Framework: Requirements and Evaluation Criteria to adopt Cloud Solutions". In: *Proceedings of the 20th European Conference on Information Systems*. Ed. by Jan Pries-Heje et al. 2012. ISBN: 978-84-88971-54-8.

[137] E. Rescorla. *HTTP Over TLS*. RFC 2818 (Informational). Updated by RFCs 5785, 7230. Internet Engineering Task Force, May 2000. URL: http://www.ietf.org/rfc/rfc2818.txt.

[138] E. Rescorla and A. Schiffman. *The Secure HyperText Transfer Protocol*. RFC 2660 (Experimental). Internet Engineering Task Force, Aug. 1999. URL: http://www.ietf.org/rfc/rfc2660.txt.

[139] Marcel Risch and Jörn Altmann. "Enabling Open Cloud Markets Through WS-Agreement Extensions". In: *Grids and Service-Oriented Architectures for Service Level Agreements*. Ed. by Philipp Wieder, Ramin Yahyapour, and Wolfgang Ziegler. Boston, MA: Springer US, 2010, pp. 105–117. ISBN: 978-1-4419-7319-1. DOI: 10.1007/978-1-4419-7320-7_10.

[140] V. Roberto et al. "A semantic web service-based architecture for the interoperability of e-Government services". In: *Web Information Systems Modeling Workshop (WISM 2005) / 5th International Conference on Web Engineering (ICWE 2005)*. The Open University, 2005, n/a. URL: http://oro.open.ac.uk/3005/.

[141] Miguel ángel Rodríguez-García et al. "Creating a semantically-enhanced cloud services environment through ontology evolution". In: *Special Section: The Management of Cloud Systems, Special Section: Cyber-Physical Society and Special Section: Special Issue on Exploiting Semantic Technologies with Particularization on Linked Data over Grid and Cloud Architectures* 32 (2014), pp. 295–306. ISSN: 0167-739X. DOI: 10.1016/j.future.2013.08.003.

[142] Juthasit Rohitratana and Jörn Altmann. "Impact of pricing schemes on a market for Software-as-a-Service and perpetual software". In: *Future Generation Comp. Syst.* 28.8 (2012), pp. 1328–1339. DOI: 10.1016/j.future.2012.03.019.

[143] Dumitru Roman et al. "Web Service Modeling Ontology". In: *Applied Ontology*. Vol. 1. IOS Press, 2005, pp. 77–106.

[144] Salesforce. *AppExchange*. 2014. URL: https://appexchange.salesforce.com (visited on 06/10/2014).

[145] Jonan Scheffler. *Ruby 3x3: Matz, Koichi, and Tenderlove on the future of Ruby Performance*. 2016. URL: https://blog.heroku.com/ruby-3-by-3 (visited on 08/17/2017).

[146] Annika Selzer. "Datenschutz bei internationalen Cloud Computing Services". In: *Datenschutz und Datensicherheit - DuD* 38.7 (2014), pp. 470–474. ISSN: 1862-2607.

[147] S. Sengupta, V. Kaulgud, and V. S. Sharma. "Cloud Computing Security–Trends and Research Directions". In: *IEEE World Congress on Services (SERVICES 2011)*. 2011, pp. 524–531. DOI: 10.1109/SERVICES.2011.20.

[148] Clay Shirky. *Shirky: The Semantic Web, Syllogism, and Worldview*. 2014. URL: http://www.shirky.com/writings/semantic_syllogism.html (visited on 08/19/2014).

[149] Alex Simov and Marin Dimitrov. *WSMO Studio*. 2008. URL: http://sourceforge.net/projects/wsmostudio/files.

[150] SixSq. *Nuvla*. 2017. URL: https://nuv.la (visited on 01/18/2017).

[151] M. Slawik et al. "CYCLONE: The Multi-Cloud Middleware Stack for Application Deployment and Management". In: *7th Workshop on Network Infrastructure Services as part of Cloud Computing (NetCloud 2017) in conjunction with CloudCom 2017*. 2017, pp. 347–352. DOI: 10.1109/CloudCom.2017.56.

[152] Mathias Slawik. “The Trusted Cloud Transfer Protocol”. In: *5th International Conference on Cloud Computing Technology and Science (CloudCom)*. Ed. by IEEE. Vol. 2. 2014. isbn: 9781479915484. doi: 10.1109/CloudCom.2013.126.

[153] Mathias Slawik. *TU-Berlin-SNET/tresor-proxy-prototype: The Grizzly NIO-based prototype of the TRESOR proxy*. 2012. url: https://github.com/TU-Berlin-SNET/tresor-proxy-prototype (visited on 11/09/2016).

[154] Mathias Slawik and Yuri Demchenko. *CYCLONE Deliverable D4.1: Security Infrastructure Specification and Initial Implementation*. Ed. by CYCLONE Project. 2015. url: http://www.cyclone-project.eu/deliverables.html.

[155] Mathias Slawik and Axel Küpper. “A Domain Specific Language and a Pertinent Business Vocabulary for Cloud Service Selection”. In: *Economics of Grids, Clouds, Systems, and Services*. Ed. by Jörn Altmann, Kurt Vanmechelen, and Omer F. Rana. Vol. 8914. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 172–185. isbn: 978-3-319-14608-9.

[156] Mathias Slawik et al. “An Economical Security Architecture for Multi-cloud Application Deployments in Federated Environments”. In: *Proceedings of the 13th International Conference on Grids, Clouds, Systems and Services*. Ed. by Jörn Altmann. Springer, 2016.

[157] Mathias Slawik et al. *CYCLONE Deliverable D4.2: Multi-cloud Security*. Ed. by CYCLONE Project. 2016. url: http://www.cyclone-project.eu/deliverables.html.

[158] Mathias Slawik et al. “CYCLONE: Unified Deployment and Management of Federated, Multi-Cloud Applications”. In: *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2015, pp. 453–457. doi: 10.1109/UCC.2015.81.

[159] Mathias Slawik et al. “Securing Medical SaaS Solutions Using a Novel End-to-end Encryption Protocol”. In: *ECIS 2014 proceedings*. Ed. by Michel Avital, Jan Marco Leimeister, and Ulrike Schultze. AIS Electronic Library, 2014. isbn: 9780991556700.

[160] Mathias Slawik et al. “The Open Service Compendium. Business-Pertinent Cloud Service Discovery, Assessment, and Selection”. In: *Economics of Grids, Clouds, Systems, and Services: 12th International Conference, GECON 2015, Cluj-Napoca, Romania, September 15-17, 2015, Revised Selected Papers*. Ed. by Jörn Altmann, Gheorghe Cosmin Silaghi, and Omer F. Rana. Cham: Springer International Publishing, 2016, pp. 115–129. isbn: 978-3-319-43177-2. doi: 10.1007/978-3-319-43177-2_8.

[161] Wendell R. Smith. “Product differentiation and market segmentation as alternative marketing strategies”. In: *The Journal of Marketing* (1956), pp. 3–8. issn: 0022-2429.

[162] SOA4ALL Consortium. *SOA4ALL: Service Oriented Architectures for All*. 2011. url: http://cordis.europa.eu/project/rcn/85536_en.html.

[163] Steve Souders. *The HTTP archive*. 2013. url: http://httparchive.org/.

[164] C. M. Sperberg-McQueen and Henry Thompson. *XML Schema*. 2014. URL: http://www.w3.org/XML/Schema.

[165] Josef Spillner and Alexander Schill. “A Versatile and Scalable Everything-as-a-Service Registry and Discovery”. In: *CLOSER 2013 Proceedings*. Ed. by Frédéric Desprez et al. SciTePress, 2013, pp. 175–183. ISBN: 978-989-8565-52-5.

[166] Squid. *Squid: Optimising Web Delivery*. 2016. URL: http://www.squid-cache.org/ (visited on 11/11/2016).

[167] William Stallings. *Cryptography and network security: principles and practices*. Pearson Education India, 2006.

[168] Rudi Studer, Stephan Grimm, and Andreas Abecker. *Semantic web services: Concepts, technologies, and applications*. Berlin and New York: Springer, 2007. ISBN: 9783540708933.

[169] G. Succi and M. Marchesi. *Extreme Programming Examined (XP)*. Addison-Wesley Longman, Amsterdam, 2001.

[170] Willy Tarreau. *HAProxy - The Reliable, High Performance TCP/HTTP Load Balancer*. 2016. URL: http://www.haproxy.org.

[171] Dirk Thatmann et al. “Deriving a Distributed Cloud Proxy Architecture for Managed Cloud Service Consumption”. In: *CLOUD 2013 Proceedings*. IEEE, 2013, pp. 614–620. ISBN: 978-0-7695-5028-2.

[172] Dirk Thatmann et al. “Towards a Federated Cloud Ecosystem: Enabling Managed Cloud Service Consumption”. In: *Economics of Grids, Clouds, Systems, and Services*. Berlin, Germany: Springer-Verlag, 2012, pp. 223–233.

[173] The CYCLONE project. *CYCLONE - Home*. 2017. URL: http://www.cyclone-project.eu/ (visited on 01/17/2017).

[174] The DIP consortium. *Data, Information, and Process Integration with Semantic Web Services - Project DIP -*. 2008. URL: http://dip.semanticweb.org/.

[175] The Eclipse Foundation. *Eclipse Modeling Framework Project (EMF)*. 2014. URL: http://www.eclipse.org/modeling/emf.

[176] The Eclipse Foundation. *Virgo - Home*. 2012. URL: http://www.eclipse.org/virgo/ (visited on 11/09/2016).

[177] The European Commission. *A Recovery On The Horizon: Annual Report on European SMEs 2012/2013*. Ed. by Deborah Cox et al. 2013.

[178] The Open Web Application Security Project. *Application Threat Modeling*. 2015. URL: https://www.owasp.org/index.php/Application_Threat_Modeling.

[179] The Open Web Application Security Project. *Threat Risk Modeling*. 2015. URL: https://www.owasp.org/index.php/Threat_Risk_Modeling.

[180] The OpenSSL project. *OpenSSL: Cryptography and SSL/TLS Toolkit*. 2016. URL: https://www.openssl.org/ (visited on 11/25/2016).

[181] TRESOR Consortium. *About TRESOR*. 2012. URL: http://www.cloud-tresor.com (visited on 11/08/2016).

[182] Abdulbaki Uzun, Eric Neidhardt, and Axel Küpper. "OpenMobileNetwork: A Platform for Providing Estimated Semantic Network Topology Data". In: *International Journal of Business Data Communications and Networking (IJBDCN)* 9.4 (2013), pp. 46–64. issn: 1548-0631. doi: 10.4018/ijbdcn.2013100103.

[183] Bill Venners. *The Simplest Thing that Could Possibly Work: A Conversation with Ward Cunningham, Part V*. 2004. url: http://www.artima.com/intv/simplest.html.

[184] W3C OWL Working Group. *OWL 2 Web Ontology Language Document Overview: W3C Recommendation 11/12/2012*. 2012. url: http://www.w3.org/TR/owl2-overview.

[185] Peter Wayner. "13 ways the cloud has changed (since last you looked)". In: *InfoWorld* 2016 (2016). url: http://www.infoworld.com/article/3030138/cloud-computing/13-ways-the-cloud-has-changed-since-last-you-looked.html.

[186] Jon Weissman and Siddharth Ramakrishnan. "Using Proxies to Accelerate Cloud Applications". In: *Proceedings of HotCloud 09 - Workshop on Hot Topics in Cloud Computing*. 2009. url: https://www.usenix.org/legacy/event/hotcloud09/tech/full_papers/weissman.pdf.

[187] Xiaoxin Wu, Lei Xu, and Xinwen Zhang. "Poster: a certificateless proxy re-encryption scheme for cloud-based data sharing". In: *Proceedings of the 18th ACM conference on Computer and communications security*. CCS '11. New York, NY, USA: ACM, 2011, pp. 869–872. isbn: 978-1-4503-0948-6. doi: 10.1145/2093476.2093514.

[188] Xpra. *xpra home page*. 2014. url: http://xpra.org/ (visited on 01/17/2017).

[189] Miranda Zhang et al. "A Declarative Recommender System for Cloud Infrastructure Services Selection". In: *Economics of grids, clouds, systems, and services*. Ed. by Kurt Vanmechelen, Jörn Altmann, and Omer F. Rana. Vol. 7714. Lecture Notes in Computer Science. Berlin: Springer, 2012, pp. 102–113. isbn: 978-3-642-35194-5. doi: 10.1007/978-3-642-35194-5_8.

[190] Miranda Zhang et al. "An Infrastructure Service Recommendation System for Cloud Applications with Real-time QoS Requirement Constraints". In: *IEEE Systems Journal* 11.4 (2017), pp. 2960–2970. issn: 1932-8184. doi: 10.1109/JSYST.2015.2427338.

[191] Qi Zhang, Lu Cheng, and Raouf Boutaba. "Cloud computing: state-of-the-art and research challenges". In: *Journal of Internet Services and Applications* 1 (2010), pp. 7–18.

[192] Wenwu Zhu et al. "Multimedia Cloud Computing". In: *Signal Processing Magazine, IEEE* 28.3 (2011), pp. 59–69. issn: 1053-5888. doi: 10.1109/MSP.2011.940269.

[193] S. Zickau and A. Küpper. "Towards Location-based Services in a Cloud Computing Ecosystem". In: *Ortsbezogene Anwendungen und Dienste - 9. Fachgespräch der GI/ITG-Fachgruppe Kommunikation und Verteilte Systeme*. Vol. 9. Chemnitz, Germany: Universitätsverlag Chemnitz, Sept. 2012, pp. 187–190. isbn: 978-3-941003-77-4. url: http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-104609.

[194] S. Zickau et al. “Enabling Location-based Policies in a Healthcare Cloud Computing Environment”. In: *Proceedings of the 3rd IEEE International Conference on Cloud Networking (IEEE CloudNet)*. IEEE, 2014, pp. 333–338. DOI: 10.1109/CloudNet.2014.6969017.

[195] Begüm İlke Zilci, Mathias Slawik, and Axel Kupper. “Cloud Service Matchmaking using Constraint Programming”. In: *2015 IEEE 24th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. IEEE. 2015, pp. 63–68.

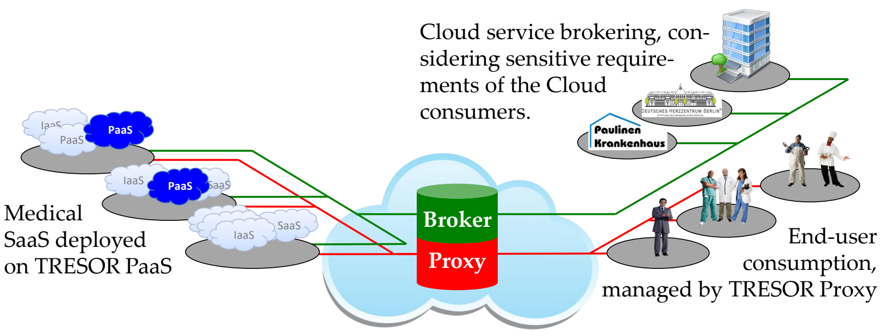
[196] Begüm İlke Zilci, Mathias Slawik, and Axel Küpper. “Cloud Service Matchmaking Approaches: A Systematic Literature Survey”. In: *2015 26th International Workshop on Database and Expert Systems Applications (DEXA)*. IEEE, 2015, pp. 181–185. DOI: 10.1109/DEXA.2015.50.

Appendix A

Further Information About TRESOR and CYCLONE

This appendix provides further information on TRESOR and CYCLONE, especially the respective key concepts and use cases.

A.1 TRESOR Key Components



Cloud service brokering, considering sensitive requirements of the Cloud consumers.
Paulinen Krankenhaus
IaaS
PaaS
PaaS
IaaS
PaaS
SaaS
IaaS
SaaS
Broker
Proxy
Medical SaaS deployed on TRESOR PaaS
End-user consumption, managed by TRESOR Proxy

Figure A.1: TRESOR overview, adapted from [181]

The three key TRESOR components Broker, Proxy, and PaaS are depicted in Figure A.1 and explained in the following paragraphs. At the end of this section, the TRESOR Federation Provider as well as the distributed authorization components are also presented.

TRESOR Broker

The *TRESOR Broker* serves as a mediator of medical services between health centers and cloud providers, especially considering sensitive requirements of the users. The brokering functionality is based on an extensive cloud service repository as well as a comprehensive cloud service description language. It covers diverse description areas, such as technical interfaces, legal constraints, business models, service-level agreements (SLAs), compliance, and price models. All in all, it eases the cloud service discovery, assessment, and selection of the health centers considerably.

TRESOR Proxy

The *TRESOR Proxy* completely manages end user's cloud consumption - all service requests are made to the proxy (instead of the cloud service). The proxy provides a comprehensive set of features, for example, guaranteeing regulatory compliance through applying XACML-based access policies [122], extensive monitoring, as well as integration with enterprise SSO and the other TRESOR components. Common lock-in effects, such as vendor tool dependencies and proprietary protocols are avoided, as the proxy provides open APIs and is based on regular HTTP(S) communication, fitting for the REST-based architectures found in Cloud Computing.

TRESOR PaaS

The *TRESOR PaaS* is a secure platform used to deliver medical SaaS in a trustworthy manner. It relies on OSGi [124] to provide modularity and offers a number of supporting services to cloud applications.

TRESOR Federation Provider

When multiple institutions, such as the two hospitals involved in TRESOR, share data through a common service, it is beneficial to integrate their respective identity providers so that end-users can reuse their existing identities. This includes other potential personal identification mechanisms as well, for examples, personalized smart cards. In order to prevent each service to separately integrate each accessing organization, TRESOR relies on the Federation Provider which provides a uniform user identity representation to all consuming services. As most health centers use Microsoft ActiveDirectory, the Federation Provider is based on the "ActiveDirectory Federation Services", following the "Claims-based Authentication"[1] paradigm proposed by Microsoft.

Distributed Authorization Components

Accessing medical cloud services is often governed by access policies that, for example, restrict access to patient data to a set of users on certain times from certain locations. Furthermore, compliance requirements require that each access and modification of personal data should be auditable. TRESOR includes distributed authorization components that enable the separation of the policy definition from its decision and enforcement. Policies can be shared with multiple services and the service providers are exempt from laboriously implementing access control mechanisms that are flexible enough for each cloud consumer. TRESOR follows the Extensible Access Control Markup Language (XACML) which is explained in Section 2.4.2.

[1]http://claimsid.codeplex.com/

A.2 TRESOR Use Cases

Two Berlin hospitals ensured relevance and applicability of the proposed Cloud ecosystem and its components: the "Deutsches Herzzentrum Berlin", a "specialized hospital for the diagnosis and treatment of cardiovascular disease"[2] and the "Paulinenkrankenhaus", focusing on "post-operative treatment of patients of the German Heart Institute Berlin and of the Charité, on prevention, and on the comprehensive treatment of cardiac conditions"[3]. Based on expert interviews, TRESOR identified two fields of application for potentially helpful Cloud Computing solutions [31]: medical history documentation without any media breaks as well as checking drug interactions.

Use Case 1: Medical History Documentation Avoiding Media Breaks
One of the greatest challenges of cross-hospital medical treatment is the design of a documentation process that is free of media breaks, continuous, self-contained, and compliant to data protection directives. This process would directly benefit patients and their attending physicians. Even after inpatient stay, this process could be used to diagnose mandatory changes in therapy at an early stage and allow immediate intervention. Related approaches do not scale well and are associated with interoperability and data protection challenges.

The TRESOR standards-compliant PaaS platform as well as a privacy requirements-aware cloud broker address those issues. Additional value could be created by adopting, selecting, and presenting additional patient data, for example, blood pressure, weight, self-, and third-party-observations, and providing a combined view to healthcare staff independent of date and means of access via a secure channel. This channel can also be used by patients to voluntarily share their position and to allow target-oriented treatment.

Use Case 2: Optimizing the Treatment Chain Using the Example of Checking Drug Interactions
Undesirable drug effects are often attributed to non-coordinated drug combinations and dosages. They play a key role in patient care complication rates and mortality. The demographic and clinical patient data, such as age, sex, weight, body surface, allergies, as well as kidney and liver functions, are also decisive for the occurrence of such problems. In addition, complex questions often arise along with the treatment process, which are dependent on attributes to be specified, such as care diagnoses, procedures, and other medical data. They can be answered by means of nursing guidelines, specialist information, and scientific publications. To display unwanted drug interactions or other information offers, today, locally installed information systems are used, whose data is updated with a time lag in a laborious process. These systems also have the disadvantage that media breaks occur during the input of the mentioned patient data.

A cloud ecosystem that gives access to drug discovery databases and other personalized information to different actors, such as hospitals, physicians' practices, and pharmacies, can solve these problems. TRESOR established a cloud-based information service for drug interaction testing, which can be invoked

[2]https://www.dhzb.de/en/
[3]https://www.paulinenkrankenhaus.de/home.html

by locally installed medical systems and which can be orchestrated with other application services of the cloud ecosystem, for example for anonymous access to patient data. Access to this information service by the different actors is managed and controlled by the cloud proxy.

A.3 CYCLONE Overview: General Approach and Flagship Use Case

For the work that TU Berlin was responsible which I was coordinating, we applied a bottom-up approach, consisting of a thorough analysis of the CYCLONE use cases and deriving requirements together with the involved stakeholders. This includes the functional and non-functional requirements that provide the constraints for our work. Borrowing from product development, we started with the "Minimum Feature Set" as summarized well by Blank in [27], that is, with the smallest multi-cloud problem that the use case users would use the CYCLONE stack for. Then, we continually built upon these requirements and the implementation to further extend the software stack. The limited project resources caused us to focus the implementation on the main requirements in the following areas:

1. Deployment of multi-cloud applications, especially handling the deployment of bioinformatics clusters
2. Using federated identities for authentication and authorization in web applications as well as for SSH login
3. Securely interconnecting VMs of cloud application deployments, providing multi-cloud VPN, firewalling, and load balancing services

The strategies from Extreme Programming (XP) guide our work, for example, the DTSTTCPW and YAGNI strategies, as summarized by Succi et al. in [169, p. 208]: "… we must learn to let designs emerge and not anticipate what will be. XP says 'do the simplest thing that could possibly work' (DTSTTCPW) because 'you aren't gonna need it.' (YAGNI).". The effect of following these strategies is the leaning of CYCLONE towards applying simple solutions fitting well to the use cases instead of trying to use highly sophisticated generic solutions that would first have to be streamlined for their application.

CYCLONE Flagship Use Case: Bioinformatics

Using current technology, sequencing bacterial genomes is very cheap, costing only a few hundred Euros. Therefore, many end users from the bioinformatics domain are no longer satisfied with analyzing just single genomes: they additionally require comparing collections of related genomes, so called "strains". Faced with an ever-increasing number of sequenced genomes, biologists need efficient and user-friendly tools to assist them in their analyses. In this context, tools that facilitate comparative genomics analyses of large amounts of data are needed. This includes the conservation of gene neighborhood, presence/absence of orthologous genes, phylogenetic profiling, and other specialized functions.

One of the CYCLONE use case partners, the French Institute of Bioinformatics (L'Institut Français de Bioinformatique), consists of 36 bioinformatics platforms (PF) spanning the entire French territory as well as a national hub, the "UMS 3601–IFB-core", which is the representative in the CYCLONE project. The IFB has deployed a cloud infrastructure on its own premises at IFB-core and aims to deploy a federated cloud infrastructure over the regional PFs. This cloud infrastructure is devoted to the French life science community, research, and industry, with services for the management and analysis of life science data. More concretely, bioinformaticians can use this "IFB Bioinformatics Cloud" to deploy VMs containing useful bioinformatics research tools.

The CYCLONE middleware stack is fully utilized in the use case, managing the deployment of the applications, using the federated identity for authenticating and authorizing application users, as well as interconnecting the components of clustered tools securely. Therefore, the Bioinformatics use case is an excellent fit to the challenge areas addressed by CYCLONE.[4]

A.4 CYCLONE Middleware Components

There is a lot of information already available about the components of the CYCLONE cloud middleware, for example, on the website [173] and within other publications of the project. The following content concentrates on the most important components for the integration into the bioinformatics use case and explain them briefly. This will guide the explanation of the subsequent "CYCLONE in Action" section.

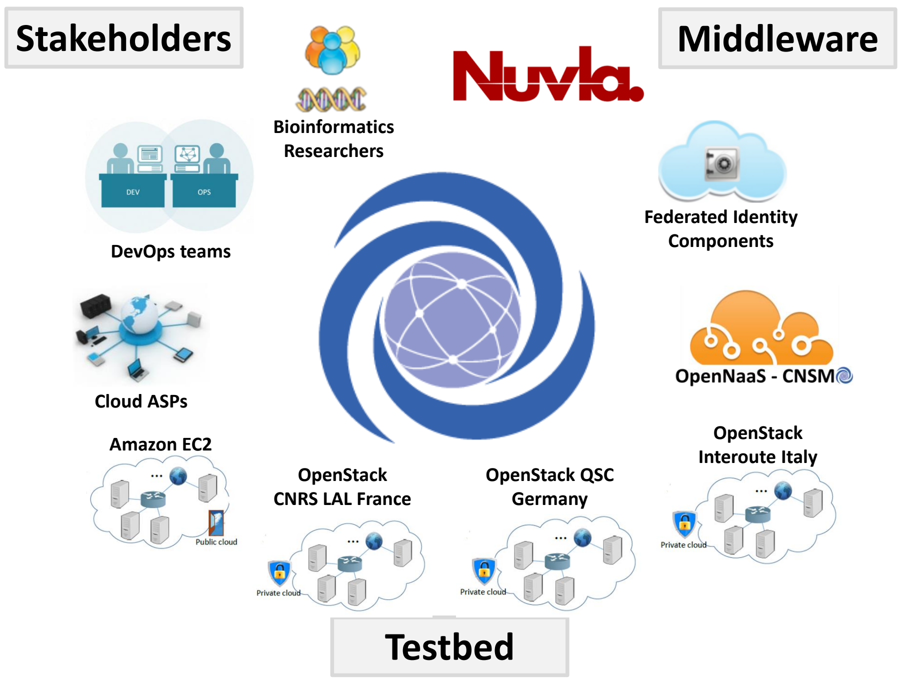


Figure A.2: CYCLONE constituents, adapted from own image in [151]

[4]The content for this section was mainly provided by Christophe Blanchet (IFB) for [151] and edited by me to fit to the narrative of this thesis.

Figure A.2 provides an overall view of the main CYCLONE constituents: **use case stakeholders** (e.g., bioinformaticians) which use the **CYCLONE middleware** stack and validate it within the **testbed infrastructure**. The testbed infrastructure comprises both private and public clouds to demonstrate the deployment of hybrid multi-cloud applications.

Nvla: Deploy Applications on any Cloud

SlipStream, a cloud application management platform, allows developers to define portable cloud applications and operators to deploy automatically those applications on multiple cloud infrastructures. With SlipStream, the operators can manage the full lifecycle of cloud applications, including provisioning, scaling, migration, and clean up. SixSq releases the SlipStream Community Edition under the Apache 2 license and the source code can be found in the SlipStream organization in GitHub. In addition, SixSq operates a free SlipStream SaaS called Nuvla [150] which can be used to access a number of public clouds.

Managing virtual machines as large binary files complicates cloud application management and portability. The large files, typically tens of gigabytes, are costly to transport. The binary files hide the knowledge about the virtual machine service configuration. Use on multiple platforms requires conversion that is both time-consuming and error-prone. To avoid these problems, SlipStream uses a "recipe"-based model for defining and managing applications.

Developers define an application in SlipStream by referencing a base image (typically a minimal distribution of a common operating system) and providing a set of "recipes" that are executed at various points in a VM's lifecycle. These recipes describe the software installation and configuration process to transform a minimal machine into the desired application component. This process captures the deployment knowledge for everyone on a DevOps team and ensures that the application can be deployed on any cloud. Recipes can also be defined for scaling actions, allowing an application to adapt to changes in load or performance.

Most production services contain many components; for example, typical web applications have stateless frontends, load balancers, and databases for persistent storage. SlipStream provides a "parameter database" for each deployment which allows an application's components to share information (e.g., the database address) and to coordinate themselves (e.g., ensure the database is available before starting a client). This provides a simple but effective mechanism for reliable and automated application deployment. Developers can even provide tests to their recipes to validate the service before opening it to users.

The Service Catalog, a core feature of SlipStream, contains "offers" from cloud service providers, detailing VM resource configurations, locations, prices, and other information. Developers and operators can attach "policies" that describe constraints to an application. SlipStream then uses those policies to filter the available offers to eliminate those that do not meet the application requirements. The operator can then select any acceptable offer manually or allow SlipStream to choose the least costly offer automatically. These policies are completely general and can be used, for example, to deploy an application to a particular country for legal reasons or to choose a particular combination of CPU cores, RAM, and disk space.

For DevOps teams, SlipStream provides a convenient, collaborative platform that allows both developers and operators to take advantage of multiple cloud services while simplifying and automating their workflows.[5]

CYCLONE Federation Provider: Use Federated Identities with Ease

Identity Management is a challenge area often faced by application developers and operators. When applications using federated identities are deployed on multiple clouds, the secure design and rapid implementation becomes a complex endeavour. CYCLONE provides the *CYCLONE Federation Provider* to ease the hardships of federated multi-cloud identity management. The project also makes special arrangements to ease the integration of preexisting academic identities that are federated through eduGAIN [62], as the end-users in many implemented CYCLONE use cases are academic researchers.

From a conceptual perspective, using a centralized authentication server decouples the application authentication and reduces the functional footprint of application nodes. As the Federation Provider relies on widely used standards, the integration of Web-based SSO is easier as supporting libraries are widely available. Furthermore, the Federation Provider transforms different user identities into a consistent attribute format (JSON Web Token), decoupling the application node authentication, OpenID Connect, from the different authentication methods used at the Federation Provider.

From an implementation perspective, the CYCLONE Federation Provider extends and enriches the Keycloak identity and access management solution [133] which is sponsored by RedHat. Keycloak has a rich feature set, mainly *single sign-on*, supporting both SAML2 (as used by eduGAIN) as well as OpenID Connect (as used by many cloud applications). Keycloak can also *broker identities*, allowing end users to select which credentials they want to use for authentication, even supporting social network logins such as Facebook. The CYCLONE extensions to Keycloak comprise a data privacy aware session removal, an interface for self-service registration, as well as templates that include terms of conditions and data privacy statements for each OpenID connect tenant. They were explained in detail in Section 3.4.2.

There is a shared Federation Provider Instance in the CYCLONE testbed which is integrated with eduGAIN. Using such a shared instance is beneficial in two ways: first of all, integrating an application with eduGAIN is a manual process that, from experience, can take weeks and differs for each university. Second, CYCLONE offers the eduGAIN identities within an OpenID Connect flow, thus easing the implementation of relying cloud applications. In addition, CYCLONE also provides software sources so that, for example, other ASPs can implement their own Federation Providers with less efforts.

Besides the similar name, the CYCLONE Federation Provider does not share much similarities with the TRESOR Federation Provider which was introduced in Section A.1. Instead of proprietary Microsoft technology, CYCLONE relies on open source software. Furthermore, CYCLONE relies on Open ID Connect instead of SAML for its authentication API.

PAM Module and Xpra Wrapper: WebSSO for Fun and Profit

[5]The content for this section was mainly provided by Charles Loomis (SixSq) for [151] and edited by me to fit to the narrative of this thesis.

Many users that leverage multiple clouds face the problem of having a large number of user accounts with different services. As detailed in the last chapter, this problem can be reduced using web-based single sign-on. However, no SSO implementation can be used satisfyingly for Secure Shell Login. This problem is amplified in the Bioinformatics use case as researchers share datasets and results by letting other people log into the VMs. Currently, involved researchers need to create a new user account for every person whom they share data with.

The CYCLONE PAM module `pam_openid_connect` [20] allows SSH logins using the federated identities of the end users, for example, the Bioinformatics researchers. It is integrated with the SSH server through the PAM subsystem. The keyboard-interactive mode of SSH allows the PAM module to display a URL of an ephemeral web server started for this particular login session. When users follow this link, it initiates a regular Open ID Connect Authentication Code Flow with the CYCLONE Federation Provider that returns user attributes as a JSON Web Token to this ephemeral web server. The current implementation compares the user's email with a list of allowed emails in a file. This file can be edited by the bioinformaticians as well as created at deployment time by SlipStream.

Other bioinformatics software is provided as regular desktop applications. There are some preexisting tools that allow remote access to desktop applications, for example, Xpra [188]. The Xpra client and server can communicate via an SSH connection, potentially established using the CYCLONE PAM module. However, the manual setup and coordination of Xpra and SSH is not always easy for the end users. To provide such an easy way for the bioinformaticians to use remote desktop applications authenticated with their federated identity, CYCLONE provides a desktop "wrapper" around both tools, available at [19]. It uses Electron [65], a tool provided by GitHub to "build cross platform desktop apps with JavaScript, HTML, and CSS" in order to lower the implementation effort and to provide the wrapper to a large range of different users and devices.

OpenNaaS CNSMO: Connect all the Clouds

Modern computing platforms span multiple cloud infrastructures in order to achieve resilience, responsiveness, and elasticity. Most often, they require secure network connectivity, at best automatically managed and available on-demand. However, unless companies pay a significant amount of money for customized cloud infrastructure, many limitations persist in the network services offered by common public cloud vendors: first of all, the networking APIs and procedures differ widely between cloud providers, oftentimes to an incompatible degree. Secondly, tenants have little control over network services and limited visibility over networking resources that were made available to them. This severely limits tenants' flexibility and prevents them from implementing application logic in the network.

CYCLONE provides network services to cloud-based applications using OpenNaaS CNSMO (**C**YCLONE **N**etwork **S**ervices **M**anager and **O**rchestrator) which was presented in [10], available online at [119]. It is far more lightweight than comparable solutions such as Apache Mesos while still providing the essential network management APIs. The system is capable of deploying, configuring, and running multiple network services in both private and public environments. The most significant CNSMO feature is that it is **agnostic to the**

underlying IaaS provider, running on top of any cloud service and being OS independent. It avoids the need to gain access and control of IaaS infrastructures since it works on top of network overlays. Thus, CNSMO integrates networking aspects over federated clouds and allows tenants to request network services and manage them. CNSMO leverages Docker containers for easy deployment and management.

The CNSMO services are stateless, independent, and can invoke themselves. Any CNSMO service can potentially launch any other service. In effect, CNSMO is lightweight, distributed, and modular: **Lightweight** service agents coordinate themselves by communicating through a **distributed** system state. CNSMO features a **modular** micro-service architecture which is scalable and extendable: agents are atomic, single-purpose units. The CNSMO architecture is detailed in [11].[6]

Additional Components

There are additional constituents of CYCLONE that are not presented here, mainly the extension of the existing middleware with an *Intercloud Access Control Infrastructure* as well as *Trust Bootstrapping* components. The general idea is to offer the Use Case owners both a far more sophisticated, XACML-based authorization infrastructure, as well as components to bootstrap trust in multi-cloud deployments. As I have not been involved in their development and deployment and they are not ready by the time this thesis was written, they are not explained in detail at this point. The project deliverables [154, 157] explain the motivation as well as the planned integration of these components within CYCLONE.

A.5 CYCLONE in Action

One of the cornerstones of CYCLONE is the application of the CYCLONE middleware stack within a broad range of use cases. The following paragraphs highlight some use cases and show how well the stack fits to the requirements of the use case stakeholders.

Deploying Bioinformatics Software

In [93] Lacroix, et al. present Insyght, which is a comparative genomic visualization tool consisting of 3 components: a pipeline of Perl scripts to compute required data, a relational database for persisting data, and the visualization tool itself which queries the relational databases and presents the data in a user-friendly way. The platform automatically launches a set of bioinformatics tools (BLAST, PSI-BLAST, INTERPROScan, etc.) to analyse the data and stores the results of the tools in the relational PostgreSQL database. These tools use several public reference data collections. A web interface allows the end-users to consult the results and perform the manual annotation, which means manually adding metadata and biological knowledge to the genome sequence. The popularity and vast functionality make Insyght a prime candidate for offering it on the IFB Bioinformatics Cloud.

[6]The content for this section was mainly provided by José Aznar (i2Cat) for [151] and significantly edited by me to fit to the narrative of this thesis.

The Insyght deployment comprises two components: a master running the workflow, scheduling the genomes comparisons and storing the result, and several nodes to perform the genomes comparisons. Previously, they were both deployed within a single image that needs to be imported to the target cloud. However, many clouds either do not allow importing custom VM images or do not support images built for other clouds. This challenge can be easily solved using the CYCLONE middleware: using SlipStream, IFB developers can create generic deployment recipes that can be deployed to all major cloud platforms, such as the OpenStack cloud used in the CYCLONE testbed.

Deploying each component to different nodes requires one master and several worker nodes according to the size of the genomes dataset to analyze. The CYCLONE middleware stack also helps making this task easy, as Slipstream provides the facilities to deploy and scale heterogeneous applications consisting of different types of VMs. This set of VMs and their data exchange needs to be isolated from other cloud users and VMs for security and operating purposes, for example, to ease the management of the data exchange between the nodes or the NFS exports and mounts. For this purpose, CYCLONE leverages the VPN service offered by CNSMO.

Within the Bioinformatics use case, the CYCLONE Federation Provider and the PAM module provide easy and secure access management for the deployed VMs. They also provide reliable and ubiquitous identity management using user identities from eduGAIN federated identity providers. The PAM module has simplified the access of the Bioinformaticians to their VMs by liberating them of managing the SSH keys, which can be problematic according to the computing skill of the user and the operating system that is used by them. At last, the PAM module simplified the security of the Bioinformatics cloud infrastructure from both the end-user and the cloud provider point of view.[7]

Creating VPNs Over any Cloud

CNSMO is integrated with Slipstream and relies on its facilities to deploy the underlying VM image as well as execute scripts to set it up. A single SlipStream application component can run any number of CNSMO agents (one agent in each of the application VMs), depending on the network services that have been selected to be deployed together with the application. Figure A.3 shows the integration of CNSMO in the SlipStream market place.

There are three network services in CYCLONE: a multi-cloud VPN, a firewall, and a load balancer. They were the first to be requested within the flagship use cases and have been fully implemented, tested, integrated, and validated over several cloud infrastructures. The following items explain the use of CNSMO on the example of the VPN service by iterating the general bootstrapping process (see Figure A.3):

1. Before deploying an application, its SlipStream recipe is created which includes the network services that should be deployed.
2. When the application deployment is initiated through SlipStream, the SlipStream Orchestrator VM instantiates the application VM images as new VMs on the target cloud and executes the respective deployment

[7]The content for this section was mainly provided by Christophe Blanchet (IFB) for [151] and edited by me to fit to the narrative of this thesis.

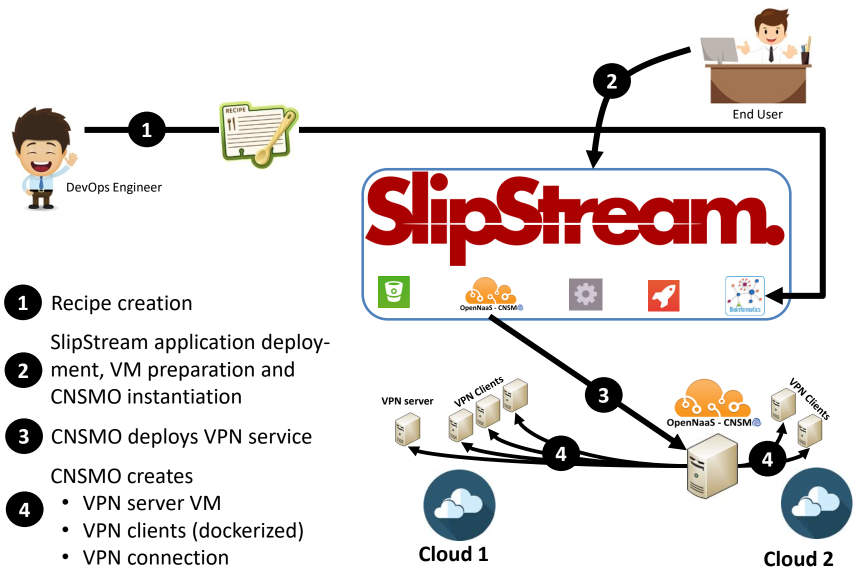


Figure A.3: Integration of CNSMO with SlipStream

recipes. Part of this deployment is the CNSMO image which contains a `systemd` unit that executes CNSMO as a system service.

3. Once CNSMO is launched, it provides an API so that SlipStream can control the deployment of the network services. To this end, the CNSMO SlipStream application uses a deployment recipe that includes the appropriate instructions for CNSMO to deploy the chosen networking services, in this case a VPN. It is important to clarify that the recipe is run by SlipStream inside the CNSMO VM. For instance, for the concrete case of a VPN network service, the SlipStream command line client calls the CNSMO API to deploy the VPN service, consisting of a VPN server as well as VPN clients.

4. The SlipStream deployment parameters are used by CNSMO to determine which services to deploy. For the exemplary distributed VPN service, CNSMO carries out the following steps:

   - It creates the VPN server VM.
   - It creates the VPN clients inside the VMs of the application that has been deployed by SlipStream.
   - It launches the VPN server to configure the VPN clients and establish the VPN service.
   - Finally, it uses the SlipStream command line client to announce to the rest of the components that the networking service has been set up,

so that SlipStream can resume the deployment of other application constituents.[8]

Using Academic Identities in Research Prototypes

The "Internet of Services Lab" (IoSL) is a teaching project of TU Berlin where students work in groups of three to six, implementing software related to numerous research projects and other topics. This type of teaching project is quite popular in other study paths related to computer science as well as at other universities. Within the IoSL, there are different areas where the application of CYCLONE provides numerous benefits:

- Rapid Provisioning of Resources for Student Projects

  Students require resources for conducting their projects, mostly virtual machines. In the current set-up it is a manual procedure to provision those resources. By leveraging the testbed as well as the deployment tools, CYCLONE minimizes the required effort of the student supervisors to create new student VMs considerably.

- Utilization of SlipStream Modules for Reproducible Application Deployments

  After students finish their course it is often problematic for other students and their supervisors to pick up their work. Most often, documentation is lacking and software versioning is not reliable, if it is available at all. By building upon SlipStream modules, students can create application deployments that can be easily reproduced, extended, and scaled. Also, students will learn how to structure their applications to leverage cloud characteristics, for example, how to create immutable application deployments.

- Integration of the CYCLONE Federation Provider for Simplified Account Management

  Teaching experience shows that every built demonstrator has its own user management, oftentimes not following security best practices. By integrating the Federation Provider into each demonstrator, students learn about federated identity and are also liberated from implementing their own user management, as all students and supervisors will be able to login to the demonstrators via their eduGAIN identities.

[8]The content for this section was mainly provided by José Aznar (i2Cat) for [151] and significantly edited by me to fit to the narrative of this thesis.

Danksagung

Obwohl ich das vorliegende Werk als rein wissenschaftliche Arbeit betrachte, so möchte ich der Gepflogenheit einer Danksagung an dieser Stelle nachkommen.

Auf privater Seite möchte ich mich zuerst bei meinen Eltern bedanken, dass sie stets mit großem Nachdruck an meiner akademischen Laufbahn interessiert waren und erhebliche Mühen für meine Unterstützung aufgebracht haben. Gleich danach danke ich meiner Frau, dass sie mir trotz zwei gemeinsamer Kinder oft den Rücken freigehalten und mir unerlässlich moralische Unterstützung zukommen lassen hat.

Auf akademischer Seite bedanke ich mich bei meinem Betreuer, Prof. Dr. Axel Küpper, für seine Unterstützung und die Einrichtung von Freiräumen für die Arbeit an der Dissertation. Weiterhin danke ich allen meinen ehemaligen Kollegen am Lehrstuhl für den intensiven und fruchtbaren Austausch in den letzten Jahren. Außerdem geht mein Dank an die weiteren Gutachter, dass sie sich die Zeit genommen haben, um mir hilfreiches Feedback zum Entwurf zu geben.

Zuletzt bedanke ich mich bei Torsten Frank und der medisite GmbH, dass sie mir durch einen flexiblen Arbeitsvertrag den zügigen Abschluss dieser Arbeit ermöglicht haben.