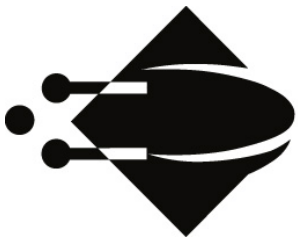


IT University of Copenhagen

PhD Dissertation

Foundations for Tools as a Service Workspace: A Reference Architecture

---

Design Space, Architecture Models, Design Tactics, Prototype and Experiences

Muhammad Aufeef Chauhan
December 2015

IT University of Copenhagen
Software and Systems Section
Rued Langgaards Vej 7
2300 Copenhagen S, Denmark

Supervisor:
M. Ali Babar, IT University of Copenhagen

Evaluation Committee:
Andrzej Wasowski, IT University of Copenhagen
Christian W. Probst, Technical University of Denmark
Rami Bahsoon, The University of Birmingham

Abstract

Nowadays, on-demand provisioning of computing resources following a pay-per-use service model have enabled client organizations to have easy and on the fly access to the resources. The resources that are provisioned using the service model are generally characterized as *resource as a Service* (*aaS). However, providing the resources corresponding to a domain following *aaS requires specific challenges associated with the domain to be addressed. Software as a Service (SaaS) model enables software vendors to offer their software solutions to end users following pay-per-use model. SaaS also enables end users to have access to the software system without being bound into long-term license commitments and without incurring additional infrastructure and maintenance overheads. Though SaaS has been successful in providing stand-alone applications where users can perform a specific set of activities using an application, applicability of SaaS for scenarios where users need to use a number of software systems to perform activities and associated tasks is limited. A typical such use case is the activities associated with software engineering domains in general and software architecting domain in particular.

This dissertation presents an approach to provide Tools as a service workSPACE (TSPACE), which is characterized as provisioning of a bundled suite of Tools as a Service (TaaS) required to perform activities associated with a specific domain as part of a cloud-enabled workspace. The presented approach is focused on addressing the challenges using an architecture centric solution by providing a *Software Reference Architecture* for TSPACE. As a specific case, software architecting domain and the tools used for software architecting have been focused in this dissertation.

This dissertation explores (a) the challenges associated with software architecting of cloud-enabled systems, (b) TSPACE reference architecture design guidelines, (c) TSPACE requirements, (d) information structuring needs and methods of TSPACE reference architecture, (e) a detailed description of TSPACE reference architecture sub-systems and components, and (f) TSPACE reference architecture evaluation.

The research goals are achieved by applying systematic literature review method, general literature surveys, reference architecture design methodologies, cloud and general architecture styles and patterns, architecture prototyping and architecture evaluation methods as tools.

The main results of this dissertation are (a) a systematic review of the literature that identifies the challenges associated with software architecting of cloud-enabled systems, (b) a framework that guides the development of

TSPACE reference architecture and its evaluation, (c) the business, functional, quality and software architecture significant requirements of TSPACE, (d) TSPACE reference architecture meta-models and a set of ontologies to structure concepts and elements of TSPACE (e) a detailed description of the reference architecture based on established design practices and architecture patterns and (f) the detailed process of TSPACE reference architecture evaluation based on using architecture evaluation methods and a prototype of TSPACE reference architecture along with integrated tools to analyze applicability, effectiveness and usefulness of TSPACE reference architecture.

The main conclusions of this dissertation are (a) as the nature of the tools that are used in a specific domain vary, the domain models play a vital role in design of the TSPACE reference architecture for the respective domain (b) TSPACE reference architecture should have the capability to identify tools needed for the specific tasks (e.g. software architecting tasks and activities) and be able to bundle the needed tools in a suite of tools to provision TaaS, (c) as the nature of the tools vary in terms of the activities that are supported (e.g. specifying architecture significant requirements, capturing architecture knowledge and modeling architecture components), abstraction level of the artifacts that are produced and the way artifacts are exchanged among the tools and support for semantic as well as process-centric integration among the tools play a vital role in the adaptation of the reference architecture and (d) the reference architecture should be flexible enough to accommodate a wide variety of the tools used in a domain and the operations that are performed on the artifacts using the tools.

Acknowledgements

I would like to express my deepest gratitude to Professor Ali Babar for giving me an opportunity to carry out PhD research under his supervision and for mentoring me throughout this exciting journey. His contributions on publications, and his comments and suggestions on every stage of my research work have always been valuable. I also thank him for making me familiar with Danish education system.

I acknowledge Professor Sheng for providing an opportunity to visit The University of Adelaide for my stay abroad period and for hosting me at the university. I acknowledge his guidance and feedback on my PhD work. I would also like to thank members of Centre for Research on Engineering Technologies (CREST) and all my colleagues in Software and Systems section.

Muhammad Aufeef Chauhan
Copenhagen, December 2015

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

In this chapter, we provide an overview of the background, the research objectives, the research methods that are used to address the research objectives and the structure of this dissertation. We have organized this dissertation in multiple chapters. We describe a brief overview of the research that is presented in different chapters of this dissertation and references to research papers that have been published as an outcome of the presented research.

Parts of this chapter have been presented in [2].

1.1. Software Architecture (SA)

The growth in the complexity of a specific domain raises the need to have higher levels of abstractions that are easily understandable by a majority of the stakeholders associated with the domain. The higher-level abstractions act as a tool of communication between stakeholders so that they can express their concerns regarding a specific system. Like other domains, a growth in the complexity of a software system also raises the need to have higher levels of software abstraction. For software system the higher-level abstractions are covered under the umbrella of software architecture. There are many definitions and perspectives on software architecture.

Clements et al. [3] define software architecture as *"the structure of the components of a system, their relationships, and principles and guidelines governing their design and evolution over time"*.

Buschmann et al. [4] define software architecture as *"a description of the subsystems and components of a software system and relationships between them, and different views of the subsystems and components to show relevant functional and non-functional properties of a software system"*.

Bass et al. [5] define software architecture as *"a set of structures needed to reason about the system, which comprise software elements, relations among the elements, and properties of both the elements and the relations"*.

Gorton [6] elaborates the role of architecture as: *architecture defines structure, specifies communication among the sub-systems and components and addresses non-functional requirements of a software system*.

ISO/IEC/IEEE 42010:2011 architecture description standard [7] defines

software architecture as *"the recommended practices of the fundamental organization of a system, embodied components, their relationships to each other, their relationship to execution environment, and the principles governing design and evolution of the architecture"*.

Although software architecture is defined in various ways, there are common properties and elements in different software architecture definitions. By analyzing above-mentioned definitions of software architecture, following dimensions of the software architecture can be observed.

- Software architecture abstracts different elements of a software system and elaborates relationship between the elements.
- The elements and their relationships are abstracted with respect to different functional and quality requirements of the system.
- Software architecture elaborates the properties of components and their relations.
- Architecture defines system structures in terms of components and specifies communication between the components.
- Architecture guides the design, development and evolution of a software system over time.
- Architecture defines the recommended practices for design and organization of a software system.

It is clear from various software architecture definitions that software architecture is not only about defining and organizing software components, but that the process through which software architecture is analyzed, designed, represented and evaluated also plays a crucial role. Software architecture literature covers multiple models for software architecture design [8] including ***Attribute Driven Design*** [9], ***Rational Unified Process*** using ***4+1 views*** [10] and ***Business Architecture Process and Organization*** [11] that drive design of a software system. All the models focus on three common activities named architecture analysis, architecture synthesis and architecture evaluation [8]. Architecture analysis activity focuses on identifying architecture concerns, context, architecture significant requirements and candidate design solutions [8]. Architecture synthesis activity focuses on combining candidate architecture solutions corresponding to multiple architecture significant requirements, and architecture evaluation activity focuses on evaluating the effectiveness of candidate architecture solution and choosing the ones that best satisfy architecture significant requirements.

Architecture patterns and styles are a proven way to reuse intra and inter domain architecture and design knowledge [4]. Representing architecture of a specific system using the patterns and styles facilitates adoption of the architecture and its evolution. Pipes and filters, layers, and broker are commonly used examples of architecture patterns [4]. Many methods have been proposed for evaluation of software architecture including ***Software Architecture Analysis Method (SAAM)*** [12], ***Architecture Tradeoff Analysis Methods (ATAM)*** [13] and ***Quality-driven Architecture Design and Analysis Method (QADA)*** [14]. The choice of method to be used for the evaluation of a software architecture depends upon the goals of evaluation activity and the nature of the project.

Hence, it can be concluded that software architecture of a specific system should not only present components and relations among the components but also the guidelines for the design, evaluation and implementation of the system. Moreover, architecture design should be documented using architecture styles and patterns for its easy adoption and enhancement.

1.2. Software Reference Architecture (RA)

While a concrete software architecture aims to provide the design of a single system, a software reference architecture aims to "facilitate design and development of multiple systems of same nature and domain" [15]. Concrete software architectures are designed within a specific project and organizational context, and focus on well-defined business goals and requirements (both functional and quality requirements). On the contrary, reference software architectures are less defined and try to address generic business goals and requirements of a specific domain [15].

Bass et al. [5] describes software reference architecture as *"a division of functionality together with data flow between the pieces mapped onto software elements (that cooperatively implement the functionality) and the dataflow between the elements" of a specific domain*.

Avgeriou [16] explains that the description of software reference architecture is based upon: best practices of describing architectures of software intensive systems, the process that guides analysis, design and development of the reference architecture and how different components of the architecture are modeled. Avgeriou emphasizes that a software reference architecture should describe stakeholders' concerns in terms of different viewpoints, describe architecture using different architecture views [10], cover architecture quality characteristics of the domain and show design of the reference architecture using architecture patterns [16]. The use of architecture patterns to describe the reference architecture has also been emphasized by Angelov et al. [15].

Therefor as architecture patterns and styles are generic solutions to commonly occurring architecture design problems, their use in software reference architecture results in an architecture solution that addresses a generic set of functional and quality requirements of a specific domain. Reference models [5] also play a critical role in the design of the reference architectures [15].

Context, ***goal*** and ***design*** are three main dimensions of a software reference architecture [15]. The context dimension elaborates the stakeholders that can play a critical role in design of the reference architecture, potential uses of the reference architecture, the context in which a reference architecture is defined and whether the reference architecture is a preliminary proposal or a standardization effort [15]. The goal dimension elaborates objectives of defining the reference architecture [15]. The design dimension elaborates what components, protocols, algorithms and guidelines are proposed in a reference architecture, in how much detail different elements of the reference architecture are elaborated and what notations are used to represent a reference architecture [15].

Reference architectures are classified into different categories based upon the goals of the reference architecture. Angelov et al. [15] have proposed four major types of software reference architectures, as elaborated in following points.

- Type 1: Standardization of classical architectures that are designed to be implemented in a single organization. Standardization is performed by a single organization that intends to use the reference architecture.

- Type 2: Standardization of classical architectures that are designed to be implemented in multiple organizations. Standardization is performed by a group of organizations that intend to use the reference architecture.

- Type 3: Standardization of classical architectures to facilitate architecture design in multiple organizations and the standardization activity is carried out by an independent organization.

- Type 4: Standardization of preliminary architectures to facilitate architecture designs of architectures of systems that will be needed in future. These reference architectures are designed by an independent research center or a group of independent research centers.

Identification of reference architecture type is important as it determines the approach to be adopted for the construction of the reference architecture and how the reference architecture should be evaluated. Evaluation of a reference

architecture is specific as compared to evaluation of a concrete architecture because of number of reasons including: a generic nature of software reference architecture, unclear group of stakeholders, higher levels of architecture abstractions because of the generic nature of the reference architecture and ability of the reference architecture to address more architecture qualities as compared to a concrete architecture [17]. As a result, an effective evaluation strategy for the evaluation of a reference architecture does not only require tailoring of architecture evaluation methods such as ATAM [17], but can also require a combination of different architecture evaluation methods (for example combine different scenario based evaluation techniques including but not limited to SAAM, ATAM and QADA). Moreover, evaluating a reference architecture in terms of its applicability using a prototype implementation is also an effective way to demonstrate its feasibility [16].

1.3. *aaS Model of Cloud Computing

Cloud computing has become an active area of research and practice over the last few years. It is based on computing utility and service provisioning approaches. It offers organizations an opportunity to have on-demand scalability and flexibility of computing as well as storage resources [18-20]. This utility model enables organizations to save upfront investment costs that are needed for setting up and running large-scale computing infrastructure. It frees organizations from low-level infrastructure-related tasks and allows them to concentrate on their core business operations. This growth trend is supported by big players of IT including Amazon[21], Google, Microsoft and SalesForce[22]; that are providing cloud based infrastructure and services to consumers. Applications of heterogeneous domains ranging from social networking sites and gaming portals to scientific workflow systems and business applications are utilizing the power of the cloud computing platform [23].

Different people have different interpretations of the term cloud computing, and there are many definitions [24-26]. US National Institute of Standards and Technology (NIST) has a more comprehensive definition of cloud computing and defines it as "*A model for enabling convenient, on-demand network access to shared pool of configurable computing resources (e.g. storage, application services, servers and network) that can be rapidly provisioned and released with minimal management effort or service provider interaction*" [26]. Key feature of this paradigm is the ability to deliver services and infrastructure as pay-per-use basis [23]. Service level agreements (SLAs) are used for specification of QoS requirements between cloud service providers and consumers.

In order to achieve flexible hardware and software resource provisioning, cloud computing infrastructure should be capable of on-demand resource acquisition, accommodating billing schemes to charge users and resource publication through a single provider [27]. Cloud computing solutions offered by public cloud providers are broadly classified into three services and five deployment models [18, 19, 28-30]. The three categories of service models are: *Infrastructure as a Service (IaaS)*, *Platform as a Service (PaaS)* and *Software as a Service (SaaS)*. Five deployment models are: *public, private, hybrid, community and virtual private clouds*.

IaaS cloud provides abstraction to underlying computing, storage and network resources using virtualization technologies. It also provides basic software resources such as operating system for utilizing the virtualized hardware resources. IaaS poses additional overhead to applications and technical staff for monitoring and optimizing resources to meet QoS requirements specified in SLAs. IaaS has the advantage of support for customization. Additional tools and software can be installed as per requirements of the applications and end users. Amazon Elastic Cloud [31], Amazon Simple Storage Services [32], Eucalyptus [33] and OpenNebula [34] are example of IaaS cloud platforms. ***PaaS*** cloud provides application programmable interfaces (APIs) for developing applications. Application built using PaaS APIs, do not need to handle resource provisioning of underlying infrastructure. Google App Engine [35], Microsoft Azure platform [36] and SalesForce [22] are examples of the PaaS. Although PaaS provides support for seamless scalability and easy way to develop applications for cloud, it also has some disadvantages [27, 37]. One major disadvantage is that applications developed on PaaS are tightly coupled with the PaaS platform. Porting these applications on other platforms may require major refactoring and has negative impact on long-term evolution of the system. Enhancements in applications that are developed using a specific PaaS are also tightly coupled with new features supported by the PaaS provider. Moreover, as PaaS does not provide support for customization, it may not be straightforward to deploy application using multiple frameworks on PaaS because of unavailability of required frameworks. ***SaaS*** represents applications that are built on top of either IaaS or PaaS clouds and offer business solutions to end users. One of the key features of these applications is multi-tenancy. It enables single instance of the application to service a large number of organizations and end users. SaaS provides limited support for customization. Though some characteristics distinguish SaaS from PaaS, the boundary between PaaS and SaaS is blurry, and determining whether to classify a cloud-enabled solution as PaaS or SaaS depends on the context of the usage.

Public cloud represents cloud infrastructure and software resources maintained by an organization and is offered to end users for lease on basis of

some pricing model. End users can access infrastructure by using Internet. Amazon Elastic Compute Cloud (EC2) and Simple Storage Service (S3), Google App Engine and Microsoft Azure are examples of public clouds. ***Private cloud*** represents infrastructure and software resources that are maintained by an organization for its internal use. In some cases, organizations adopt a hybrid strategy and combine private infrastructure with public clouds. It is called ***hybrid cloud***. ***Virtual private cloud*** (VPC) [30] and ***community cloud*** [28] are build on top of the public and private clouds [27]. A VPC utilizes resources of a public cloud with additional features of a virtual private network. It provides support for customizable network topology and network security settings [27]. In some cases, organizations with shared business objectives decide to collaborate with each other and form a common cloud by combining their private clouds. It is referred as community cloud.

Different cloud deployment models have their advantages and disadvantages. Public clouds offer advantage to organizations of leasing resources from third parties only when needed and do not require investing in infrastructure. However, in public clouds, application and data is hosted at third party's premises, so the organization has less control over applications and data. In a private cloud, an organization has control over resources but it requires investment and maintenance of infrastructure and requires additional training of staff. A private cloud is more suitable with high data security and privacy requirements. The organizations with a private infrastructure also offload some processing on public clouds during peak hours using a hybrid approach. The hybrid approach enables the organizations to have their data on secure premises and utilize processing capabilities of cloud. This approach also has a drawback as it introduces latency delays as a result of network speed limits that may become a significant problem when public cloud infrastructure is at a distant geographic location. A community cloud provides more control over data and resources but have less flexibility of resource acquisition because of limited resource availability.

1.4. Research Motivations for Tools as a Service Workspace (TSPACE)

The research that is being presented in this dissertation is motivated by the need to provide a reference architecture that can be used to provision the ***Tools as a Service (TaaS)*** on demand as part of the cloud-enabled workspace. We refer to TaaS as a paradigm in which tools used to perform software analysis, design and implementation can be provisioned on demand [38]. Although TaaS and SaaS are similar in many aspects, the need for the tools to be able to perform as a part of a suite of tools distinguishes TaaS from SaaS. We refer ***Tools*** as a service ***workSPACE (TSPACE)*** as a set of tools are offered as a service as part of the cloud-enable workspace and formally define it as:

> **TSPACE** is *an aggregated platform that facilitates activity or task specific tools selection and provisioning on demand, provides support for integration among heterogeneous types of tools and artifacts managed by the tools in a TSPACE, and raises awareness of the operations that are performed on the artifacts using the provisioned tools [1].*

A specific runtime invocation of the TSPACE with a selected set of tools is referred as TSPACE instance. Our research effort has been motivated by the need to provide TSPACE reference architecture in terms of key specifications, TSPACE architectural design guidelines, detailed reference architecture design by providing reference architecture components and an evaluation of TSPACE architecture based on *aaS model. To limit the scope of the dissertation and to keep the discussion focused, we are considering tools that are used for software architecture analysis and design related activities as a specific case of TaaS. TSPACE reference architecture aims at utilizing underlying IaaS cloud resources to provision the tools.

1.5. Research Design

Software architecture primarily aims to provide decomposition of a software system into its constituents and relationships among the constituents. There can be multiple perspectives that govern and guide software architecture design activity. The process of designing a software architecture for different types of system domains share some commonalities in terms of the steps that are involved in the design activity and the views [10] used to represent different elements of a software architecture. However, designing software architectures for complex and emerging domains is a challenging task. Software reference architectures aim to guide the development of complex software systems for a specific domain by providing a generalized solution that can be adopted in different context, as explained in Section 1.2. The objective of TSPACE reference architecture is to provide an adoptable and extendable reference architecture that can be adopted for TaaS provisioning. In this section, we described the research objectives of the dissertation and the research approaches that have been adopted to address the research objectives along with concrete research steps that have been taken.

1.5.1. Research Objectives

The research goal (RG) of this dissertation is:

> **RG:** To provide a TSPACE reference architecture that can facilitate design and development of concrete TSPACE architectures to facilitate TaaS provisioning for software engineering tools in general and software architecting tools in particular.

We have chosen software architecting domain as a specific case of TaaS. The research goal is addressed by incorporating multiple research objectives (RO) into the research, as described in following points.

- **RO1:** The first objective of this research work is to investigate challenges associated with software architecting of cloud-enabled software systems and available solutions to address the challenges.

- **RO2:** The second objective is to identify critical design process stages that can lead to TSPACE reference architecture analysis, design, evaluation and implementation in context of a specific domain.

- **RO3:** The third objective is to identify main business, technological and quality requirements of TSPACE reference architecture.

- **RO4:** The fourth objective is to provide a conceptual model that can be used to identify and capture all the elements and relationships among the elements of TSPACE reference architecture.

- **RO5:** The fifth research objective is to provide details of TSPACE reference architecture in terms of reference architecture elements and relations between the elements.

- **RO6:** The sixth research objective is to present the value of the reference architecture presented in this dissertation in terms of its compliance with the research objectives and its applicability.

TSPACE reference architecture can be classified as Type 4 reference architecture (Section 1.2) because the reference architecture development is carried out at a research institute (i.e. IT University of Copenhagen) and the proposed reference architecture provides standardizations for the development of future software systems (i.e. TaaS Workspaces).

1.5.2. Research Approach and Research Steps

The research approach follows the principle of separation of concerns, and different research steps (RS) are taken to achieve the research objectives.

1. *Performing a systematic literature review (SLR) on software architecture challenges and solutions for cloud-enabled systems (RS1)*

To identify software architecture challenges and solutions for cloud-enabled software systems (**RO1**), a systematic literature review (SLR) was performed following SLR guidelines that are presented by Kitchenham [39]. Through SLR we have identified and reviewed the studies that are reporting the software architecture and architecture-related challenges and solutions. We have synthesized the identified challenges and corresponding solutions into multiple categories. Some of the identified challenges are used as a source of quality requirements of TSPACE reference architecture, and corresponding synthesized solutions are used to achieve the quality requirements in TSPACE (partially addresses **RO3**).

2. *Analyzing reference architecture design and documentation approaches and elaborate a process framework that can lead to TSPACE reference architecture concepts, identify relationships between the concepts and guide the design, development and evaluation of TSPACE reference architecture (RS2)*

The process framework guides TSPACE reference architecture by elaborating the design process stages (**RO2**) to identify TSPACE elements, a reference architecture documentation approach suitable for TSPACE reference architecture, domain models that can lead to the analysis, design and evaluation of the reference architecture. We have specifically focused on *aaS constrains of TSPACE reference architecture while elaborating the process framework.

3. *Performing structured review of the literature on the tools to identify business, technological and some of the quality requirements of TSPACE (RS3)*

This step leads to address third research objective (**RO3**).

4. *Analyzing domain models and standardized reference models of software architecture domain to identify TSPACE elements, relation between the elements and conceptual models (RS4)*

This step is used to address fourth research objective (**RO4**) and provides a foundation for TSPACE reference architecture detailed design.

5. *Designing and evaluating the reference architecture (RS5)*

The TSPACE reference architecture requirements (***RS3***) and conceptual domain model (***RS4***) are used to identify multiple abstraction levels of TSPACE reference architecture. A tailored process framework (***RS2***) has been

used to guide the design of TSPACE reference architecture. Well known architecture design principles, architecture styles and patterns, and some of the solutions that are identified as a result of ***RS1*** have been used for the detailed design of TSPACE reference architecture (**RO5**). TSPACE reference architecture consists of architecture abstractions for three core functionalities of TSPACE: (a) tools selection and provisioning, (b) runtime tools bundling and process-centric and semantic integration among the tools (c) workspace specific characteristics including awareness of the operations that are performed on the artifacts. Multiple reference architecture design strategies have been adopted. For tools selection and provisioning, findings from the existing literature and solutions have been synthesized and tailored according to specific needs of TSPACE reference architecture. For runtime tools bundling and workspace specific characteristics, architecture solutions have been proposed that can facilitate to incorporate business, functional and quality characteristics in TSPACE reference architecture. Multiple architecture evaluation methods [12-14] and reference architecture evaluation approaches [16, 17, 40] have been studied to devise an optimal strategy for the evaluation (**RO6**) of proposed TSPACE reference architecture. A prototype of the reference architecture using a selected set of tools that are used for software architecting has been developed to demonstrate the feasibility and applicability of the proposed reference architecture.

Figure 1 depicts pictorial representation of TSPACE reference architecture design research process.

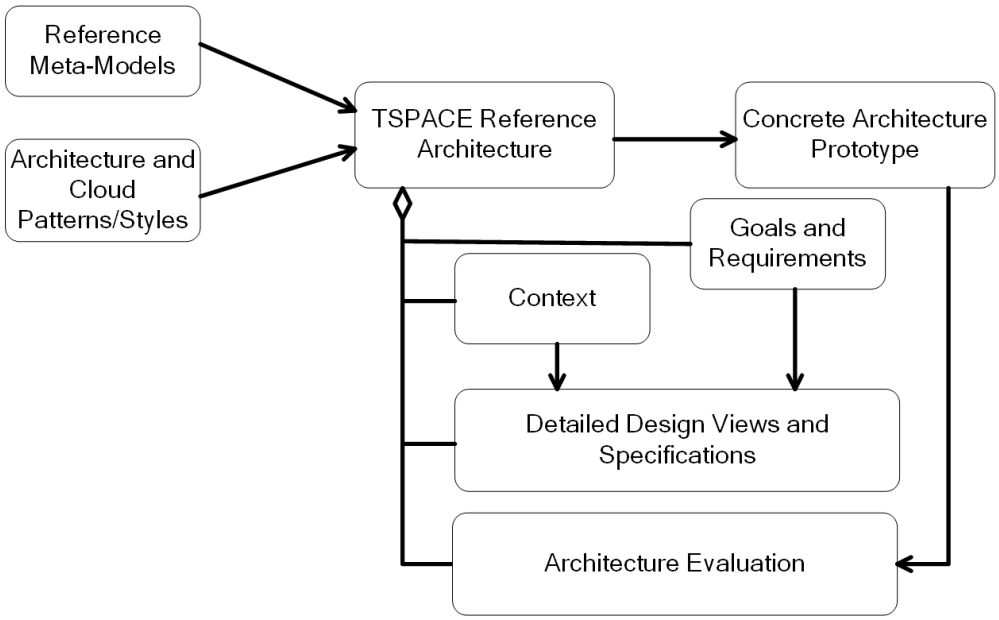


Figure 1: TSPACE Reference Architecture Research

1.6. Dissertation Structure

This dissertation consists of multiple chapters. The structure of the dissertation follows the order of the research steps defined in Section 1.5.2. Because TSPACE reference architecture focuses on *aaS model, literature related to architecting of cloud-enabled systems has been discussed in a separate chapter (Chapter 2). All the other chapters have a related work section to discuss the findings reported in the literature related to the concepts discussed in the respective chapter. This dissertation is organized as follows:

Chapter 2 identifies software architecture challenges and solutions for cloud-enables software systems (**RO1**) and presents a systematic literature review (SLR) that is performed following SLR guidelines presented by Kitchenham [39]. Through SLR we have identified and reviewed 111 studies that are published as journal papers and are reporting the software architecture and architecture-related challenges and solutions. We have classified the challenges into 44 unique categories and have synthesized the solutions discussed in the selected studies of each category. Some of the identified challenges are used as a source of quality requirements of TSPACE reference architecture (**RO3**) and corresponding synthesized solutions are used to incorporate the quality requirements in TSPACE reference architecture design (**RO5**). As industrial cloud providers play a major role in cloud computing adoption, the findings from SLR have also been discussed in terms of their relations with commercial and open source cloud environments. Parts of this chapter have been presented in [41, 42].

Chapter 3 describes TSPACE reference architecture development process framework. Various reference architecture design methods [15, 16, 43, 44] and evaluation approaches [16, 17, 40, 45] have been considered to develop the tailored process framework. We have also utilized our experiences with designing cloud-based systems [27, 37, 46] while identifying different stages of the process. Software architecting domain models and TSPACE requirements are used to identify elements of the TSPACE reference architecture, the structure and relationship among the elements and meta-model for TSPACE design. TSPACE elements are used as a foundation for TSPACE functional demarcation, analysis of TSPACE *aaS requirements and detailed design of the reference architecture and its components. The process framework also guides TSPACE reference architecture evaluation, implementation and instantiation. Information presented in this chapter addresses second research objective (**RO2**).

Chapter 4 describes TSPACE reference architecture business paradigm and functional and quality requirements. TSPACE reference architecture documentation requirements emphasize describing reference architecture in

terms of its context, goals, detailed architecture design, reference architecture evaluation and guidelines for its instantiations. TSPACE functional requirements describe tools selection, provisioning, bundling, integration needs and workspace-specific characteristics. Quality requirements are described in two categories: quality requirements of TSPACE system and quality requirements of TSPACE reference architecture. This chapter addresses third research objective (**RO3**). Parts of this chapter have been presented in [38].

Chapter 5 discusses information structuring needs of the TSPACE. Software architecture domain models IEEE 1471-2000 [47] and ISO/IEC/IEEE 42010:2011 [7] have been used as a baseline to identify elements and relations among the elements. The domain models are then extended and tailored for the specific needs of TSPACE. An ontology-driven approach [48] has been adopted. The approach consists of a suite of ontologies to characterize activities, tasks and artifacts, and to capture stakeholders' requirements, tools' features, and methods to provide semantic integration among artifacts that are consumed or produced by the tools. The ontologies provide mechanisms to raise awareness (artifacts' addition, modification and conflicts) of the operations that are performed on the artifacts using the tools. A selected set of algorithms that use ontologies for TSPACE operations has also been presented. The research presented in Chapter 5 addresses fourth research objective (**RO4**). The parts of this chapter have been presented in [49].

Chapter 6 presents detailed design of TSPACE reference architecture. The reference architecture has been designed by leveraging well-known design principles, architecture styles/patterns [5, 50] and has been documented using a views-based approach [51]. The reference architecture has been presented in terms of its context, goals and design elements with respect to the requirements, design tactics, and different components of the reference architecture at multiple levels of abstraction. The detailed design of the selected components is presented using multiple views [51]. The details of the evaluation of the reference architecture, its prototype implementation and the software architecting tools used in the prototype are also described. The research presented in this chapter addresses fifth research objective (**RO5**). The parts of this chapter have been presented in [1, 38, 52].

Finally, **Chapter 7** concludes this dissertation by presenting our experiences with designing TSPACE reference architecture, lessons learned and directions for future work.

1.7. Contributions and Demarcation

This section describes the contributions of the research that is being presented in this dissertation in terms of publications and describes scope of the research.

1.7.1. Research Output in terms of Publications

The chapters of this dissertation are based on published and/or submitted work. This section lists the publications on which this dissertation is based on (is also summarized in Table 1). The publications are also cited in the beginning of each chapter for a quick reference.

- [2] M. A. Chauhan, "A reference architecture for providing tools as a service to support global software development," in *Proceedings of 2014 IEEE/IFIP Conference on Software Architecture (WICSA) Companion Volume*, Sydney, Australia, 2014.

- [38] M. A. Chauhan and M. Ali Babar, "Cloud infrastructure for providing tools as a service: quality attributes and potential solutions," in *Proceedings of 2012 WICSA/ECSA Companion Volume*, Helsinki, Finland, 2012.

- [53] M. A. Chauhan and M. Ali Babar, "Towards a Reference Architecture to Provision Tools as a Service for Global Software Development," in *Proceedings of 2014 IEEE/IFIP Conference on Software Architecture (WICSA)*, Sydney, Australia, 2014.

- [1] M. A. Chauhan, M. Ali Babar, and Q. Z. Sheng, "A Reference Architecture for a Cloud-Based Tools as a Service Workspace," in *Proceedings of 2015 IEEE Conference on Service Computing (SCC)*, New York, USA, 2015.

- [41] M. A. Chauhan, M. Ali Babar, and B. Benatallah, "Architecting Cloud-Enabled Systems: A Systematic Survey of Challenges and Solutions," *Software: Practice and Experience Journal*.

- [49] M. A. Chauhan, M. Ali Babar, Q.Z. Sheng, "Reference Architecture for Tools as a Service Workspace: Meta-model, Ontologies and Design Elements," *Future Generation Computer Systems Journal*.

- [42] M. A. Chauhan and M. Ali Babar, "A Systematic Mapping Study of Software Architectures for Cloud Based Systems," *Technical Report TR-2014-175,* IT University of Copenhagen, 2014.

Additional Publication: Following publication is not included in this disertation as it is not directly related to the research that is being presented. However, some of the findings from this publication have been used while desribing reference architecture development process framework that is presented in Chapter 3. In this study we have described process guidelines for migrating existing software systems to cloud computing paradigm.

- [46] M. A. Chauhan and M. Ali Babar, "Towards Process Support for Migrating Applications to Cloud Computing," in *Proceedings of 2012 IEEE International Conference on Cloud and Service Computing (CSC)*, Shanghai, China, 2012.

Table 1: Publication Summary

Venue	Reference	Peer-reviewed	Published	Under Review	Contributions
WICSA'2014	[2]	✓	✓		Research Objectives and Requirements
NordiCloud'2012 (WICSA/ECSA' 2012)	[38]	✓	✓		Functional and Quality Requirements, and Architecture Design Tactics
WICSA'2014	[53]	✓	✓		Reference Architecture Details
IEEE SCC'15	[1]	✓	✓		Reference Architecture Details and Views
SPE Journal	[41]	✓		✓	Software Architecting Challenges and Solutions for Cloud-enabled Systems
FGCS Journal	[49]	✓		✓	Reference Architecture Meta-models, TSPACE Ontologies, Algorithms and Architecture Views
Technical Report	[42]	✗	✓		Architecture Styles and Patterns for Cloud-enabled software systems

1.7.2. Demarcation

Although TSPACE is designed to facilitate a large variety of tools that can be used to perform various software engineering activities, to confine the discussion and keep the scope of this dissertation focused, we have considered software architecting domain and the tools that are used to perform software architecting related activities. The tools that are used to perform software engineering activities in general and software architecting activities in particular are designed and implemented using different technological paradigms [54] and often four to five tools are required to perform a specific software engineering activity [55]. Although diversified nature of the tools has been considered while designing TSPACE reference architecture, design and implementation of individual tools lie outside the scope of this dissertation. Hence, an implementation of TSPACE reference architecture can

be considered as a middleware platform that can bundle the tools together and provision the tools as part of a cloud-enabled workspace. TSPACE acts as a bridge between different types of tools and facilitates tools' operations in the workspace. As a result, the functional and quality requirements that are considered while designing TSPACE reference architecture do not encompass the features that can be provided by an individual tool or a group of tools bundled together in a TSPACE instance. We have only focused on functional and quality characteristics of TSPACE. TSPACE utilizes cloud resources from underlying IaaS clouds to provision the tools as well as its own components and services.

Chapter 2. Architecting Cloud-Enabled Systems: A Systematic Survey of Challenges and Solutions

In this chapter, we provide an overview of software architecting challenges for cloud-enabled systems along with high-level solutions. The literature on the challenges of and potential solutions to architecting cloud-based systems is rapidly growing, but is scattered. It is important to systematically analyze and synthesize the existing research on architecting cloud-based software systems in order to build a cohesive body of knowledge of the reported challenges and solutions. We have systematically identified and reviewed 111 journal papers that report architecture related challenges and solutions for cloud-based software systems. This chapter reports the methodological details, findings, and implications of our systematic review that has enabled us to identify 44 unique categories of challenges and associated solutions for architecting cloud-based software systems. We assert that the identified challenges and solutions classified into the categories form a body of knowledge that can be leveraged for designing or evaluating software architectures for cloud-based systems. Our key conclusions are that a large number of primary studies focus on middleware services aimed at achieving scalability, performance, response time and efficient resource optimization. Architecting cloud-based systems presents unique challenges as the systems to be designed range from pervasive embedded systems and enterprise applications to smart devices with Internet of Things (IoTs). We also conclude that there is a huge potential of research on architecting cloud-based systems in areas related to green computing, energy efficient systems, mobile cloud computing and IoTs.

Parts of this chapter have been presented in [41, 42].

2.1. Introduction

The increasing popularity and adoption of Cloud Computing has also surfaced a large number of challenges that need appropriate and resilient solutions. One of the key challenges of Cloud Computing is designing, evaluating and implementing suitable architectural solutions for cloud-enabled software intensive systems and services. Like in any other large-scale software intensive system, Software Architecture (SA) plays a vital role in cloud-enabled systems. The role of SA in cloud computing needs to be well-understood in terms of how SA can help to design cloud-based systems and can facilitate the bridging of the gap between higher-level abstractions and low-level algorithmic details. Like many other areas of Cloud Computing, researchers have started conducting extensive and systematic research on identifying the key challenges of and devising appropriate solutions to address

the challenges of architecting cloud-based systems. Being an emerging area of research and practice, the reported literature on SA challenges and solutions for cloud-enabled systems is growing fast but is scattered. It is difficult for researchers and practitioners to gain an easy access to systematically identified peer-reviewed studies reporting challenges and solutions for cloud-based systems. Whilst there have been some reviews on Cloud Computing, to the best of our knowledge, there has been no effort to systematically identify and rigorously analyze and report the literature on SA-related challenges and solutions for cloud-enabled systems. In order to fill this gap, we decided to conduct a Systematic Literature Review (SLR) [39] of SA-related challenges and solutions to cloud-based software systems. The primary objective of this research is:

> ***To provide a systematic map and review of literature related to software architecture for cloud-enabled systems and to analyze and synthesize the selected primary studies in order to identify: (i) the challenges that need to be addressed for architecting cloud-enabled systems and (ii) the key attributes of the solutions that are proposed for addressing the identified challenges.***

This primary objective of our SLR has been operationalized into several questions that this research sought to answer. The research questions of this study and their respective rationale have been reported in Table 1. We have designed and evaluated the research protocol using the SLR guidelines reported in [39]. Our study planning, execution, and reporting were guided by our experiences of using and extending SLR methodology and the reports on best practices of and lessons learnt from SLR [56-58]. The primary contributions of this work include:

- A systematic review of the state of the art of challenges of and solutions to architecting cloud-based software systems. The systematically discovered and synthesized knowledge can be leveraged by the practitioners for designing and evaluating appropriate architecture for cloud-based software intensive systems.
- A taxonomy of SA research on Cloud-based systems for studying and categorizing the identified challenges and solutions related to architecting cloud-based systems. The main categories in the taxonomy are: Resource and Service Management, Workflow Management, Service Level Agreement (SLA) Compliance and Energy Awareness. These main categories have been further subdivided into subcategories for analyzing and reporting the findings from this review.
- Identification of a set of quality attributes that have been frequently reported in the context of cloud-enabled software systems; these quality

attributes can be used as guide for designing and evaluating architectures of the cloud-based systems, especially platforms and applications.

This chapter is organized as follows. Section 2.2 provides an overview of the methodology, research objectives and the approach used to synthesize the findings. Section 2.3 describes the selected studies' classification into different research themes and analyses the findings corresponding to the research objectives. Section 2.4 provides a comprehensive overview of the identified architecture challenges and synthesizes the corresponding solutions. Section 2.5 provides a perspective on commercial cloud offerings versus research approach. Section 2.6 describes how the threats to validity were addressed while conducting the presented research. Section 2.7 provides concluding remarks.

2.2. Research Methodology

We used a Systematic Literature Review (SLR) [39] method. An SLR is a systematic and repeatable research process to identify, extract, assess, synthesize and report all available evidence (or information) on a particular research topic (i.e., architectures for cloud-based systems). We used SLR because we intended to carry out and report creditable analysis and evaluation of the published literature on SA-related challenges and solutions for cloud-based systems. Our research began by systematically designing and reviewing and implementing research protocols. Following are the activities and artifacts of this research study.

2.2.1. Research Protocols

The research study protocol included research study background and motivation, research objectives, research questions, criteria for inclusion and exclusion of target studies, search strategies, selection of target electronic data sources along with customized search string for each data source, detail of search and selection process for relevant publications, and data extraction and synthesis. The protocol also specified a set of measures to assess the quality of the selected studies.

2.2.2. Research Questions

Our research questions were derived from the objectives of our study. Table 2 presents the research questions and their respective rationale.

Table 2: Research Questions and their respective Rationale

Research Questions and Rationale		
ID	**Research Questions**	**Rationale**
RQ1	What are different dimensions of software architecture for cloud-based systems that are addressed by researchers?	This research questions is aimed at identifying different areas of research focused by the cloud computing research community.
RQ2	What are the publication venues and trends of studies on SA of Cloud Computing?	This research question aims at highlighting the important publication venues of the cloud computing research and provide information on the research publication trends.
RQ3	Which cloud platforms and deployment models are used for implementation and evaluation of the proposed solutions?	This research question aims at highlighting cloud environments and deployment models popular among researchers for implementing and testing their solutions.
RQ4	What quality attributes are primarily focused for evaluation of the proposed solution?	This question aims at identifying the important quality attributes that have attracted the research efforts for architectural challenges and solutions for cloud-based systems.
RQ5	What is maturity level of published studies and what is reliability level of proposed solutions?	This question determines the maturity and reliability of the research published in the selected studies.
RQ6	What are the major challenges of and solutions for designing and implementing architecture of cloud-based systems?	This question is aimed at identifying the main challenges of and solutions reported in the literature on SA for cloud-based systems.

2.2.3. Target Data Sources

We performed searches on electronic databases that had been accessible online. We did not look for information in books and printed sources. Table 3 presents a list of our selected electronic databases. These four electronic databases are expected to cover most of the peer-reviewed literature on software engineering and computer sciences according to our work reported in [56, 59].

Table 3: Electronic Data Sources

Data Sources	
Electronic Database	**URI**
IEEE	http://ieeexplore.ieee.org/Xplore/
ACM	http://dl.acm.org/
Springer	http://www.springerlink.com/
ScienceDirect	http://www.sciencedirect.com/

2.2.4. Search Query

We performed searches on the chosen digital libraries to retrieve the relevant studies. We used following criteria to obtain the keywords for the search queries.

- Derived the major terms from the research objectives and the research questions.
- Identified alternatives and related terms. Literature related to Cloud Computing is often referred with different cloud service models such as infrastructure as a service, software as a service and platform as a service. Therefore, we included the names of different service models while preparing our search string. We also included different names used for Cloud Computing including cloud and cloud technologies in our search string.
- Used Boolean "or" and "and" operators to link the major terms of the strings for target databases when the search engines allowed the use of Boolean operators.
- Performed pilot searches to validate the effectiveness of the constructed search queries.

The following search string represents our generic search query based upon terms related to cloud computing and combining AND and OR operators.
("cloud computing" OR "cloud" OR "cloud technologies") AND ("architecture" OR "architectures" OR "software as a service" OR "SaaS" OR "platform as a service" OR "PaaS" OR "infrastructure as a service" OR "IaaS")

We included "* as a service" and "*aaS" in the search query to minimize the risk that a potentially relevant paper is missed during our search. We did not use "AND" clause with architecture keyword because our initial searches had revealed that there were not many studies that were using architecture keyword in title or abstract. We customized the generic search query

according to standard of each of the target electronic database to get more accurate search results. We performed searches using customized search strings on documents' metadata including both title and abstract.

2.2.5. Inclusion and Exclusion Criteria

We selected papers published in peer-reviewed journals till May 2015. We decided to include only journal papers so that our review includes high quality studies reporting mature and complete research results that are usually published in journals. We excluded the studies that were not related to SA on Cloud Computing or did not address any aspect of SA. In case of different versions of the same paper were published, we included only the most comprehensive version. Table 4 shows our inclusion and exclusion criteria for selecting the papers in this review.

Table 4: Inclusion and Exclusion Criteria

Study Selection Criteria	
Inclusion Criteria	**Exclusion Criteria**
Material related to SA of cloud-based systems published till May 2015, focusing on architecture related challenges and solutions.	Material not related to SA of cloud-based systems or published after May 2015.
Published in peer-reviewed venues.	Published in non-peer reviewed venues.
Research papers published in journals.	Conference papers, workshops, books, panel discussions, presented slides, prefaces, tutorials and book reviews.
Material published in English language.	Papers published in languages other than English.

2.2.6. Search and Study Selection Process

SLRs usually take a long time to complete and report. Since we were targeting a topic that is increasingly evolving in terms of technological advancements and adoption, we decided to search, select, and review the relevant papers in multiple stages. We started the searches in July 2011 and used those searches as pilot. These searches provided us with the first set of papers that were selected and reviewed to gain an initial understanding of the literature to be reviewed. The pilot searches were followed by three rounds of extended searches: April 2013, December 2013 and May 2015. Our multi-staged strategy to carry out this SLR enabled us to include a large number of papers

reporting mature research published in journals. Figure 2 presents a diagrammatic view of SLR stages and the studies selected in each stage.

For the first stage search process, the electronic databases mentioned in Table 3 were searched using the search query described earlier. We retrieved 1491 papers from all searches of the selected data sources. In the second stage, we filtered the papers according to the inclusion and exclusion criteria and selected 14 journal studies, which were rigorously analyzed to provide a foundation for the next three stages. We repeated the whole search process again in April 2013 to update our review and include the papers between 2011 and April 2013. After performing the searches and merging them together, we got 2529 papers. After filtering the studies according to inclusion and exclusion criteria, we selected 38 papers. We performed third and fourth round of searches by following the same process in December 2013 and May 2015. After combining the search results from previously selected papers and removing the duplicates, we selected 111 journal papers to be included in this review.

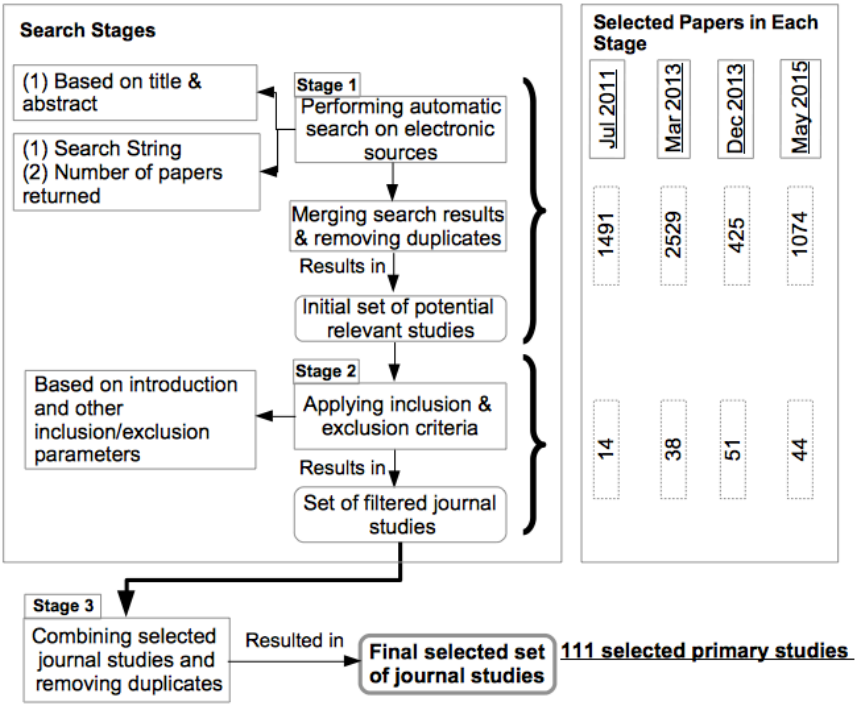


Figure 2: Search and Study Selection Process

2.2.7. Data Extraction, Synthesis and Classification

We performed the data extraction according to the data extraction form that is shown in Table 26 (Listing A). We used EndNote and Excel spreadsheets for

maintaining the bibliographic information of the studies and extracted data. The extracted data were organized in groups based on the main focus of this SLR.

We synthesized the data with respect to architecture challenges and corresponding solutions. Whilst the review is focused on architecture-level solutions, the information about algorithms proposed was also extracted and synthesized to guarantee completeness of the results as some of the challenges had been addressed by combining high-level architecture abstractions with low-level algorithmic strategies. Several approaches have been proposed for synthesizing qualitative data from a SLR (such as Noblit and Hare [60] and Cruzes and Dybå [61]). We used the multi-stage approach of thematic synthesis that has been recommended by Cruzes and Dybå [61]. According their guidelines, the data synthesis approach begins by identifying codes corresponding to the concepts of interest. Then the codes are translated into themes and sub-themes. In the last stage, relations between themes and sub-themes are investigated to create higher order themes. We developed a catalogue of codes. That catalogue consisted of multiple sub-catalogues. The codes were assigned to selected studies according to: the main challenges they were addressing, the venues that published the selected studies, different cloud environments that were utilized for evaluation of the solutions, different maturity stages of the studies and delivery model of the solutions proposed in the selected studies. The codes for architecture challenges and solutions were assigned to the selected study according to the main architecture challenges a particular study addressed, and were used to perform synthesis at two levels of abstraction: main categories of themes and sub-categories of themes. If a study addressed architecture challenges that belonged to more than one category, it was classified in both categories.

We used the line of argument approach [60] to combine the parts of the challenges and the solutions in order to provide a comprehensive overview of the challenges and associated solutions. We used reciprocal translation synthesis approach [60] in order to combine similar or related solutions. Section 2.3.1 describes the identified categories of the themes, the sub-themes and the corresponding primary studies included in the relevant categories.

Figure 3 presents the taxonomy that we used to address the research objectives. We built the taxonomy based on the studies included in the first round depicted in Figure 3. We classified the studies into different categories and subcategories based on the main theme of each of the studies. Each category and its subcategories are explored in term of the challenges and the proposed solutions to enable readers to have an in-depth view of the reported approaches, the quality attributes and the target deployment models used to identify suitability of the cloud environments with respect to a particular

dimension of architecture quality. An analysis of the evaluation criteria is used to measure the maturity of the proposed solutions.

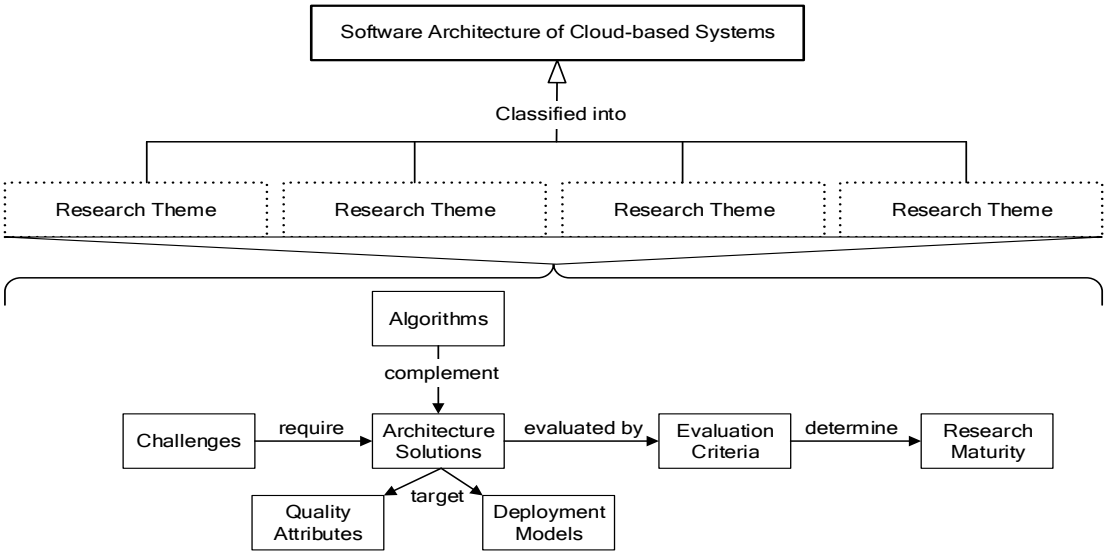


Figure 3: Taxonomy of SA Research on Cloud Computing

2.3. Results and Analysis

In this Section, we detail the classification of the reviewed studies, distribution of the studies over publication venues, cloud environments and deployment models, maturity and quality of the architecture solutions, and a map of the quality attributes used for evaluation of the proposed solutions in the reviewed papers.

2.3.1. Categories of Research Themes

We classified the selected studies into different categories based on the main focus of the studies. The studies have been classified according to their relevance to the themes as shown in Table 5. If a study belongs to more than one category, it has been assigned to multiple categories. However, while reporting the findings in Section 2.4, we have discussed the studies under one of the category to avoid repetition and keep the structure of this review uncluttered. Following is a brief description of the main categories.

- Studies classified into ***Resource and Service Management*** category provide solutions for managing applications and services on the cloud with respect to desired functional and quality requirements. The studies that are included in this category also provide solutions for protecting services and data that belong to multiple tenants.

- Studies classified into ***Workflow Management*** category provide solutions for managing distributed workflows for processing computing intensive and security sensitive data.
  Studies classified into ***Service Level Agreement (SLA) Compliance*** category provide solution to satisfy SLAs in cloud-based software systems. SLA compliance is treated as a separate category rather than making it a part of Resource and Service Management because papers included in this category of theme are not only providing resource management solutions but also providing solutions for SLA specific service discovery, monitoring pricing and billing.
- Studies classified into ***Energy Awareness*** category provide energy efficient solutions.

Table 5: Primary Studies' Distribution over Categories of Themes

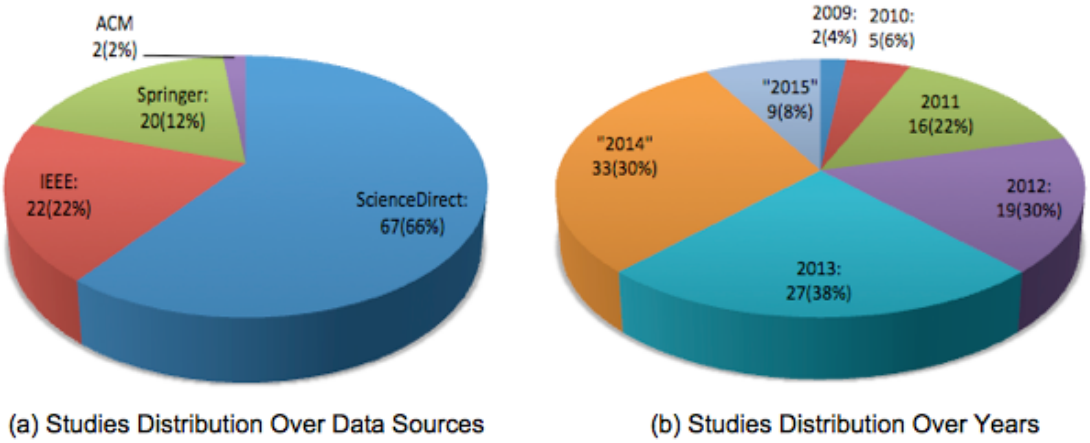
Software Architecture for Cloud Computing	
Resource and Service Management	
Quality-Specific Resource Provisioning and Management	[PS1][PS3][PS4][PS10][PS14][PS17] [PS20][PS27][PS31][PS41][PS42][PS45] [PS47] [PS48][PS51][PS54] [PS56][PS63] [PS66][PS69][PS73][PS78][PS79][PS82] [PS86][PS87][PS89][PS95][PS97][PS98] [PS102][PS103][PS107]
Pervasive Embedded Networks	[PS22][PS23][PS30][PS68][PS109]
Cloud Federation	[PS13][PS37] [PS46][PS75][PS76][PS85][PS86] [PS92][PS99][PS100][PS104]
Cache Management	[PS39][PS77]
Support for Mobile Cloud-enabled Devices	[PS2][PS6][PS29][PS35][PS36][PS44] [PS83][PS105][PS111]
High Performance and Scientific Computing	[PS80][PS94][PS101]
Multi-tenant Environments	[PS7][PS70]
Data Protection	[PS28][PS49][PS50][PS91][PS108] [PS110]
Enterprise Service Bus	[PS90]
Architectures for Data Intensive Systems	[PS19][PS38][PS51][PS60]
Workflow Management	
Business Process Management	[PS26][PS40][PS52][PS71]
Workload Distribution and Resource Management	[PS8][PS11][PS34][PS53][PS58][PS60] [PS65][PS67][PS96]
Service Level Agreement (SLA) Compliance	
Service and Data Management	[PS12][PS15][PS18][PS43][PS45][PS51]

	[PS53][PS62][PS72] [PS81]
Resource Discovery and Monitoring	[PS16][PS21] [PS32][PS33][PS57][PS59][PS61] [PS64][PS88][PS93][PS106]
Pricing and Billing	[PS10]
Energy Awareness	
Energy Aware Resource Optimization	[PS5][PS9][PS24][PS25][PS55][PS84]
Energy Efficient Process	[PS74]

Subcategories corresponding to main categories of themes are elaborated in Section 2.4. Complete list of selected studies references is provided in Listing C.

2.3.2. Data Sources

Figure 4(a) shows the distribution of studies over digital libraries on which searches were performed. ScienceDirect is the most prominent source with 67 (66%) studies. IEEE is the second with 22 (22%) studies published followed by Springer with 20 (12%) studies. There are 2(2%) studies that are published by ACM.



(a) Studies Distribution Over Data Sources (b) Studies Distribution Over Years

Figure 4: Studies Distribution

2.3.3. Publications Over Years

Figure 4(b) shows study distribution with respect to publication years. We did not specify the lower bound when searches were performed. The results show that 2009 is the first year with 2 journal papers published discussing SA of Cloud-based systems. There is a significant increase in number of journal papers during the following years; 5 in 2010, 16 in 2011, 19 in 2012, 27 in 2013, 33 in 2014 and 9 in 2015. These results indicate that quantity of reported research on SA of Cloud-based system have increased over the years.

2.3.4. Number of Papers published in Different Journals

Table 6 shows prominent journal paper in which cloud software architecture related studies have been published. The table lists only those journals that have published two or more studies. *Journal of Future Generation Computer Systems (FGCS)* is at the top place with 36 studies as it is a prime venue to publish cloud-computing research. *Computing* journal is at second place with 6 studies. *IEEE Transactions on Service Computing*, *Journal of Systems and Software* and *Journal of Computer and System Sciences* are at third place with 4 studies published in each of them. *IEEE Transactions on Network and Service Management* has published three papers. All the remaining journals shown in Table 6 have published 2 papers each.

Table 6: Publication Venues

Study Distribution Over Publication Venues		
Publication Venue	**Publication Venue Abbreviation**	**No. of Studies**
Future Generation Computer Systems	FGCS	36
Computing	Computing	6
IEEE Transactions on Services Computing	ITSC	4
Journal of Systems and Software	JSS	4
Journal of Computer and System Sciences	JCSS	4
IEEE Transactions on Network and Service Management	ITNSM	3
IEEE Transactions on Consumer Electronics	ITCE	2
IEEE Journal of Biomedical and Health Informatics	IJBHI	2
Personal Ubiquitous Computing	PUC	2
IEEE Transactions on Emerging Topics in Computing	ITETC	2
China Communications	CC	2

2.3.5. Cloud Environments Used

The selected studies have used diversified cloud environments to implement and deploy proposed solutions as shown in Table 7. Amazon is on top of the list with 26 selected papers using Amazon as an underlying environment to implement their proposed solutions. Google App Engine and Windows Azure are at second and third place with 9 and 8 papers respectively using them as

underlying cloud environment for implementing the proposed solutions. There are also a significant numbers of papers using private cloud environments and local hardware infrastructure as a simulated cloud environment to test the proposed solutions.

Table 7: Cloud Environment used for research reported in the primary studies

Study Distribution with respect to Cloud Environments		
Cloud Environment	**No. Of Studies**	**Study References**
Amazon	26	[PS7][PS8][PS10][PS13][PS14][PS16] [PS29][PS33][PS42][PS44][PS48][PS50] [PS52][PS56][PS65][PS75][PS78][PS82] [PS84][PS86][PS88][PS91][PS93] [PS95][PS101][PS102]
Google App Engine	9	[PS29][PS30][PS40][PS48][PS49][PS80] [PS88][PS92][PS100]
Windows Azure	8	[PS6][PS33][PS37][PS38][PS48] [PS50][PS75][PS102]
Eucalyptus	6	[PS13][PS29][PS37][PS47][PS75][PS94]
Private Cloud Simulations	6	[PS10][PS14][PS17][PS35][PS71][PS91]
Cloud Simulation Toolkit	6	[PS5][PS8][PS51][PS57][PS64][PS102]
OpenStack	5	[PS1][PS7][PS63][PS79][PS93]
Xen/KVM Virtualization Environment	3	[PS61][PS78][PS97]
Rackspace	2	[PS33][PS102]
OpenNebula	2	[PS3][PS32]
OpenShift	2	[PS75][PS100]
Globus Toolkits	2	[PS12][PS54]
Cloud Storage Solution (Skydrive, Dropbox)	2	[PS50][PS92]
GoGrid	1	[PS102]
IC-Cloud	1	[PS39]
RESERVOIR	1	[PS20]
Alchemy.com	1	[PS29]
NeCTAR	1	[PS52]
Flixiscale Cloud Platform	1	[PS59]
DELL KACE,	1	[PS75]

CloudBees, dot-Cloud, Jelastic, Heroku, Appfog,		
TCloud	1	[PS82]
Nagios	1	[PS1]
WorkflowSIm	1	[PS53]
MMOG Simulator	1	[PS72]
Others (local machine cluster, local setting non cloud-based machines)	16	[PS9][PS15][PS19][PS21][PS24][PS36] [PS41][PS45] [PS55][PS62][PS70][PS77] [PS87][PS105][PS106][PS108]

2.3.6. Deployment Models Used in Studies

Cloud Computing solutions are offered for different deployment models. The reviewed studies reported solutions for not only the three commonly referred deployment models (i.e., public, private and hybrid), but some studies also used local infrastructure as a simulated cloud environment. Figure 5 shows the number of studies that used different deployment models or simulated cloud environments. Our study has revealed that Amazon Web Services (AWS), Windows Azure and Google App Engine are commonly used public cloud environments. Eucalyptus, OpenStack, Amazon WS and Windows Azure are common choices for building hybrid solutions. Eucalyptus, Flexiscale cloud platform, UC-Cloud and RESERVOIR have been used for private clouds. Grids, cloud simulations toolkits and local machines clusters have also been used for building and using simulated cloud environments.

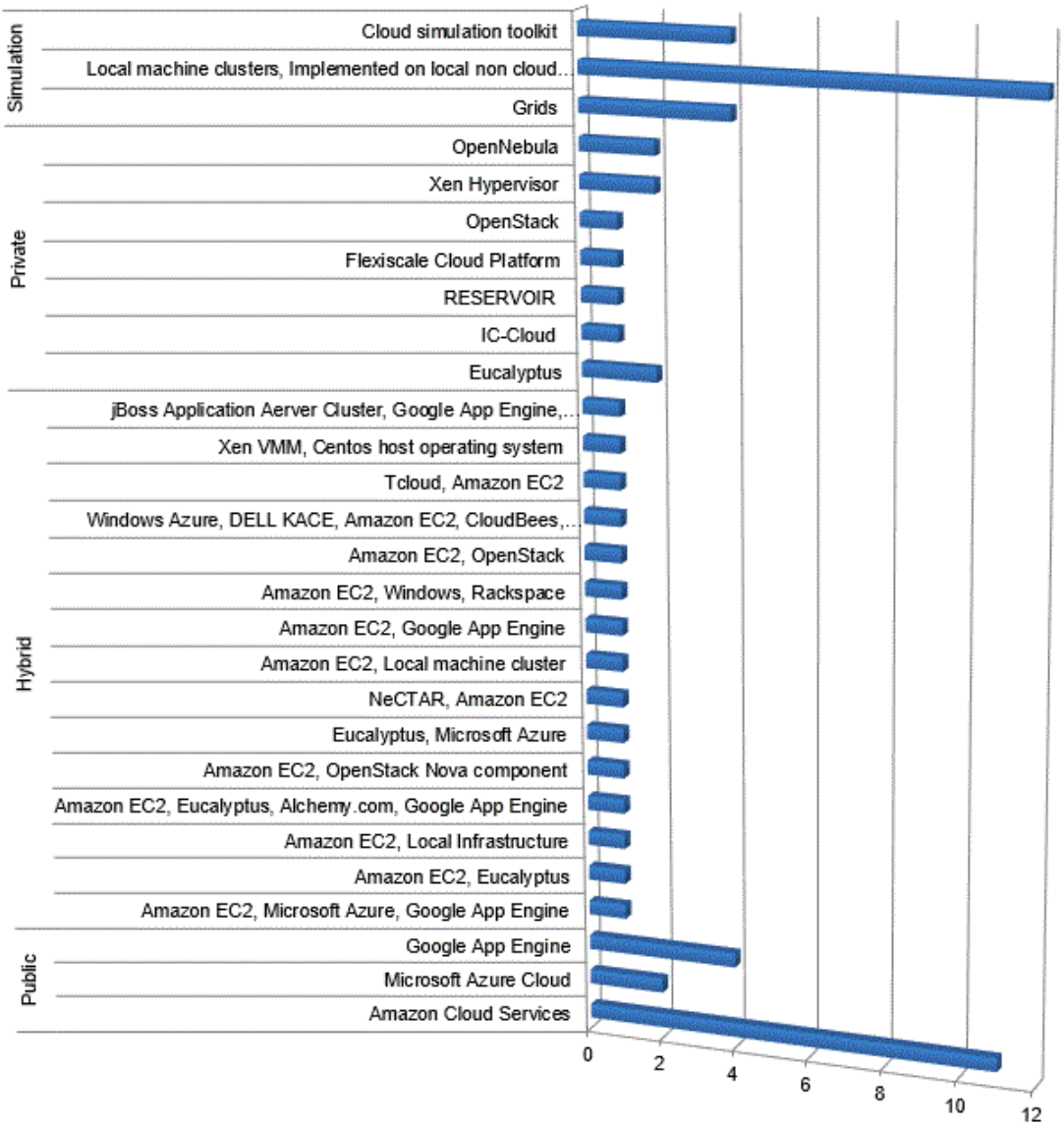


Figure 5: Studies Distribution with respect to Cloud Deployment Models

2.3.7. The Named Algorithmic Solutions

The selected studies have reported solutions to the challenges of cloud-enabled systems with primary focus on architecture centric solutions, algorithm centric solutions or a combination of both. During the selection of studies and extraction of data, we considered algorithmic aspects that complement architecture solutions for specific quality attributes. For example, to achieve scalability based on cost, the primary study [PS39] proposes an algorithm to complement architecture solution for scalability. There are 25 studies that provide algorithmic-centric solutions, whereas the number of

papers providing architecture-specific solutions and combination of algorithmic and architecture solutions are 56 and 30 respectively, as shown in Figure 6. Table 8 lists the studies providing algorithms specific to corresponding architecture solutions. The table does not show the algorithms that have not been reported with a particular name.

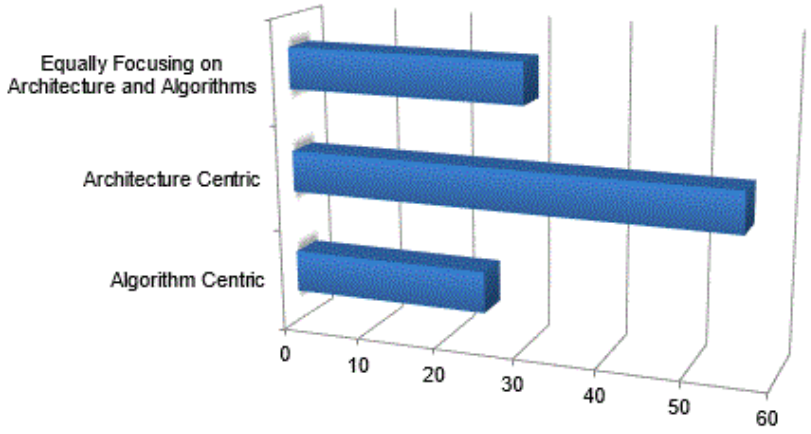


Figure 6: Studies Distribution with respect to Solutions Abstraction

Table 8: Algorithms used or proposed in the Selected Studies

Algorithms Used to Complement Architecture Based Solutions			
Algorithm-Centric			
SI-Cache	[PS77]	Task Management Algorithm	[PS95]
Cost-Aware Scaling Algorithm	[PS39]	Modified Vickrey Auction (MVA) and Continuous Double Auction (CDA)	[PS103]
Additive Homomorphic Probabilistic public key Encryption (AHPE)	[PS91]	HEFT, Greedy task queue and LATE Scheduling Algorithms	[PS8]
Proxy Re-encryption	[PS91]	Median-Edge Detector (MED) and Inter-Slice Predictor (ISP) algorithms	[PS17]
Additive Homomorphic Proxy Re-encryption	[PS91]	Partitioning Algorithm based on TABU Search	[PS34]
Extension to MapReduce	[PS38] [PS19] [PS60]	Place/Transition Petri nets based SBP Model	[PS71]
Feature Placement Algorithm	[PS70]	O-SLA and R-SLA Algorithms	[PS72]
Local Consistency and Global Consistency Auditing Algorithms	[PS82]	Energy Optimization Method based on Lyapunov	[PS25]

Meta-data index creation, query expansion, peer tracker and information retrieval algorithms	[PS87]	Genetic Algorithms	[PS31] [PS111]
Algorithm and Architecture-Centric			
Spot Instance-Aware Provisioning Algorithm	[PS14]	Assisted Anycast (AA)	[PS9]
Backtracking Algorithm	[PS66]	Multi-fixed Sequencer Protocol	[PS21]
Reactive & Predictive Algorithm Models	[PS47]	TF-IDF Algorithm	[PS88]
Service Provider Search Engine (SPSE) Algorithm	[PS107]	Virtual Machine Share Allocation Strategy and Thick Client Reserved Allocation Optimization Strategy	[PS45]
ProfminVM, ProfRS & ProfPD Algorithms	[PS102]	MT-PerfMod and Mt-ResElas Algorithms	[PS56]
Automatic Data Streaming Service (ADSS) Algorithm	[PS96]	LHS, Moldflow and GA Algorithms	[PS63]
Partitioned Balanced Time Scheduling (PBTS)	[PS11]	Posterioir Playfair Searchable Encryption (PPSE)	[PS86]
Modified Best Fit Decreasing (MBFD) Algorithm	[PS5]	Particle Swarm Optimization (PSO) Algorithm	[PS94]
Minimization of Migration (MM) Policy	[PS5]	Ciphertext Policy Attribute-based Encryption (CP-ABE)	[PS68]
Extension to MOGAs, NSGA-II & SPEA2 Algorithms	[PS41]	Sieving Algorithm	[PS53]
Full Anycast (FA)	[PS9]	BestFit, BFResvResource and BFReschedReq Algorithms	[PS64]

2.3.8. Quality Attributes Map

The reported solutions focus on achieving certain quality attributes. Figure 7 shows the distribution of the studies with respect to quality attributes that are shown on X-axis and the categories are shown on Y-axis. Scalability, performance, efficient resource utilization, CPU utilization and response time are the frequently reported quality attributes. The bubble at the intersection of X-axis and Y-axis shows the number of corresponding studies. For example, intersection of Performance and Resource and Service Management indicates that there are 22 studies that address reliability in the presented solutions.

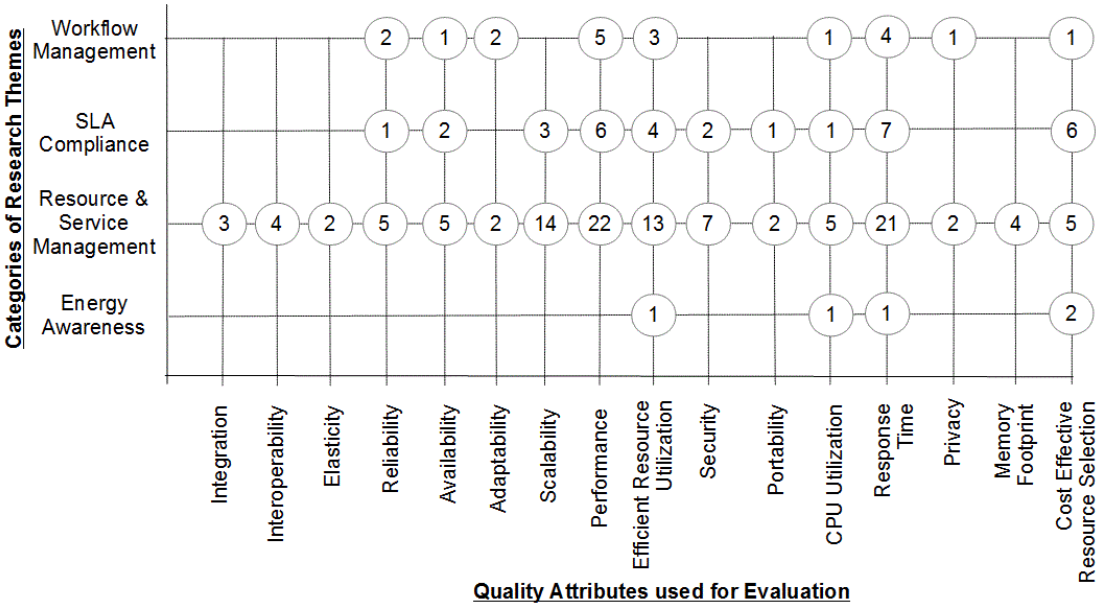


Figure 7: Quality Attributes distribution with respect to the Categories

2.3.9. Maturity of the Selected Studies

The identified solutions have been classified into five maturity stages. We decide the maturity of the solutions based on the implementation and evaluation reported in the selected paper using five maturity stages of the technology maturity model of Redwine and Riddle [62].

i) *Basic Research:* The studies that are classified in this maturity stage provide theoretical solutions for the problems but do not provide details on how the solutions can be implemented. That is, the studies neither provide implementation strategy nor evaluation of the proposed solutions.

ii) *Prototype Implementation:* The studies that are classified in this maturity stage propose solutions to the stated problems and provide prototype implementation but do not provide evaluation.

iii) *Evaluated in Simulated Environments:* The studies that are classified in this maturity stage provide evaluations of the proposed solutions in simulated environments.

iv) *Evaluated in Real Cloud Environments:* The studies that are classified in this maturity stage provide solutions to the described problems, describe implementation details and provide evaluation along with results by using commercially available private or public cloud environments.

v) *Popularization:* The studies that are classified in this maturity stage demonstrate the applicability of the proposed solutions in real world applications.

Table 9 shows the distribution of the studies into different maturity stages. Most of the studies are at maturity stage (iv) and the proposed solutions have been evaluated using commercial public or private clouds. There are only four studies that belong to the popularization stage.

Table 9: Study Distribution according to their Maturity Stages

Studies Maturity Map	
Maturity phase	**Study Reference**
Basic Research (without prototype implementation and evaluation)	[PS10][PS58][PS74][PS76][PS83][PS89][PS90][PS104] [PS110]
Prototype Implementation	[PS2][PS25][PS34][PS36][PS40][PS43][PS45][PS46] [PS51][PS60][PS62][PS65][PS68][PS85] [PS92][PS99] [PS109][PS111]
Simulations (Simulator implementation, Simulated platform)	[PS3][PS4][PS11][PS15][PS17][PS18][PS19][PS21] [PS24][PS28][PS31][PS41][PS53][PS54] [PS55][PS57] [PS64][PS67][PS70][PS72][PS73][PS81][PS87][PS96] [PS97][PS98][PS103][PS105][PS106][PS107] [PS108]
Evaluation in Real Private or Public Cloud Environment	[PS1][PS5][PS6][PS7][PS8][PS9][PS12][PS13][PS14] [PS16][PS20] [PS22][PS29][PS30][PS32][PS33][PS35] [PS37][PS39][PS42][PS44][PS47] [PS48][PS49][PS50] [PS52][PS56][PS59][PS61] [PS63][PS66][PS69][PS71] [PS75][PS77][PS78][PS79][PS80][PS82][PS84][PS86] [PS88][PS91] [PS93][PS94][PS95][PS100][PS101] [PS102]
Popularization	[PS23][PS26][PS27][PS38]

2.3.10. Quality Assessment of the Studies

We intended to assess the quality of the studies and their reliability in terms of quality of the proposed solutions. Dybå and Dingsøyr [63, 64] have proposed quality assessment criteria that were used to devise criteria for assessing the papers in our review. We added one more question (Q5). Table 10 shows the questions used to assess the quality of the included papers.

Table 10: Quality Assessment Criteria (a tailored version of the propositions from [63, 64])

Quality Assessment Criteria	
Id	**Question**
Q1	Is paper based upon research or is it merely a "lessons learned" report based upon expert opinion?
Q2	Is there a clear statement of aim of the research and research objectives?
Q3	Is there an adequate description of the evaluation context in which proposed solutions are evaluated?
Q4	Have the data been reported to support evaluation findings?
Q5	Are selected primary studies reporting limitations and areas for improvement?

We used ternary scale with values of yes, partial or no. To quantify our assessment criteria, we assigned values 1, 0.5 and 0 corresponding to yes, partial and no respectively. With Q1, we assessed if a primary study's findings were based on research or not (on opinions). Question 2 accessed whether or not a study reported clearly the research objectives and the challenges addressed. Question 3 was used to analyze if a study had clearly stated the evaluation setting in which the proposed solutions and the corresponding implementations were evaluated. Question 4 was used to evaluate whether or not a study had reported the data to support the evaluation results. Question 5 helped us to assess whether or not a study clearly reported the limitations of the proposed solutions, stating directions for improvement and future enhancements in the solutions. The answer to this question helped us to determine the future research scope of the problem areas discussed in a paper.

Table 27 (Listing B) shows the quality score for each of the papers in this SLR. All the papers have value score 1 for Q1 as none of the papers had reported lessons learned and experience reports. The average scores for Q2, Q3 and Q4 are 0.99, 0.86 and 0.86 respectively. It shows that aim and context of the research is clearly reported in all of the studies and the findings have been reported in an adequate manner. An average value score of Q5 is 0.64; that means many of the selected papers have not provided clear directions for their future work and enhancements. The aggregated average quality assessment score is 4.27, which is an indication of high-quality research on the reviewed topic being reported.

2.4. Analysis of the Challenges and Solutions

In this section, we provide a detailed analysis of the selected papers in terms of identified architecture challenges (problems) and the corresponding solutions. We have classified the papers reporting similar challenges in the same category of theme. When a paper has reported diversified set of challenges, that study has been classified into more than one category or subcategory. The primary studies have been further classified into sub-groups to have a specialized grouping of closely related topics. The challenges associated with the categories and subcategories have been tabulated in the following sections and the solutions corresponding to the problems have been described.

2.4.1. Resource and Service Management

The papers that have been classified into this category report the challenges related to architecture of cloud middleware and related services. The middleware acts as a bridge between applications and underlying IaaS cloud resources. Based on the extracted data, we analyzed and classified the challenges into ten subcategories. The following subsections describe the reported challenges and solutions.

2.4.1.1. Quality-Specific Resource Provisioning and Management

Challenges: One of the primary challenges associated with resource and service management on clouds is the incorporation of quality characteristics in the solutions. Table 11 shows the challenges associated with incorporated quality attributes.

Table 11: Resource Provisioning and Management - Quality Attribute

Problems		
Challenge	**Description**	**Study Reference**
Interoperability	Support for interoperability to simultaneously use multiple collaborative cloud services and data.	[PS41] [PS86]
Privacy	Trusted cloud services to process private data.	[PS4]
	Data placement strategies with respect to privacy requirements.	
	Trusted service providers (to achieve desired level of security and privacy).	[PS10]
	Compliance with legal and regulatory	

	requirements.	
Availability	Acquiring extra nodes for the high availability of applications.	[PS31]
	Isolating components deployed on distributed nodes.	
	Identifying bottlenecks associated with response time and resolving them automatically.	[PS47]
	Avoid overloading of infrastructure resources and SLA violations.	[PS102]
	Availability according to performance parameter.	[PS3]
Scalability	Accurately identifying traffic patterns for dynamic scalability.	[PS47]
	Autonomous and scalable self-organized clouds for utilizing publically acquired resources.	[PS103]
Portability	Improving portability of applications and services	[PS78]
	Avoiding vendor lock-in for low-level resources and application-level services.	
Security	Secure management of 3D medical images data.	[PS17]
	Security as a Service to support IaaS cloud users.	[PS97]
	Handling security liabilities of cloud providers and hosted virtual machines (VMs).	[PS98]
Elasticity	Resource elasticity according to QoS parameters.	[PS56]
Consistency	Consistency of the replicated services on multiple clouds.	[PS82]
Performance	Efficient multi-media information retrieval.	[PS87]
Adaptability	Support adaptability of service transmission environment according to specific QoS requirements in cloud and provide communication space specific to a customer's needs.	[PS73]

Solutions: To address the challenges of *simultaneously using multiple collaborative cloud services* to satisfy the needs of end users while maximizing profit, Hassan et al. [PS41] propose to share resources among partners. An architecture of Combinatorial Auction (CA) based Cloud Market

model, CACM, which provides an auction policy for a virtual organization-based dynamic cloud platform among cloud providers. Ribeiro et al. [PS86] present a proxy-based architecture to preserve interoperability, confidentiality and searchability of shared cross-enterprise documents. The proxy is based on Posterioir Playfair Searchable Encryption (PPSE) algorithm, which maintains data confidentiality by hiding search patterns.

Two studies [PS4] and [PS10] have proposed an architecture to handle the privacy challenges (data hosting on secure places and trusted service provisioning) using trusted cloud services, data placement strategies and trusted service providers. In order to overcome the challenges of trusted cloud services and data placement strategies, a study [PS4] presents a platform to allow home services to be selectively opened to remote users and semi-trusted external services. To tackle the issues of trusted service providers and legal and regulatory compliance, Buyya et al. [PS10] propose a concept of cloud market, where users can interact with the market and make request for resources according to the applications' needs with the help of cloud broker. The cloud broker facilitates service selection for end users.

To counter the challenges of acquiring extra nodes for the high availability of applications and isolating components deployed on distributed nodes, Frincu [PS31] presents an architecture-centric solution which finds the optimal number of component types needed on nodes so that every type is presented on every allocated node by using Genetic Algorithms. Iqbal et al. [PS47] present an architecture-centric solution for availability by automatically detecting the bottlenecks associated with response time using heuristics and active profiling of CPU utilization. Wu et al. [PS103] present two economic strategies for scalable and autonomous resource allocation mechanisms: (i) Modified Vickrey Auction (MVA) when the resources are sufficient and (ii) Continuous Double Auction (CDA) when resources are insufficient. By dynamically negotiating resources among Cloud Coordinators in the InterCloud environment, Wu et al. [PS102] intend to solve the problem regarding overloading resources and SLA violations. An et al. [PS3] present a publisher/subscriber based replication framework for autonomous virtual machines management using different types of performance and availability mechanisms. Petcu et al. [PS78] present a layered architecture to increase portability of the applications among cloud environments and to avoid vendor lock-in. The architecture proposes loosely coupled applications layers, cloud neutral APIs and unified resource representation from multiple cloud environments.

Castiglione et al. [PS17] present a security architecture for dynamic and adaptive security 3D medical data imagery using security watermarks in images. Varadharajan et al. [PS97] present a security architecture to provide a

baseline security for protecting cloud infrastructure. The architecture is based on Service Provider Attach Detection (SPAD) and Tenant-Specific Attach Detection (TSAD) approaches. Vera-del-Campo et al. [PS98] present DocCloud security architecture that focuses on plausible deniability, anonymity of indexer, recommenders and intermediate nodes, and oblivious routing. Kaur et al. [PS56] present a framework for dynamic scalability of cloud resources based on desired performance parameters by examining incoming request patterns and their corresponding response rates. Qin et al. [PS82] present a two level auditing architecture, which is based on heuristic auditing strategy and looks for commonalities of violations and staleness of the data so that the users can verify data consistency. Rocha et al. [PS87] present a layered architecture for meta-data and video segment retrieval using ontologies, and to track and share video segments among cloud nodes.

Noh and Kim [PS73] present a communication bus network architecture for multimedia services in cloud. The solution consists of adaptation middleware and communication bus providing logical and physical end-to-end connections. The proposed model supports profiles of collected information from the user, devices and network to support adaptation.

Table 12: Resource Provisioning and Management – Monitoring and Deployment

Problems		
Challenge	**Description**	**Study Reference**
Monitoring	System monitoring for compliance with dynamic scalability scheme.	[PS54] [PS79]
	Adaptation of monitoring infrastructure with respect to quality requirements.	
	Capturing and monitoring applications' key performance indicators.	
	Non-intrusive monitoring of cloud services.	
Resource Allocation	Selecting suitable service providers for hosting application services.	[PS27]
	Managing stakeholders' conflicts during resource allocation.	
	Quantitatively accessing and evaluating stakeholders and their satisfaction.	
	Autonomic resource allocation and adjustment against risk, trust, reliability and economic efficiency.	
	Explicit architectural constraints for resource allocation, replication, migration and de-allocation.	[PS20]

	Modeling and simulation of cloud resources with respect to different types of QoS parameters.	[PS51]
	Optimized hosting of the resources in cloud data centers.	[PS63]
	Mapping multi-media tasks to Virtual Machines (VMs) and deploy the VMs on physical servers.	[PS95]
Service Selection	Autonomous selection of appropriate services and service providers in order to reduce resource utilization cost.	[PS48] [PS102]
	Collaboration among cloud service providers (service discovery, advertisement and composition).	[PS48]
	Providing scalable service scheduling mechanism that considers dynamic number of service providers and extensible QoS requirements.	[PS107]
	Market-oriented resource and service provisioning.	[PS14]
	Discovery of cloud resources that are compliant with end user needs.	[PS89]
Service Deployment	Handling a large number of service deployment requests in a short period of time.	[PS66]
	Avoid redundant deployment of services in the cloud.	
	Resource deployment according to optimization criteria.	

Challenges: Applications and services deployed on the cloud need to be monitored for their execution according to desired quality parameters. In case an anomaly is detected, additional resources need to be allocated. The papers classified in this subcategory deal with monitoring, selection and deployment of services on the cloud according to specific parameters. The challenges are listed in Table 12.

Solutions: Katsaros et al. [PS54] present a layered architecture [4] to monitor cloud-based applications. The monitoring components are deployed on, PaaS for monitoring the infrastructure and application data, IaaS for monitoring the hardware resources, and SaaS for monitoring applications. Povedano-Molina et al. [PS79] present a Distributed Architecture for Resource manaGement and mOnitoring in cloudS (DARGOS) to determine status and availability of physical resources and services. The presented publisher subscriber paradigm intends to help to accurately measure physical and virtual resources in cloud with the help of monitoring metrics. Ferrer et al. [PS27] propose a holistic approach for cloud service provisioning and single abstraction of multiple coexisting cloud architectures for broader cloud service eco-system. To address the issues of *resource allocation, replication, and migration,*

Chapman et al. [PS20] present a policy including the requirements and constraints that a provider must specify while deploying and hosting a multi component application on cloud. Jararweh et al. [PS51] present CloudExp, a cloud simulation toolkit to simulate different types of quality parameters for testing cloud applications. Li et al. [PS63] present an optimization strategy that is based on decomposing a deployment task into sub-tasks, which are then converted into virtual applications that can be deployed on suitable IaaS cloud resources. Song et al. [PS95] present a queue-based approach for task management, which is based on allocation deadline to optimize the performance of multi-media services. Itani et al. [PS48] present the architecture of a routing decision engine named ServBGP. It is designed by reusing the decision logic of the standard Border Gateway Protocol (BGP). The ServBGP decision engine facilitates the selection of appropriate service provider path based on specific quality attributes and autonomously forwards the customer service request along with appropriate provider paths based on pricing and reputation criteria.

Wu et al. [PS102] report three algorithms for maximizing resource utilization and countering the challenge of selecting appropriate IaaS service providers. The algorithms focus on maximizing the utilization of already initiated Virtual Machines (VMs) and the profit by rescheduling and exploiting penalty delays. A platform presented by Rodriguez-Garcia et al. [PS89] uses semantic technologies to facilitate discovery of cloud resources. The presented framework combines semantic annotation technique, ontology evolution, term extraction and resource indexing to annotate cloud services. Zhao et al. [PS107] propose a service-scheduling algorithm named Service Provider Search Engine (SPSE). The presented algorithm is based on the job request enforcing by QoS requirements (e.g., response time, trust degree, and monetary cost).

Calheiros et al. [PS14] propose solutions to problems of market-oriented resource provisioning, seamless integration of enterprise computing resources and provide a framework named Aneka to support different programming models including thread, task and MapReduce. Aneka framework provides a common root application model, which provides a mechanism for defining common properties of a distributed application. Liu et al. [PS66] discuss the challenges associated with handling large number of service deployment requests in a short period of time, avoiding redundant service deployments in the cloud and ensuring not to omit required services during the optimization process.

2.4.1.2. Pervasive Embedded Networks

Challenges: The papers that have been classified in this subcategory provide solutions for pervasive embedded networks on cloud which are characterized as collection of networked services hosted on tiny and resource constraints devices [PS22][PS23]. The summary of the challenges is presented in Table 13.

Table 13: Pervasive Embedded Networks

Problems		
Challenge	**Description**	**Study Reference**
Service Compliance	Management of mash-up services on shared cloud resources.	[PS22] [PS23]
	Collaboration between heterogeneous devices.	
	Compliance with concrete semantic structures for information presentation and communication.	
	Embedding flowable services in pervasive environments.	[PS109]
Data Compliance	Data management from medical wireless sensor networks.	[PS68]
	Data visualization from various types of data sources (wireless devices, web applications and medical images) in ubiquitous healthcare services.	[PS42]
	Data collection using different types of sensors.	[PS30]
Context Awareness	Determine situational context of data and select services according to the context.	

Solutions: Two studies [PS22] and [PS23] present a Knowledge Aware and Service Oriented (KASO) middleware platform, which carries out common tasks by sharing resources. The middleware platform has been implemented following the Perceptual Reasoning Agents (PRAs) paradigm. Zhu et al. [PS109] present a framework for human centric context aware flowable services. The framework takes advantage of users' context information to support proactive human activities and service integration. Lounis et al. [PS68] present a secured architecture for collecting and processing large volumes of medical sensor data. The security mechanism guarantees confidentiality, integrity and fine-grained access to the data. A multi-layered cloud platform for ubiquitous healthcare services to satisfy high magnitude of concurrent requests have been presented by He et al. [PS42]. The platform

consists of a Cloud Engine that acts as a broker and supports cooperation of the components that are distributed over multiple layers. Forkan et al. [PS30] present a scalable Context Aware Middleware (CAM) framework to facilitate data flow between sensors capturing data and cloud components processing data for facilitating ambient assisted living. The presented middleware handles processing of context data, context-aware service management, security of medical records and mapping between context and services.

2.4.1.3. Cloud Federation

Challenges: Cloud federation is a collection of cloud-enabled resources collaborating with each other [65]. The studies that are classified in this subcategory focus on the challenges associated with cloud federation. Table 14 presents the challenges.

Table 14: Federated Cloud

Problems		
Challenge	**Description**	**Study Reference**
Interoperable Cloud Services	Cloud brokerage for interoperable clouds.	[PS99] [PS100]
	Increase cloud capacity through delegation of tasks on federated clouds.	
	Isolation of services in federated clouds.	
	Inter layer mappings of corresponding layers of reference cloud mode among federated clouds.	
	Interoperability of cloud services.	[PS85] [PS92]
	Maintain decentralize deployment infrastructure that is provided by multiple cloud providers.	[PS76]
	Limited resource in a single cloud provider in stressed data centers.	[PS13]
	Autonomous services composition from multiple clouds.	[PS46]
	Portability of services among clouds.	[PS75]
	Data discovery and selection from heterogeneous cloud services.	[PS104]
Vendor Lock-in	Avoid cloud vendor lock-in.	[PS37]

Solutions: To address the challenges associated with cloud federation, Villegas et al. [PS99] propose to provide inter-cloud federation only at corresponding layers of NIST reference architecture model. This approach

defines different federation methods at each layer to allow elastic and fault-tolerant behavior at different stages without restricting them to a given cloud environment and deployment model. Walraven et al. [PS100] present a middleware platform named PaaSHopper for developing interoperable and multi-tenant cloud services. Rezaei et al. [PS85] present a service-oriented architecture to provide interoperability among the clouds by autonomously converting semantic information to service syntax information and making the services available to consumers. Silva et al. [PS92] provide a platform named Service Delivery Cloud Platform (SDCP) to support common APIs to interact with services of distinct cloud providers using "normalized interfaces". The approach provides services that are secure, redundant and capable of on-the-fly deciphering. The platform focuses on providing interoperability between cloud providers, services delivery using multiple underlying cloud resources and service composition using decoration and orchestration.

The architecture of an agent-based intelligent cloud infrastructure and platform management framework based on Extensible Messaging and Presence Protocol (XMPP) has been presented by Peifeng et al. [PS76]. The aim of the proposed architecture is to simplify server management and increase flexibility and scalability. The framework consists of Management Server Agents for broad level control monitoring and communication, and Host Agents to collect metrics from the cloud platforms. Calheiros et al. [PS13] propose an approach to enable independent cloud data centers to dynamically negotiate resources and to seamlessly meet elastic applications' SLA by scaling applications across various data centers. The proposed approach consists of a negotiation engine that allows Cloud Coordinators to negotiate for selling or buying local or remote resources. Incheon et al. [46] report a staged architecture for discovering services from multiple clouds, selecting best available candidate services and executing the services by composing these in a workflow manner. Paraiso et al. [75] present a component based PaaS named soCloud to support portability and high availability of the services across multiple clouds. Xu et al. [PS104] present an architecture to support video data discovery from heterogeneous cloud services with the help of semantic annotation and semantic searches on the data. A framework to implement platform neutral cloud-based application has been presented by Guillen [PS37]. The framework separates the application code and cloud management and handles the management of cloud resources.

2.4.1.4. Cache Management

Challenges: The papers classified into this subcategory support caching, which is essential for applications requiring high performance and throughput [66]. Table 15 lists the challenges addressed by the research reported in this categories of papers.

Solutions: Two studies [PS39] and [PS77] report middleware-based solutions to address cache management challenges. To tackle the issues of high throughput and managing cache for stateful and transactional applications, an elastic multi-tier architecture has been devised [PS77]. The proposed architecture uses SI-Cache to attain high performance by using snapshot isolation and extends the basic cache functionality by adding support for replication. Han et al. [PS39] present an architecture that focuses on minimizing application's cost while maintaining QoS through a cost-aware and workload-adaptive cache-based scaling approach.

Table 15: Cache Management

Problems		
Challenge	**Description**	**Study Reference**
Increase Throughput	Increase throughput between application layers and provide quick access to the data.	[PS77]
Minimize Execution Cost	Minimize applications execution cost while maintaining QoS.	[PS39]

2.4.1.5. Support for Mobile Devices

Challenges: Cloud computing enables the provisioning of computing intensive services to users using devices with limited computing and storage resources by offloading computing intensive tasks to cloud. The papers classified into this subcategory discuss cloud middleware solutions targeting mobile devices. The identified challenges are shown in Table 16.

Table 16: Support for Mobile Devices

Problems		
Challenge	**Description**	**Study Reference**
Resource Optimization	Satisfying QoS requirements on heterogeneous networks including data synchronization and presentation requirements.	[PS35]
	Off-loading processing tasks from mobile devices on the cloud.	[PS2][PS29] [PS45][PS111]
	Resource optimization and reliability enhancement.	[PS44][PS35] [105]
	Overcome limitation of wireless network	[PS83]

	bandwidth.	
Properties Matching	Providing a match between properties of mobile application and cloud resource.	[PS29]
Data Collection	Data collection, harvesting and analysis via mobile devices.	[PS6] [PS36]

Solutions: A solution to minimize the time for data transfer and task execution on mobile devices has been presented by Gkatzikis et al. [PS35]. The tasks are migrated to clouds for processing as follows: First, the lifetime of a task on current host is calculated; an estimated migration cost and multi-tenancy cost are calculated; migration gain for each potential host is calculated. Finally, a task is migrated to a new host if potential hosts provide migration gain. Amoretti et al. [PS2] present an architecture to provide autonomic offloading of the tasks from mobile devices to the cloud services. The architecture uses KLAIM language semantics for network-automated machines to provide automaticity. A Mobile Cloud Middleware (MCM) framework that supports interoperability, synchronous delegation of mobile tasks and dynamic allocation of cloud infrastructure resources is presented by Flores and Srirama [PS29]. Interoperability between services is provided with the help of Interoperability Engine, which generates adapters to facilitate interoperability. Zixue et al. [PS111] present a three-tier architecture to offload portions of computing tasks from wearable devices to cloud nodes. The offloading strategy maximizes the number of tasks that can be executed on the devices while guaranteeing acceptable response rate to the users. Hu et al. [PS44] present a cyber-physical system (CPS) named as Vita to support mobile users while performing crowd-sensing tasks. The system has been designed using SOA and supports intelligent distribution of tasks. The platform addresses the challenges of low computing and communication overhead on mobile devices. Zhang et al. [105] present an OSGi based pervasive cloud infrastructure for services-offloading to the cloud. The infrastructure uses an elastic open service gateway to migrate services from small mobile devices to powerful cloud nodes.

An architecture named Cooperative Terminals Service Environment (CTSE) for a Cloudlet solution has been presented by Qing et al. [PS83]. CTSE increases bandwidth efficiency for supporting content delivery services on mobile devices by pushing content closer to users to facilitate cooperation among users. Hung et al. [PS45] present a broker-centric architecture for data distribution and collaboration between thick servers and thin mobile clients according to quality of service (QoS) parameters. Benharref and Serhani [PS6] present an architecture for collection and analysis of chronic diseases data that is collected from wearable mobile devices and processes on the cloud. Gronli et al. [PS36] present a three-tier architecture for context aware information harvesting in context of ubiquitous computing. The architecture

consists of android client applications, a server side application and remote Google cloud services and uses meta-tagging of users' activities, social settings and geographical locations.

2.4.1.6. High Performance and Scientific Computing

Challenges: The on-demand resource-provisioning model of Cloud Computing makes it an ideal platform for scientific and high performance computing. The reviewed papers classified in this subcategory propose different strategies to offloading computing intensive tasks to clouds. The challenges are listed in Table 17.

Table 17: High Performance Computing

Problems		
Challenge	**Description**	**Study Reference**
Cost Effective Computing	Providing support for low cost computing cycles to perform complex scientific tasks.	[PS80]
	Offering scientific applications following SaaS model.	[PS101]
Resource Management	Efficient resource management for deadline specific job completion.	[PS94]

Solutions: Prodan et al. [PS80] present a generic master slave framework to implement computing intensive algorithms on Google App Engine (GAE). The framework consists of a master application that manages logic and parallelization algorithm that can split a task into several parallel jobs. The slave applications perform the jobs. The results from slave jobs are collected by the master application and are combined together. The framework presented by Wong and Goscinski [PS101] is used to deploy and expose High Performance Computing (HPC) application on cloud as services. The framework sets up an execution environment by resolving dependencies (library decencies and searchable program paths) and provides a unique interface for constructing application services as SaaS. Somasundaram and Govindarajan [PS94] present CLOUDBR, which consists of a layered architecture and Particle Swarm Optimization (PSO)-based resource allocation and job scheduling mechanism. The framework guarantees deadline specific job completion of scientific applications.

2.4.1.7. Multi-Tenant Environments

Challenges: The primary studies included in this subcategory address challenges regarding security on multi-tenant cloud applications. Table 18 shows the challenges.

Table 18: Security Management in Multi-tenant application

Problems		
Challenge	**Description**	**Study Reference**
Multi-tenancy	Managing authorization and authentication for multi-tenant cloud systems.	[PS7]
	Lowering cost by sharing instances among tenants.	[PS70]

Solutions: Bernabe et al. [PS7] propose a hierarchical RBAC (hRBAC) and conditional RBAC (cRBAC) authorization that is a role based model. It provides better expressiveness to describe advance authorization and federation rules. The system can specify fine-grained definition of the resources available for a particular tenant using its multi-tenancy support and federation capabilities that are defined by means of a trust model, which determines business alliances among cloud tenants. Moens et al. [PS70] present a feature-based cloud resource management model using product line engineering methods to share instances among multiple tenants. The presented cost effective resource allocation model analyses cost of failure to place services on cloud instances and cost of using the instances in terms of price and energy consumption.

2.4.1.8. Data Protection

Challenges: Protecting security and privacy of the data hosted on cloud is a primary quality concern for architects of cloud-based applications. Table 19 enlists the main challenges addressed by the selected papers classified into this subcategory.

Table 19: Data protection

Problems		
Challenge	**Description**	**Study Reference**
Data Security	Ensuring confidentiality, security, integrity and authenticity of the data.	[PS49][PS50] [PS91][PS108]

	Addressing traditional hardware, software and network specific threats.	[PS110]
Secure Data Search	Performing search (or SQL operations) on encrypted data.	[PS28][PS50]

Solutions: Itani et al. [PS49] present a PaaS security framework, SNUGE, for supporting confidentiality, integrity and authenticity of enterprise application's data in PaaS. The framework supports decoupling among security domains to prevent spread of vulnerability from one domain to another. Zissis and Lekkas [PS110] identify the threats in the traditional security implementations and proposes the use of a certified hardened operating system (OS) on a bootable media that is open to extensive audits. The paper proposes a physical security mechanism, which eliminates the threat of installing malicious software on a system and provides elastic cloud architecture to prevent Distributed Denial of Service (DDoS) attacks. Jammalamadaka et al. [PS50] provide a client side interoperable security middleware, named iDataGuard, to facilitate adoption of heterogeneous interfaces of data storage facilities on the Internet. The middleware consists of a security model that provides confidentiality and integrity of outsourced data by indexing encrypted data, and allows searching on the encrypted data. Ferretti et al. [PS28] present an architecture centric solution and a formal model that combines data encryption, key management, and authentication and authorization solutions. The presented formal model guarantees enforcement of confidential access control to meta-data and corresponding data.

Samanthula et al. [PS91] present a Secure Data Sharing (SDS) framework to prevent leakage of data from clouds. The approach specifically addresses cases when an unauthorized or revoked user rejoins a system by using holomorphic encryption and proxy re-encryption scheme [PS91]. The architecture that is presented by Zhou et al. [PS108] proposes a role-based encryption (RBE) approach that integrates cryptography techniques with role-based access control (RBAC) for encrypting data that is stored in public clouds. The approach allows storing data in public cloud while maintaining sensitive information in a private cloud. This scheme allows data encryption in such a way that cloud providers with appropriate roles and privileges can decrypt the data.

2.4.1.9. Enterprise Service Bus (ESB) on Cloud

Challenges: Providing standard end-point abstraction and protecting publicly exposed application interfaces are the challenges to consider for providing cloud-based enterprise service bus. Table 20 lists the challenges that have motivated the solutions reported in the studies classified into this subcategory.

Table 20: Security in Enterprise Service Bus

Problems		
Challenge	**Description**	**Study Reference**
Secure ESB	Protect internal applications when the interfaces are publicly exposed.	[PS90]
	Standards-based endpoint abstraction to provide secure service integration.	

Solutions: The architecture presented by Ryan [PS90] focuses on the security of Enterprise Service Bus (ESB) on the cloud. To tackle the problem of protecting internal application, a solution to include a hardware-based and application-aware appliance to provide protection from attacks (e.g., XML-based, message level and field level attacks) has been proposed. Message privacy, integrity and access control policies are enforced using a consistent configuration driven interface to provide secure integration. A SOA gateway is used extend the traditional ESB role to solve data sensitivity problem.

2.4.1.10. Architectures for Data Intensive Systems

Challenges: The studies that are classified in this category report the challenges related to architecture for service deployment and fault tolerant execution at runtime [PS38]. Table 21 lists the identified challenges.

Table 21: Data Intensive Architecture Challenges

Problems		
Challenge	**Description**	**Study Reference**
Service Deployment	Large data sets processing without huge infrastructure.	[PS19][PS60]
Fault Tolerant Execution	Parallel execution of sufficiently large subsets of data and efficient use of cloud computing environment.	[PS38]

Solutions: Chao et al. [PS19] present a FPGA-based acceleration solution using MapReduce framework. The framework combines hardware-based and software-based acceleration to process data of massive scale and complexity. Kramer et al. [PS60] present a modular and flexible architecture that can encompass multiple big data processing algorithms for processing geospatial data. The data processing tasks are defined as workflows using domain specific language and are processed using cloud resources. To handle data

intensive computations, an architecture based on an extension to MapReduce has been presented by Gunarathne et al. [PS38] by adding a merge step and additional input parameters to Map and Reduce APIs to support the loop variant delta inputs on Windows Azure Platform named Twister4Azure. The Merge function is introduced as a new step to MapReduce for supporting iterative MapReduce computation [PS38]. The framework also supports re-execution of the failed tasks until the completion of the iterative computations.

2.4.2. Workflow Management

The reviewed papers classified in this category focus on architecture-support for the workflow management in cloud-based systems. Based on the extracted data, we categorized the papers into two subcategories: business process management and resource management.

Table 22: Workflow Management

Problems		
Challenge	**Description**	**Study Reference**
Business Process Management	Global access to the applications that are supporting multiple grid types.	[PS26]
	Management of different workflow types (traditional, sustainable and parameter based).	[PS26] [PS52]
	Business processes cooperation for processing sensitive data.	[PS40]
Resource Management	Accessing the dedicated resources in existing cloud environments.	[PS96]
	QoS enforcements mechanisms.	
	Determine the right amount of resource (or resource capacity) for a specific task execution.	[PS11][PS34]
	Provide a framework for rationally fragmenting a workflow model.	[PS58]
	Deploy workflow fragments on the underlying collaborative architectural components.	
	Resource allocation and tasks scheduling for distributed workflows.	[PS8][PS65] [PS71]
	Supporting computing and data integrity in distributed workflows.	[PS53]
	Secure outsourcing of scientific data workflows in the cloud.	[PS67]

2.4.2.1. Business Process Management (BPM)

Challenges: We discuss the reported solutions for global access to the applications supporting multiple grids and managing multiple workflow types. The challenges associated with this category of theme have been described in Table 22 against business process management column.

Solution: Farkas and Kacsuk [PS26] propose a generic solution to support multiple types of workflows including traditional and parametric workflow (run a given workflow according to the number specified in parameters). The presented solution introduces BLACKBOX METAJOB for handling the core workflow execution jobs, executing e-Workflow (executable workflow) instances and running them as normal workflows. The Object Modeling System (OMS) based framework presented by Javadi et al. [PS52] utilizes open source software approach and allows users to design, develop and evaluate loosely coupled service models. The service models (using annotation) are used to support workflow composition and enactment. Han et al. [PS40] propose a distributed processing model named PAD (**P**rocess enactment, **A**ctivity execution and **D**ata storage) for addressing problems related to cooperation on cloud-based BMP platforms (decentralized architecture supporting user-end distribution of business processes) and processing sensitive data.

2.4.2.2. Resource Management

Challenges: The challenges associated with the resource management include Quality of Service (QoS) enforcement, traceability between business requirements and architecture evaluations, and collaboration among fragmented architectures as shown in Table 22.

Solutions: An architecture for enabling staged enforcement of QoS as each stage of a workflow has been presented by Tolosana-Calasanz et al. [PS96]. The proposed solution provides access to dedicated resources in existing cloud environments and supports QoS enforcement mechanisms. The performance rate is used to control transmission of data between two workflow stages in case of network congestion. Byun et al. [PS11] discuss the problem of determining the right amount of resource or resource capacity for a specific task execution. To address this challenge, the authors have presented an algorithm, called Partitioned Balanced Time Scheduling (PBTS), which has the responsibility of identifying executable tasks, estimating resource capacity, scheduling and then executing the tasks on the cloud resources. Ghafarian et al. [PS34] present a workflow execution scheme on hybrid cloud resources. Their architecture can divide a complex workflow into

sub-workflows and deploying workflow nodes on suitable cloud resources. The execution of the sub-workflows is monitored and monitoring values are accounted in the feedback loop to access suitability of the cloud resources for specific workflow tasks. For addressing the issues of providing a framework for rationally fragmenting a workflow and deploying the workflow fragments on the underlying collaborative architectural components, Kim [PS58] proposes a model-driven workflow fragmentation approach. The approach consists of a set of fragmentation algorithms that semantically fragment a workflow model and disseminates its fragments into runtime components of the underlying collaborative workflow system.
Bux and Leser [PS8] present DynamicCloudSim, which is an extension of CloudSim simulation toolkit to support changes in performance and robustness quality parameters at runtime while scheduling scientific workflows. Liu et al. [PS65] present a bioinformatics workflow platform for reliable and highly scalable large scale sequencing analysis. The platform is based on Galaxy workflow system and adds data management capabilities to transfer large quantities of data efficiently and reliably among the processing nodes. Mohamed et al. [PS71] present an approach of dynamically adding autonomic management to cloud workflow resources even though the resources are designed without considering atomicity. A model-driven development environment based on OCCI standard has been proposed to describe resource requirements and elasticity parameters for service-based business processes. Jrad et al. [PS53] present an architecture and ontology model to support functional and non-functional quality of service (QoS) requirements for acquiring computing and storage resources from underlying IaaS cloud, and to execute distributed workflows. Liu et al. [PS67] present a security overhead model for scientific workflow outsourcing that is based on tasks, control flows and datasets to be processed.

2.4.3. Service Level Agreement (SLA) Compliance

This category concentrate on the reviewed work related to QoS-aware service composition, license management in a distributed environment, collaborative QoS and data management. This category is further classified into three subcategories; services and data management, resource discovery and monitoring, and architecture-level support for pricing and billing.

2.4.3.1. Services and Data Management

Challenges: The problems associated with services and data management include service management, service composition, license management in distributed environments, service collaboration and data management. Table 23 shows the architectural challenges discussed.

Table 23: QoS Aware Services and Data Management

Problems		
Challenge	**Description**	**Study Reference**
QoS-Aware Service Composition	Limited availability of qualified candidate services on cloud platforms.	[PS81] [PS72]
	Determine optimal composition of services according to QoS requirements.	
License management in a distributed environment	Lack of support to provide hardware-based licensing in the virtualized infrastructure environments.	[PS12]
Service Collaboration and adaptability	De-coupling and structuring methodologies for SOA dependability.	[PS43]
	Incorporating QoS aspects such as service availability, reliability and security.	
	Service adaptability on heterogeneous cloud environments.	[PS18]
Data Management	Characterizing compliance and regulatory requirements for data retention, migration and confidentiality.	[PS62]
	Lack of fine-grained enforcement policies for managing data during runtime operations.	
	Supporting and enforcing data assurance policies on persistence data objects.	
Consideration for SLA Implementation	SLA compliance in terms of scheduling, security, billing and pricing.	[PS10] [PS15]

Solutions: In order to overcome the identified challenges associated with QoS-aware service composition, Qi et al. [PS81] propose a solution that determines qualified web service composition to satisfy end users' QoS constraints. For each task in the service composition process, the candidate services are searched using the local optimization strategy. In the next step, all the possible web service composition solutions are enumerated to pursue a QoS mechanism near to global optimal. Nae et al. [PS72] present three-tier architecture for massive multiplayer online games to SLA issues using O-SLA mathematical model. Cacciari et al. [PS12] discuss architecture level support for license management in distributed environment and focuses on decoupling authorization for using a license. The licenses are managed through specialized services. To tackle the problems associated with service collaboration, Hiltunen and Schlichting [PS43] propose a Collaborative Quality of Service solution that supports collaboration among the services

through translucent QoS interfaces and Test Collector services (used to report positive or negative service experience with a provider). Castro et al. [PS18] provide a semantic agent-based architecture for SLA management. The shared knowledge plan is used to have a common knowledge base for SLA management agents on heterogeneous cloud environments. To tackle the problems of data management, Li et al. [PS62] propose a policy modeling and enforcement framework. That framework is used for defining data assurance policies along with customer specific requirements and for enforcing them at runtime. Canuto and Guitart [PS15] present a cloud middleware platform EMOTIVE that provides a policy management framework to generate scheduling code on demand according to scheduling requirements of the application. The framework is based on LEPIC language.

2.4.3.2. Resource Discovery and Monitoring

Challenges: These studies report architectural challenges associated with QoS-aware resource discovery, monitoring and management as shown in Table 24.

Table 24: QoS-aware Resource Discovery, Monitoring and Management

Problems		
Challenge	**Description**	**Study Reference**
Resource Management	Decentralization of consistency and scalability management of the services.	[PS21]
	Resource management according to workload conditions and user behaviors.	[PS57] [PS59] [PS16]
	Achieving QoS on shared hardware.	[PS61]
	Minimizing cost while maintaining SLAs.	[PS64]
Service Discovery	SLA specific cloud services discovery.	[PS88] [PS33]
SLA Monitoring, Anticipation and Compliance	SLA compliance for heterogeneous devices.	[PS106]
	SLA monitoring based on data from multiple data sources using dynamic metrics.	[PS93]
	Cloud system behavior anticipation according to specific QoS requirements.	[PS32]
	Monitoring cloud services based on customizable monitoring parameters.	[PS1]

Solutions: Chen et al. [PS21] provide a solution to satisfy consistency requirements of the application when cloud services are replicated. The proposed architecture introduces the notion of consistency regions and

proposes a Region-Based Election Protocol (REP) to elastically balance workload among the regions. The architecture that is described by Kertesz et al. [PS57] focuses on service virtualization following SLA and supports interoperable service execution in a diverse and distributed virtualized service system. MAPE (Monitor, Analyze, Plan and Execute) pattern is used for processing SLA-based Service Virtualization (SSV) architecture. A behavioral-based resource management approach that can manage resources across multiple cloud layers (i.e., infrastructure, platform, and software), minimize cost of satisfying QoS requirements and optimize infrastructure capacity has been presented by Kousiouris et al. [PS59]. The proposed approach analyzes information related to the application terms of use and utilizes this information to estimate low-level resource attributes using time series analysis method and Artificial Neural Networks. Casalicchio and Silvestri [PS16] analyze the problem from the perspective of an Application Service Provider (ASP) and uses a cloud infrastructure to achieve scalable provisioning of its services with respect to QoS constraints using self-adaptation.

Krebs et al. [PS61] present metrics to quantify performance isolation in cloud-based systems. The proposed metrics are based on QoS impact and workload ratios. The study [PS64] proposes a customer-driven SLA-based resource provisioning architecture. The architecture uses customers' profiling and service providers' quality parameters to handle dynamic customer requests. The architecture that is described by Rodriguez-Garcia et al. [PS88] focuses on SLA-specific cloud services discovery and facilitates services discovery by creating a service repository using semantic discovery of cloud services. Similarity between the query and available services is calculated with the help of a cosine function of the semantic queries and services' QoS vectors. Garg et al. [PS33] propose a framework to create healthy competition among Cloud providers and to satisfy SLAs. The framework (SMICloud) consists of a set of business-relevant key performance indicators (KPIs) and provides a standardized method for measuring and comparing business services.

Zhang and Zhou [PS106] present a service-centric computing environment to implement socio temporal extension to von-Neuman architecture. The presented approach exploits the idea that programs are stored and executed on different virtualized computing resources and can be accessed by users from different devices. Smit et al. [PS93] present an architecture to offer Monitoring data as a Service (Monitoring-as-a-Service). The architecture is based on a stream-processing framework, which intends to work on streaming data instead of stored data. The monitoring infrastructure is built following publisher-subscriber pattern, where the publisher acquires and converts the metrics from the original sources and the subscribers consume the metrics. Garcia et al. [PS32] present a platform (Cloudcompaas) for assessment of the

cloud resources. The platform facilitates static resource deployment scheduling based on a resource definition model. The monitoring feature of the platform specifies SLA terms and checks the service execution with respect to the SLA. When the SLA is violated, the monitor performs corrective actions to restore proper functioning of the service. Calero et al. [PS1] present an adaptive distributed monitoring PaaS architecture named MonPaaS, for monitoring cloud providers based on consumers-specific SLA metrics. The architecture monitors both virtual and physical resources.

2.4.3.3. Architecture Support for Pricing and Billing

Challenges: Billing of resources according to SLAs is a primary concern for cloud providers.

Solutions: For solving the issue related to SLA implementation in terms of billing, pricing and security, Buyya et al. [PS10] proposes an architecture-level support for fine-grained pricing and billing mechanisms. The solution is based on the concept of cloud market where users can request for the resources according to the needs. The framework also supports workload balancing and provisioning of resources from public cloud environments.

2.4.4. Energy Awareness

The reviewed papers included in energy awareness category provide solutions for resource optimization and business process management according to energy efficiency requirement. This category is further classified into two subcategories as described in the following sections. Table 25 lists the reported challenges.

Table 25: Energy Awareness

Problems		
Challenge	**Description**	**Study Reference**
Resource Optimization	Optimizing profit margin.	[PS5]
	Reducing energy footprint.	
	Algorithms satisfying quality of service (QoS) requirements and SLAs.	
	Energy efficient scheduling and routing according to QoS parameters.	[PS9]
	Energy efficiency solutions by powering off idle nodes.	[PS24]
	Virtual Machines placement according to optimum energy consumption patterns.	[PS55]
	Handling tradeoff between power and	[PS25]

	performance in SaaS cloud platforms.	
Energy Efficient Business Process Management	Cost effective and ecologically friendly business process.	[PS74]
	Business process management with respect to changes in execution environment.	

2.4.4.1. Resource Optimization

Challenges: Resource optimization challenges deal with optimizing profit margins, reducing energy footprint, and satisfying energy related QoS.

Solutions: Beloglazov et al. [PS5] propose an algorithmic solution, called Modified Best Fit Decreasing (MBFD). The algorithm validates whether the energy utilization of a host is more than acceptable threshold. If so, the Virtual Machines (VMs) are migrated from the host machine. Buysse et al. [PS9] present a heuristic-based routing and scheduling algorithms to minimize total energy consumption by switching off unused resources. The heuristic model is based on finding an "IT endpoint" to process the request and finding an optimal route (corresponding to Network and IT Infrastructure Utilization) from the source to the endpoint in a network. Alfonso et al. [PS24] present an energy management system for HPC clusters and cloud infrastructures. The system focuses on integration with existing middleware and implementation of different energy saving policies. The approach connects Local Resource Management System (LRMS) with an energy saving system that manages power and treat LRMS as black box

Katsaros et al. [PS55] present a service framework to optimize energy consumption. The presented framework monitors the energy consumption of a cloud platform and analyzes energy consumption for effectively managing VMs with the help of external sensor devices. Fangming et al. [PS25] discuss a unified profit maximizing objective method (Lyapunov optimization method) that takes into consideration both revenue and cost. The presented approach uses buffering to alleviate resource surge and improves robustness of a system by associating power budget with each resource. Raycroft et al. [PS84] present an analysis of energy consumption of real-world VM allocations policies. The most prominent energy consumption policies are Watts per Core policy and Stripping and Load Balancing policy.

2.4.4.2. Energy Efficient Process

Challenges: The papers classified into this category deals with the challenge of cost effectiveness plus eco-friendly business process management according to energy efficiency requirements as shown in Table 25.

Solutions: Nowak et al. [PS74] present a pattern-driven adaptation methodology that annotates the green business process to the available application components. In the next step, a green business process pattern is selected that best fits with general strategic objective of an organization and provides solution of ecological critical part of the process. The pattern formats are described in terms of how resources are offered by a cloud platform and which services are accessing the cloud resources.

2.5. Discussion on Commercial Cloud Solutions and Research Outcomes

Commercial and Open Source Cloud Computing solution providers (such as Amazon cloud services [21], Microsoft Azure [36], Google App Engine [35], Eucalyptus [33] and OpenNebula [34]) are focused on providing IaaS, PaaS and SaaS based offerings. In this section, we briefly discuss how the research outcomes of our SLR are related to the commercial cloud solutions. Software Architecture (SA) targeting cloud-based environments deals with different levels of abstractions based on commonly known services models, i.e., IaaS, PaaS and SaaS. We limit the scope of the comparison by using the well-known Cloud Services, i.e., Amazon, Google App Engine and Azure. Most prominent offerings from Amazon cloud services and Microsoft Azure are IaaS and PaaS cloud resources. Amazon also provides additional services to manage IaaS cloud resources. For example, Amazon's auto scaling [67] feature in combination with elastic load balancer [68] and cloud watch [69] enable replicated allocation and de-allocation of VM instances based on predefined rules on VMs' CPU cycles and incoming requests. This type of solution can provide a certain level of scalability for stateless applications, but is far from autonomous scalability requirements of modern day applications that are not always stateless. Amazon Simple Workflow Service is another prominent service and can be referred as PaaS [70]. It facilitates using amazon infrastructure for computing intensive workflows. Microsoft Azure [36] offers a broad spectrum of services including virtual machines (VM), BizTalk application integration services, and SaaS applications such as Visual Studio online and SharePoint portal. Azure services also provide support for integrating Azure infrastructure with organizations' existing infrastructure and services.

Google App Engine [35] and Salesforce [22] are the examples of PaaS cloud and provide APIs that can be used to write applications. The PaaS model abstracts the management of underlying infrastructure. Google App Engine provides a generic solution by offering application developers to write applications using Python, Java, PHP or Google Go languages. It also provides support for NoSQL and relational databases. Applications that are deployed on Google App Engine platform run in a sandbox and underlying infrastructure is managed by the platform [71, 72]. Everything associated with application's deployment and scalability are hidden from users. It is guaranteed that a server response is generated within 30 seconds against every request to the applications and services that are hosted on the platform, although average response time is less than a second. While Google App Engine provides developers with ease and convenience for using the platform by abstracting all the information related to infrastructure management, application architects and developers lose control on QoS and SLA compliance parameters. Salesforce provides a more specialized PaaS solution by providing APIs for office automation and Customer Relationship Management (CRM) solutions, and supports multiple programming languages including Java, Rails, Scala, Python, Node and Clojure [22].

Whilst commercial cloud providers are offering services for application development and deployment, researchers and some industrial initiatives have focused on providing solutions that facilitate the optimal utilization of resources provided by IaaS and PaaS cloud services according to specific requirements of the applications and their respective domains. An additional interesting observation is that certain industrial initiatives focus on providing reference architecture models while academic research is focusing on solutions to small and fine grained problems associated with software architectures of cloud-based systems that can be implemented and evaluated in academic settings. The reviewed papers specifically focus on the following aspect of architectures for Cloud Computing:

- Identifying trusted services from the available services that comply with quality of service (QoS) requirements of the application and their end users.
- Utilizing cloud services according to the legal and regulatory restrictions.
- Utilizing cloud offerings to facilitate availability of the applications that are in compliance with end-user service level agreements (SLAs).
- Guaranteeing execution of the applications on cloud infrastructure with respect to QoS (e.g. scalability and reliability) requirements.
- Supporting portability of applications among IaaS cloud to take benefits of cloud bursting on hybrid cloud environments.
- Provisioning resource from underlying IaaS clouds in cost and energy efficient manner.

- Discovering optimal cloud resources that best match the desired requirements.
- Providing mechanisms for resource monitoring.
- Supporting application and services on federated cloud environment.
- Providing support for multiple tenants, and securing tenant specific data and services.

There are also some initiatives and solutions that have been reported but not included in our review because we have only selected journal paper for this SLR. The ***WSO2*** carbon platform [73] implements multi-tenancy features in cloud middleware. This is a componentized framework that is used to configure cloud application servers. It consists of components that can fit into servers' runtime environment and offers web service APIs for service management. It also supports provisioning of bundled components on tenant-specific instances. ***IBM Altocumulus*** middleware framework [74] enables interoperability across computing clouds. It is useful when organizations use a combination of private and public clouds. A middleware named ***YML*** [75] can be used for managing dedicated and non-dedicated computing resources, hosting multiple types of operating systems, writing customized algorithms, utilizing other middleware by specifying interfaces for each of them and supporting user interaction with services through YML frontend.

Nix Package Manager [76] supports building packages from resources using a functional language. This supports multiple versions of the packages in a system with atomic upgrade and roll back features. ***Disnix*** [76] supports distributed deployments of packages using model-based descriptions. ***MeDICi*** framework [77] supports the processing of data-intensive applications on distributed nodes. It facilitates the creation of a pipeline-based data processing system. It is used for performing analysis on huge volumes of data by deploying nodes of the processing framework on physical or virtual nodes. ***FraSCAti*** [78] is used for runtime management and monitoring of components' or services' properties dealing with their association with other services. It can also be used for activating and deactivating components and services. ***Choco*** [78], a Java based library, is used to define constraints and associated domain functions as a model. It is also used for optimization problems on cloud. ***IBM Tivoli Provisioning Manager (TPM)*** along with ***Apache HTTP load balancer*** is used to automate manual tasks of configuring and provisioning cloud resources [79].

2.6. Threats to Validity

The quality of this SLR was ensured by developing and reviewing research protocol following the guidelines of conducting SLRs reported in [39]. The research protocol helped us to minimize the potential bias in the papers

selection process. The research protocol contained research questions, search strategies, inclusion and exclusion criteria, data extraction form, and literature synthesis approach to be used in our review. The research protocol was developed by the first author and was reviewed by the second author and an external expert. We ensured that the search strings were appropriately derived from research questions.

Accuracy and consistency during the review process are usually based on a common understanding among the authors; misunderstandings can result in biased effects. One of the main limitations of the review could be the possibility of bias in selecting the papers to be reviewed. To help ensure that the selection process was as unbiased as possible, we developed detailed guidelines in the *review protocol* prior to the start of the review. During the papers screening phase (i.e., study selection and filtration), we documented the reasons for including or excluding each paper. Finally, we rechecked the selected papers again based on the inclusion and exclusion criteria.

The review process was designed to address the threats to conclusion, internal and construct validity. We used data extraction form to address the threats to ***conclusion validity*** [80]. The use of a well-defined data extraction form is expected to reduce bias in data extraction process and ensures data extraction that is consistent and relevant to research questions. An uneven number of studies from publication sources could also be a threat to conclusion validity and was addressed by systematic study selection and review process. Bias in papers selection process could be a threat to ***internal validity*** [80]. This was addressed by our multi-stage search and papers selection process as described in Section 2. The first author selected an initial set of included papers. The second author and a research assistant reviewed the initial set of selected papers and a disputed set of papers were included or excluded after discussions among the authors and the research assistant. Validity of data selection and its representation to address research questions is referred as ***Construct Validity*** [80]. The searches were performed on multiple databases to get relevant journal papers. The research protocol was developed in order to minimize the potential threat to construct validity. The research question, inclusion and exclusion criteria, search strings used for searches and data extraction strategy ensured consistent data extraction process and valid results.

2.7. Conclusions

Cloud Computing is being widely adopted with an increasing number of applications being deployed on Cloud-based platforms. Designing and evaluating software-intensive systems and applications for Cloud platforms are enormously complex and challenging undertakings. SAs of Cloud-based

systems should be designed for fully exploiting Cloud Computing features. Given the importance of identifying and addressing the challenges involved in designing appropriate architectures, a large amount of literature has been published in a relatively short time period. However, there has been no attempt to systematically identify, rigorously assess and synthesize the reported challenges of and associated solutions to build an evidence-based body of knowledge on architecting for cloud computing.

We conducted a SLR of the peer-reviewed literature reporting the challenges of and the solutions for architecting cloud-based software systems. We decided to review only papers published in Journals as an indicator of their high quality and completeness of work that is usually published as a journal paper. Based on systematic analysis of 111 journal papers, we have identified 44 challenges of architecting cloud-enabled software intensive systems and classified them in different categories including *Resource and Service Management*, *Workflow Management*, *Service Level Agreement Compliance* and *Energy Awareness*.

The results of our review show that the reported solutions have used a variety of cloud environments for implementation and evaluation of the proposed approaches. Amazon, Windows Azure and Google App Engine are the most commonly used public cloud environments. A majority of the reviewed papers used architecture centric strategies to address the challenges of architecting cloud-based software systems; however, there are a significant number of studies that are proposing algorithmic solutions to complement architecture based solutions. For example, the algorithms to support scalability to minimize execution cost are described in [PS39]. The reviewed papers show an even distribution with respect to maturity stages as most of the papers have evaluated the proposed solutions in real or simulated cloud environments demonstrating that the solutions can be applied in real life settings. The architectural solutions for cloud-based systems proposed in the reviewed papers suggest that scalability, performance, response time and optimum utilization of the cloud resources are frequently researched quality characteristics.

The findings are expected to provide useful information and insights to both researchers and practitioners. For researchers, the review provides important information about different types of quality attributes that are more frequently reported with respect to architecting cloud-based software systems. We have systematically drawn a taxonomy that can be used to analyze and categorize architectural challenges of and solutions to cloud-based systems. Moreover, this review has also identified some emerging areas of research for architecting cloud-based solutions (e.g., green cloud computing, cloud for mobile and ubiquitous computing, and Internet of Things (IoTs)) and to fill

the gap in the research areas that are still not mature (e.g., enhancing SLA specification and management capabilities to cover overall life cycle of cloud-enabled applications including but not limited to security, privacy, pricing and quality concerns). The practitioners can use the findings as a source of information to identify relevant approaches, and to learn and access relevant solutions that can be tailored to a specific application's requirements. The details of our methodological approach can also provide useful insights into the processes and procedures for building evidence-based body of knowledge about a given topic. Researchers can also use the methodological details for updating this literature review.

Listing A

Table 26: Data Extraction Form

Elements of Data Extraction Form	
Element	**Description**
Study Identity	Unique identity for the study.
Reviewer	Reviewer's name.
Review Date	The date of data extraction.
Bibliographic References	Author, year of publication, title and source of publication.
Focus of the Study or Study Category	Main topic area or the problem study is trying to address.
Objective	Description of study objectives.
Motivation	Explanation of motivation behind study.
Challenges	Issues and problems being addresses in the study
Solutions	Solutions presented in the study to solve stated problems.
Algorithms	Algorithms proposed/used to solve stated problem in combination with architecture elements.
Quality Attributes Focused	Primary quality attributes affected by the solutions.
Cloud Environment	Cloud environments on which solutions are implemented and tested.
Cloud Service Model	Cloud service model on which solutions are applicable.
Maturity Stage	Specification of maturity stage of the presented research. It has one of the following values: basic research, concept formulation, development and extension, internal enhancement and exploration, and popularization.
Evaluation Settings	Environment in which the proposed solutions are evaluated. It is one of the following: Simulation Environment, Evaluated on Local Infrastructure.
Limitations and Future Research Directions	Limitations and areas of improvements in the reported solutions.

Listing B

Table 27: Detailed Quality Assessment Score of Selected Primary Studies

Quality Score													
References	**Research (Q1)**	**Aim (Q2)**	**Context (Q3)**	**Findings (Q4)**	**Enhancement (Q5)**	**Aggregated Score**	**References**	**Research (Q1)**	**Aim (Q2)**	**Context (Q3)**	**Findings (Q4)**	**Enhancement (Q5)**	**Aggregated Score**
[PS41]	1	1	1	1	1	5	[PS42]	1	1	1	1	0	4

[PS31]	1	1	1	1	1	5	[PS69]	1	1	1	1	0	4
[PS47]	1	1	1	1	1	5	[PS30]	1	1	1	1	0	4
[PS78]	1	1	1	1	1	5	[PS99]	1	1	0	1	1	4
[PS54]	1	1	1	1	1	5	[PS92]	1	1	1	1	0	4
[PS20]	1	1	1	1	1	5	[PS37]	1	1	1	1	0	4
[PS14]	1	1	1	1	1	5	[PS77]	1	1	1	1	0	4
[PS13]	1	1	1	1	1	5	[PS44]	1	1	1	1	0	4
[PS66]	1	1	1	1	1	5	[PS35]	1	1	1	1	1	4
[PS79]	1	1	1	1	1	5	[PS83]	1	1	1	1	0	4
[PS22]	1	1	1	1	1	5	[PS80]	1	1	1	1	0	4
[PS23]	1	1	1	1	1	5	[PS49]	1	1	1	1	0	4
[PS39]	1	1	1	1	1	5	[PS91]	1	1	1	1	0	4
[PS29]	1	1	1	1	1	5	[PS108]	1	1	1	1	0	4
[PS101]	1	1	1	1	1	5	[PS12]	1	1	1	1	0	4
[PS7]	1	1	1	1	1	5	[PS57]	1	1	1	1	0	4
[PS50]	1	1	1	1	1	5	[PS16]	1	1	1	1	0	4
[PS40]	1	1	1	1	1	5	[PS93]	1	1	1	1	0	4
[PS52]	1	1	1	1	1	5	[PS32]	1	1	1	1	0	4
[PS96]	1	1	1	1	1	5	[PS74]	1	1	1	0	1	4
[PS11]	1	1	1	1	1	5	[PS9]	1	1	1	1	0	4
[PS81]	1	1	1	1	1	5	[PS18]	1	1	1	1	0	4
[PS107]	1	1	1	1	1	5	[PS28]	1	1	1	1	0	4
[PS102]	1	1	1	1	1	5	[PS51]	1	1	1	1	0	4
[PS21]	1	1	1	1	1	5	[PS82]	1	1	1	1	0	4
[PS59]	1	1	1	1	1	5	[PS86]	1	1	1	1	0	4
[PS61]	1	1	1	1	1	5	[PS87]	1	1	1	0.5	1	4
[PS88]	1	1	1	1	1	5	[PS95]	1	1	1	1	0	4
[PS106]	1	1	1	1	1	5	[PS97]	1	1	1	1	0	4
[PS33]	1	1	1	1	1	5	[PS103]	1	1	0.5	1	1	4
[PS5]	1	1	1	1	1	5	[PS34]	1	1	0.5	1	1	4
[PS24]	1	1	1	1	1	5	[PS65]	1	1	0.5	1	1	4
[PS55]	1	1	1	1	1	5	[PS67]	1	1	0.5	1	1	4
[PS84]	1	1	1	1	1	5	[PS72]	1	1	0.5	1	1	4
[PS38]	1	1	1	1	1	5	[PS89]	1	1	0	0.5	1	3.5
[PS3]	1	1	1	1	1	5	[PS58]	1	1	0.5	0	1	3.5
[PS6]	1	1	1	1	1	5	[PS62]	1	1	0.5	0	1	3.5
[PS15]	1	1	1	1	1	5	[PS4]	1	1	0	0	1	3
[PS56]	1	1	1	1	1	5	[PS10]	1	1	0	0	1	3
[PS63]	1	1	1	1	1	5	[PS110]	1	1	0	0	1	3
[PS70]	1	1	1	1	1	5	[PS2]	1	1	0	0	1	3
[PS75]	1	1	1	1	1	5	[PS36]	1	1	0.5	1	0.5	3
[PS94]	1	1	1	1	1	5	[PS45]	1	1	0	1	0	3
[PS98]	1	1	1	1	1	5	[PS46]	1	1	0	1	0	3

[PS1]	1	1	1	1	1	5	[PS105]	1	1	0.5	1	0	3
[PS8]	1	1	1	1	1	5	[PS68]	1	1	0.5	1	0.5	3
[PS17]	1	1	1	1	1	5	[PS111]	1	1	0.5	1	0	3
[PS19]	1	1	1	1	1	5	[PS26]	1	1	0.5	0	0	2.5
[PS71]	1	1	1	1	1	5	[PS76]	1	1	0	0	0	2
[PS109]	1	1	1	1	1	5	[PS90]	1	0	0	0	0	2
[PS53]	1	1	1	1	1	5	[PS43]	1	1	0	0	0	2
[PS64]	1	1	1	1	1	5	[PS85]	1	1	0	0	0	2
[PS100]	1	1	1	1	1	5	[PS104]	1	1	0	0	0	2
[PS73]	1	1	1	1	1	4	[PS60]	1	1	0	0	0	2
[PS27]	1	1	1	1	0	4	[PS25]	1	1	0.5	0	0	2
[PS48]	1	1	1	1	0	4							

	Research (Q1)	Aim (Q2)	Context (Q3)	Findings (Q4)	Enhancement (Q5)	Aggregated Score
Average	1	0.99	0.86	0.86	0.64	4.27

Listing C

Following is the list of the selected studies that are included in this review.

[PS1] J. M. Alcaraz Calero and J. Gutierrez Aguado, "MonPaaS: An Adaptive Monitoring Platformas a Service for Cloud Computing Infrastructures and Services," *Services Computing, IEEE Transactions on,* vol. 8, pp. 65-78, 2015.

[PS2] M. Amoretti, A. Grazioli, V. Senni, F. Tiezzi, and F. Zanichelli, "A formalized framework for mobile cloud computing," *Service Oriented Computing and Applications,* pp. 1-20, 2014.

[PS3] K. An, S. Shekhar, F. Caglar, A. Gokhale, and S. Sastry, "A cloud middleware for assuring performance and high availability of soft real-time applications," *Journal of Systems Architecture,* vol. 60, pp. 757-769, 2014.

[PS4] P. Belimpasakis and S. Moloney, "A platform for proving family oriented RESTful services hosted at home," *Consumer Electronics, IEEE Transactions on,* vol. 55, pp. 690-698, 2009.

[PS5] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future generation computer systems,* vol. 28, pp. 755-768, 2012.

[PS6] A. Benharref and M. A. Serhani, "Novel Cloud and SOA-Based Framework for E-Health Monitoring Using Wireless Biosensors," *Biomedical and Health Informatics, IEEE Journal of,* vol. 18, pp. 46-55, 2014.

[PS7] J. Bernal Bernabe, J. M. Marin Perez, J. M. Alcaraz Calero, F. J. Garcia Clemente, G. Martinez Perez, and A. F. Gomez Skarmeta, "Semantic-aware multi-tenancy

authorization system for cloud architectures," *Future Generation Computer Systems,* vol. 32, pp. 154-167, 2014.

[PS8] M. Bux and U. Leser, "DynamicCloudSim: Simulating heterogeneity in computational clouds," *Future Generation Computer Systems,* vol. 46, pp. 85-99, 2015.

[PS9] J. Buysse, K. Georgakilas, A. Tzanakaki, M. De Leenheer, B. Dhoedt, and C. Develder, "Energy-efficient resource-provisioning algorithms for optical clouds," *Journal of Optical Communications and Networking,* vol. 5, pp. 226-239, 2013.

[PS10] R. Buyya, S. Pandey, and C. Vecchiola, "Cloudbus toolkit for market-oriented cloud computing," in *Cloud Computing*, ed: Springer, 2009, pp. 24-44.

[PS11] E.-K. Byun, Y.-S. Kee, J.-S. Kim, and S. Maeng, "Cost optimized provisioning of elastic resources for application workflows," *Future Generation Computer Systems,* vol. 27, pp. 1011-1026, 2011.

[PS12] C. Cacciari, D. Mallmann, C. Zsigri, F. D'Andria, B. Hagemeier, A. Rumpl*, et al.*, "SLA-based management of software licenses as web service resources in distributed computing infrastructures," *Future Generation Computer Systems,* vol. 28, pp. 1340-1349, 2012.

[PS13] R. N. Calheiros, A. N. Toosi, C. Vecchiola, and R. Buyya, "A coordinator for scaling elastic applications across multiple clouds," *Future Generation Computer Systems,* vol. 28, pp. 1350-1362, 2012.

[PS14] R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, and R. Buyya, "The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid Clouds," *Future Generation Computer Systems,* vol. 28, pp. 861-870, 2012.

[PS15] M. Canuto and J. Guitart, "Integrated policy management framework for IaaS Cloud middleware," *Computing,* pp. 1-24, 2014.

[PS16] E. Casalicchio and L. Silvestri, "Mechanisms for SLA provisioning in cloud-based service providers," *Computer Networks,* vol. 57, pp. 795-810, 2013.

[PS17] A. Castiglione, R. Pizzolante, A. De Santis, B. Carpentieri, A. Castiglione, and F. Palmieri, "Cloud-based adaptive compression and secure management services for 3D healthcare data," *Future Generation Computer Systems,* vol. 43–44, pp. 120-134, 2015.

[PS18] A. Castro, V. A. Villagra, B. Fuentes, and B. Costales, "A Flexible Architecture for Service Management in the Cloud," *Network and Service Management, IEEE Transactions on,* vol. 11, pp. 116-125, 2014.

[PS19] W. Chao, L. Xi, C. Peng, W. Aili, Z. Xuehai, and Y. Hong, "Heterogeneous Cloud Framework for Big Data Genome Sequencing," *Computational Biology and Bioinformatics, IEEE/ACM Transactions on,* vol. 12, pp. 166-178, 2015.

[PS20] C. Chapman, W. Emmerich, F. Márquez, S. Clayman, and A. Galis, "Software architecture definition for on-demand cloud provisioning," *Cluster Computing,* pp. 1-22, 2011.

[PS21] T. Chen, R. Bahsoon, and A.-R. H. Tawil, "Scalable service-oriented replication with flexible consistency guarantee in the cloud," *Information Sciences,* vol. 264, pp. 349-370, 2013.

[PS22] I. Corredor, J. F. Martínez, and M. S. Familiar, "Bringing pervasive embedded networks to the service cloud: A lightweight middleware approach," *Journal of systems architecture,* vol. 57, pp. 916-933, 2011.

[PS23] I. Corredor, J. F. Martínez, M. S. Familiar, and L. López, "Knowledge-aware and service-oriented middleware for deploying pervasive services," *Journal of Network and Computer Applications,* vol. 35, pp. 562-576, 2012.

[PS24] C. De Alfonso, M. Caballer, F. Alvarruiz, and V. Hernández, "An energy management system for cluster infrastructures," *Computers & Electrical Engineering,* vol. 39, pp. 2579-2590, 2013.
[PS25] L. Fangming, Z. Zhi, J. Hai, L. Bo, L. Baochun, and J. Hongbo, "On Arbitrating the Power-Performance Tradeoff in SaaS Clouds," *Parallel and Distributed Systems, IEEE Transactions on,* vol. 25, pp. 2648-2658, 2014.
[PS26] Z. Farkas and P. Kacsuk, "P-GRADE portal: a generic workflow system to support user communities," *Future Generation Computer Systems,* vol. 27, pp. 454-465, 2011.
[PS27] A. J. Ferrer, F. HernáNdez, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri*, et al.*, "OPTIMIS: A holistic approach to cloud service provisioning," *Future Generation Computer Systems,* vol. 28, pp. 66-77, 2012.
[PS28] L. Ferretti, F. Pierazzi, M. Colajanni, and M. Marchetti, "Scalable Architecture for Multi-User Encrypted SQL Operations on Cloud Database Services," *Cloud Computing, IEEE Transactions on,* vol. 2, pp. 448-458, 2014.
[PS29] H. Flores and S. N. Srirama, "Mobile cloud middleware," *Journal of Systems and Software,* vol. 92, pp. 82-94, 2013.
[PS30] A. Forkan, I. Khalil, and Z. Tari, "CoCaMAAL: A cloud-oriented context-aware middleware in ambient assisted living," *Future Generation Computer Systems,* vol. 35, pp. 114-127, 2014.
[PS31] M. E. Frîncu, "Scheduling highly available applications on cloud environments," *Future Generation Computer Systems,* vol. 32, pp. 138-153, 2014.
[PS32] A. G. García, I. B. Espert, and V. H. García, "SLA-driven dynamic cloud resource management," *Future Generation Computer Systems,* vol. 31, pp. 1-11, 2014.
[PS33] S. K. Garg, S. Versteeg, and R. Buyya, "A framework for ranking of cloud computing services," *Future Generation Computer Systems,* vol. 29, pp. 1012-1023, 2013.
[PS34] T. Ghafarian and B. Javadi, "Cloud-aware data intensive workflow scheduling on volunteer computing systems," *Future Generation Computer Systems,* 2014.
[PS35] L. Gkatzikis and I. Koutsopoulos, "Migrate or not? exploiting dynamic task migration in mobile cloud computing systems," *Wireless Communications, IEEE,* vol. 20, pp. 24-32, 2013.
[PS36] T.-M. GrØnli, G. Ghinea, and M. Younas, "Context-aware and Automatic Configuration of Mobile Devices in Cloud-enabled Ubiquitous Computing," *Personal Ubiquitous Comput.,* vol. 18, pp. 883-894, 2014.
[PS37] J. Guillén, J. Miranda, J. M. Murillo, and C. Canal, "A service-oriented framework for developing cross cloud migratable software," *Journal of Systems and Software,* vol. 86, pp. 2294-2308, 2013.
[PS38] T. Gunarathne, B. Zhang, T.-L. Wu, and J. Qiu, "Scalable parallel computing on clouds using Twister4Azure iterative MapReduce," *Future Generation Computer Systems,* vol. 29, pp. 1035-1048, 2013.
[PS39] R. Han, M. M. Ghanem, L. Guo, Y. Guo, and M. Osmond, "Enabling cost-aware and adaptive elasticity of multi-tier cloud applications," *Future Generation Computer Systems,* vol. 32, pp. 82-98, 2014.
[PS40] Y.-B. Han, J.-Y. Sun, G.-L. Wang, and H.-F. Li, "A cloud-based bpm architecture with user-end distribution of non-compute-intensive activities and sensitive data," *Journal of Computer Science and Technology,* vol. 25, pp. 1157-1167, 2010.
[PS41] M. M. Hassan, B. Song, and E.-N. Huh, "A market-oriented dynamic collaborative cloud services platform," *annals of telecommunications-annales des télécommunications,* vol. 65, pp. 669-688, 2010.

[PS42] C. He, X. Fan, and Y. Li, "Toward ubiquitous healthcare services with a novel efficient cloud platform," *Biomedical Engineering, IEEE Transactions on,* vol. 60, pp. 230-234, 2013.
[PS43] M. A. Hiltunen and R. D. Schlichting, "Is collaborative QoS the solution to the SOA dependability dilemma?," in *Architecting dependable systems VII*, ed: Springer, 2010, pp. 227-248.
[PS44] X. Hu, T. H. Chu, H. C. Chan, and V. C. Leung, "Vita: A crowdsensing-oriented mobile cyber-physical system," *Emerging Topics in Computing, IEEE Transactions on,* vol. 1, pp. 148-165, 2013.
[PS45] P. P. Hung, T.-A. Bui, M. A. Morales, M. Nguyen, and E.-N. Huh, "Optimal Collaboration of Thin—thick Clients and Resource Allocation in Cloud Computing," *Personal Ubiquitous Comput.,* vol. 18, pp. 563-572, 2014.
[PS46] P. Incheon, C. Wuhui, and M. N. Huhns, "A Scalable Architecture for Automatic Service Composition," *Services Computing, IEEE Transactions on,* vol. 7, pp. 82-95, 2014.
[PS47] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek, "Adaptive resource provisioning for read intensive multi-tier applications in the cloud," *Future Generation Computer Systems,* vol. 27, pp. 871-879, 2011.
[PS48] W. Itani, C. Ghali, R. Bassil, A. Kayssi, and A. Chehab, "ServBGP: BGP-inspired autonomic service routing for multi-provider collaborative architectures in the cloud," *Future Generation Computer Systems,* vol. 32, pp. 99-117, 2014.
[PS49] W. Itani, A. Kayssi, and A. Chehab, "SNUAGE: an efficient platform-as-a-service security framework for the cloud," *Cluster computing,* vol. 16, pp. 707-724, 2013.
[PS50] R. C. Jammalamadaka, R. Gamboni, S. Mehrotra, K. Seamons, and N. Venkatasubramanian, "A middleware approach for outsourcing data securely," *computers & security,* vol. 32, pp. 252-266, 2013.
[PS51] Y. Jararweh, M. Jarrah, M. kharbutli, Z. Alshara, M. N. Alsaleh, and M. Al-Ayyoub, "CloudExp: A comprehensive cloud computing experimental framework," *Simulation Modelling Practice and Theory,* vol. 49, pp. 180-192, 2014.
[PS52] B. Javadi, M. Tomko, and R. O. Sinnott, "Decentralized orchestration of data-centric workflows in Cloud environments," *Future Generation Computer Systems,* vol. 29, pp. 1826-1837, 2013.
[PS53] F. Jrad, J. Tao, I. Brandic, and A. Streit, "SLA enactment for large-scale healthcare workflows on multi-Cloud," *Future Generation Computer Systems,* vol. 43–44, pp. 135-148, 2015.
[PS54] G. Katsaros, G. Kousiouris, S. V. Gogouvitis, D. Kyriazis, A. Menychtas, and T. Varvarigou, "A Self-adaptive hierarchical monitoring mechanism for Clouds," *Journal of Systems and Software,* vol. 85, pp. 1029-1041, 2012.
[PS55] G. Katsaros, J. Subirats, J. O. Fitó, J. Guitart, P. Gilet, and D. Espling, "A service framework for energy-aware monitoring and VM management in Clouds," *Future Generation Computer Systems,* vol. 29, pp. 2077-2091, 2013.
[PS56] P. D. Kaur and I. Chana, "A resource elasticity framework for QoS-aware execution of cloud applications," *Future Generation Computer Systems,* vol. 37, pp. 14-25, 2014.
[PS57] A. Kertész, G. Kecskemeti, and I. Brandic, "An interoperable and self-adaptive approach for SLA-based service virtualization in heterogeneous Cloud environments," *Future Generation Computer Systems,* vol. 32, pp. 54-68, 2014.
[PS58] K. Kim, "A model-driven workflow fragmentation framework for collaborative workflow architectures and systems," *Journal of network and computer applications,* vol. 35, pp. 97-110, 2012.

[PS59] G. Kousiouris, A. Menychtas, D. Kyriazis, S. Gogouvitis, and T. Varvarigou, "Dynamic, behavioral-based estimation of resource provisioning based on high-level application terms in Cloud platforms," *Future Generation Computer Systems,* vol. 32, pp. 27-40, 2012.

[PS60] M. Krämer and I. Senner, "A modular software architecture for processing of big geospatial data in the cloud," *Computers & Graphics,* 2015.

[PS61] R. Krebs, C. Momm, and S. Kounev, "Metrics and techniques for quantifying performance isolation in cloud environments," *Science of Computer Programming,* vol. 90, pp. 116-134, 2013.

[PS62] J. Li, B. Stephenson, H. R. Motahari-Nezhad, and S. Singhal, "GEODAC: A data assurance policy specification and enforcement framework for outsourced services," *Services Computing, IEEE Transactions on,* vol. 4, pp. 340-354, 2011.

[PS63] Z. Li, H. Li, X. Wang, and K. Li, "A generic cloud platform for engineering optimization based on OpenStack," *Advances in Engineering Software,* vol. 75, pp. 42-57, 2014.

[PS64] W. Linlin, S. K. Garg, S. Versteeg, and R. Buyya, "SLA-Based Resource Provisioning for Hosted Software-as-a-Service Applications in Cloud Computing Environments," *Services Computing, IEEE Transactions on,* vol. 7, pp. 465-485, 2014.

[PS65] B. Liu, R. K. Madduri, B. Sotomayor, K. Chard, L. Lacinski, U. J. Dave*, et al.*, "Cloud-based bioinformatics workflow platform for large-scale next-generation sequencing analyses," *Journal of Biomedical Informatics,* vol. 49, pp. 119-133, 2014.

[PS66] T. Liu, T. Lu, W. Wang, Q. Wang, Z. Liu, N. Gu*, et al.*, "SDMS-O: A service deployment management system for optimization in clouds while guaranteeing users' QoS requirements," *Future Generation Computer Systems,* vol. 28, pp. 1100-1109, 2012.

[PS67] W. Liu, S. Peng, W. Du, W. Wang, and G. S. Zeng, "Security-aware intermediate data placement strategy in scientific cloud workflows," *Knowledge and Information Systems,* vol. 41, pp. 423-447, 2014.

[PS68] A. Lounis, A. Hadjidj, A. Bouabdallah, and Y. Challal, "Healing on the cloud: Secure cloud architecture for medical wireless sensor networks," *Future Generation Computer Systems,* 2015.

[PS69] J. L. Lucas-Simarro, R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, "Scheduling strategies for optimal service deployment across multiple clouds," *Future Generation Computer Systems,* vol. 29, pp. 1431-1441, 2013.

[PS70] H. Moens, E. Truyen, S. Walraven, W. Joosen, B. Dhoedt, and F. D. Turck, "Cost-Effective Feature Placement of Customizable Multi-Tenant Applications in the Cloud," *Journal of Network and Systems Management,* vol. 22, pp. 517-558, 2014.

[PS71] M. Mohamed, M. Amziani, D. Belaïd, S. Tata, and T. Melliti, "An autonomic approach to manage elasticity of business processes in the Cloud," *Future Generation Computer Systems,* 2014.

[PS72] V. Nae, R. Prodan, and A. Iosup, "SLA-based operations of massively multiplayer online games in clouds," *Multimedia Systems,* vol. 20, pp. 521-544, 2014.

[PS73] W. Noh and T. Kim, "Flexible communication-bus architecture for distributed multimedia service in cloud computing platform," *Consumer Electronics, IEEE Transactions on,* vol. 59, pp. 530-537, 2013.

[PS74] A. Nowak, T. Binz, C. Fehling, O. Kopp, F. Leymann, and S. Wagner, "Pattern-driven green adaptation of process-based applications and their runtime infrastructure," *Computing,* vol. 94, pp. 463-487, 2012.

[PS75] F. Paraiso, P. Merle, and L. Seinturier, "soCloud: a service-oriented component-based PaaS for managing portability, provisioning, elasticity, and high availability across multiple clouds," *Computing,* pp. 1-27, 2014.

[PS76] S. Peifeng, S. Chuan, and Z. Xiang, "Intelligent server management framework over extensible messaging and presence protocol," *Communications, China,* vol. 10, pp. 128-136, 2013.

[PS77] F. Perez-Sorrosal, M. Patiño-Martinez, R. Jimenez-Peris, and B. Kemme, "Elastic SI-Cache: consistent and scalable caching in multi-tier architectures," *The VLDB Journal—The International Journal on Very Large Data Bases,* vol. 20, pp. 841-865, 2011.

[PS78] D. Petcu, G. Macariu, S. Panica, and C. Crăciun, "Portable cloud applications—from theory to practice," *Future Generation Computer Systems,* vol. 29, pp. 1417-1430, 2013.

[PS79] J. Povedano-Molina, J. M. Lopez-Vega, J. M. Lopez-Soler, A. Corradi, and L. Foschini, "DARGOS: A highly adaptable and scalable monitoring architecture for multi-tenant Clouds," *Future Generation Computer Systems,* vol. 29, pp. 2041-2056, 2013.

[PS80] R. Prodan and M. Sperk, "Scientific computing with Google App Engine," *Future Generation Computer Systems,* vol. 29, pp. 1851-1859, 2013.

[PS81] L. Qi, W. Dou, X. Zhang, and J. Chen, "A QoS-aware composition method supporting cross-platform service invocation in cloud environment," *Journal of Computer and System Sciences,* vol. 78, pp. 1316-1329, 2012.

[PS82] L. Qin, W. Guojun, and W. Jie, "Consistency as a Service: Auditing Cloud Consistency," *Network and Service Management, IEEE Transactions on,* vol. 11, pp. 25-35, 2014.

[PS83] W. Qing, H. Zheng, W. Ming, and L. Haifeng, "CACTSE: Cloudlet aided cooperative terminals service environment for mobile proximity content delivery," *Communications, China,* vol. 10, pp. 47-59, 2013.

[PS84] P. Raycroft, R. Jansen, M. Jarus, and P. R. Brenner, "Performance bounded energy efficient virtual machine allocation in the global cloud," *Sustainable Computing: Informatics and Systems,* vol. 4, pp. 1-9, 2013.

[PS85] R. Rezaei, T. K. Chiew, S. P. Lee, and Z. Shams Aliee, "A semantic interoperability framework for software as a service systems in cloud computing environments," *Expert Systems with Applications,* vol. 41, pp. 5751-5770, 2014.

[PS86] L. S. Ribeiro, C. Viana-Ferreira, J. L. Oliveira, and C. Costa, "XDS-I Outsourcing Proxy: Ensuring Confidentiality While Preserving Interoperability," *Biomedical and Health Informatics, IEEE Journal of,* vol. 18, pp. 1404-1412, 2014.

[PS87] V. Rocha, F. Kon, R. Cobe, and R. Wassermann, "A hybrid cloud-P2P architecture for multimedia information retrieval on VoD services," *Computing,* pp. 1-20, 2014.

[PS88] M. Á. Rodríguez-García, R. Valencia-García, F. García-Sánchez, and J. J. Samper-Zapater, "Ontology-based annotation and retrieval of services in the cloud," *Knowledge-Based Systems,* vol. 56, pp. 15-25, 2013.

[PS89] M. Á. Rodríguez-García, R. Valencia-García, F. García-Sánchez, and J. J. Samper-Zapater, "Creating a semantically-enhanced cloud services environment through ontology evolution," *Future Generation Computer Systems,* vol. 32, pp. 295-306, 2014.

[PS90] J. Ryan, "Rethinking the ESB: building a secure bus with an SOA gateway," *Network Security,* vol. 2012, pp. 14-17, 2012.

[PS91] B. K. Samanthula, Y. Elmehdwi, G. Howser, and S. Madria, "A secure data sharing and query processing framework via federation of cloud computing," *Information Systems,* vol. 48, pp. 196-212, 2013.

[PS92] L. A. B. Silva, C. Costa, and J. L. Oliveira, "A common API for delivering services over multi-vendor cloud resources," *Journal of Systems and Software,* vol. 86, pp. 2309-2317, 2013.

[PS93] M. Smit, B. Simmons, and M. Litoiu, "Distributed, application-level monitoring for heterogeneous clouds using stream processing," *Future Generation Computer Systems,* vol. 29, pp. 2103-2114, 2013.

[PS94] T. S. Somasundaram and K. Govindarajan, "CLOUDRB: A framework for scheduling and managing High-Performance Computing (HPC) applications in science cloud," *Future Generation Computer Systems,* vol. 34, pp. 47-65, 2014.

[PS95] B. Song, M. M. Hassan, A. Alamri, A. Alelaiwi, Y. Tian, M. Pathan*, et al.*, "A two-stage approach for task and resource management in multimedia cloud environment," *Computing,* pp. 1-27, 2014.

[PS96] R. Tolosana-Calasanz, J. Á. BañAres, C. Pham, and O. F. Rana, "Enforcing QoS in scientific workflow systems enacted over Cloud infrastructures," *Journal of Computer and System Sciences,* vol. 78, pp. 1300-1315, 2012.

[PS97] V. Varadharajan and U. Tupakula, "Security as a Service Model for Cloud Environment," *Network and Service Management, IEEE Transactions on,* vol. 11, pp. 60-75, 2014.

[PS98] J. Vera-del-Campo, J. Pegueroles, J. Hernández-Serrano, and M. Soriano, "DocCloud: A document recommender system on cloud computing with plausible deniability," *Information Sciences,* vol. 258, pp. 387-402, 2014.

[PS99] D. Villegas, N. Bobroff, I. Rodero, J. Delgado, Y. Liu, A. Devarakonda*, et al.*, "Cloud federation in a layered service model," *Journal of Computer and System Sciences,* vol. 78, pp. 1330-1344, 2012.

[PS100] S. Walraven, D. V. Landuyt, A. Rafique, B. Lagaisse, and W. Joosen, "PaaSHopper: Policy-driven middleware for multi-PaaS environments," *Journal of Internet Services and Applications,* vol. 6, pp. 1-14, 2015.

[PS101] A. K. Wong and A. M. Goscinski, "A unified framework for the deployment, exposure and access of HPC applications as services in clouds," *Future Generation Computer Systems,* vol. 29, pp. 1333-1344, 2013.

[PS102] L. Wu, S. K. Garg, and R. Buyya, "SLA-based admission control for a Software-as-a-Service provider in Cloud computing environments," *Journal of Computer and System Sciences,* vol. 78, pp. 1280-1299, 2012.

[PS103] X. Wu, M. Liu, W. Dou, L. Gao, and S. Yu, "A scalable and automatic mechanism for resource allocation in self-organizing cloud," *Peer-to-Peer Networking and Applications,* pp. 1-14, 2014.

[PS104] Z. Xu, L. Mei, Y. Liu, C. Hu, and L. Chen, "Semantic enhanced cloud environment for surveillance data management using video structural description," *Computing,* pp. 1-20, 2014.

[PS105] W. Zhang, L. Chen, X. Liu, Q. Lu, P. Zhang, and S. Yang, "An OSGi-based flexible and adaptive pervasive cloud infrastructure," *Science China Information Sciences,* vol. 57, pp. 1-11, 2014.

[PS106] Y. Zhang and Y. Zhou, "Transparent computing: spatiotemporal extension on von Neumann architecture for cloud services," *Tsinghua Science and Technology,* vol. 18, pp. 10-21, 2013.

[PS107] L. Zhao, Y. Ren, M. Li, and K. Sakurai, "Flexible service selection with user-specific QoS support in service-oriented architecture," *Journal of Network and Computer Applications,* vol. 35, pp. 962-973, 2012.

[PS108] L. Zhou, V. Varadharajan, and M. Hitchens, "Achieving secure role-based access control on encrypted data in cloud storage," *Information Forensics and Security, IEEE Transactions on,* vol. 8, pp. 1947-1960, 2013.

[PS109] Y. Zhu, R. Y. Shtykh, and Q. Jin, "A human-centric framework for context-aware flowable services in cloud computing environments," *Information Sciences,* vol. 257, pp. 231-247, 2014.
[PS110] D. Zissis and D. Lekkas, "Securing e-Government and e-Voting with an open cloud computing architecture," *Government Information Quarterly,* vol. 28, pp. 239-251, 2011.
[PS111] C. Zixue, L. Peng, W. Junbo, and G. Song, "Just-in-Time Code Offloading for Wearable Computing," *Emerging Topics in Computing, IEEE Transactions on,* vol. 3, pp. 74-83, 2015.

Chapter 3. Reference Architecture Development Process Framework for TSPACE

For the implementation of Tools as a service workSPACE (TSPACE), a thorough understanding of TSPACE needs to be established. In this chapter, we define a conceptual framework that guides TSPACE reference architecture development. The framework describes the concepts that are important to design TSPACE reference architecture and to support multiple lifecycle phases for on demand provisioning of Tools as a Service (TaaS). The framework provides a foundation for TSPACE requirements and reference architecture elements (to be discussed in detail in Chapter 4, Chapter 5 and Chapter 6). The framework serves as a guiding tool for addressing the challenges that emerge as a result of the highly dynamic nature of design time and runtime functional requirements as well as architecture quality requirements of TSPACE. The framework consists of multiple stages corresponding to the reference architecture requirements identification, reference architecture documentation approach, TSPACE meta-models design, and detailed architecture design and evaluation approach.

3.1. Introduction

The framework described in this chapter provides a general overview of TSPACE and the design process that leads to the concrete architecture design of TSPACE reference architecture. The framework contains concepts that describe TSPACE environment as well as the relations between the concepts. The identified concepts need to be addressed during TSPACE architecture analysis, design and development. The framework provides process guidelines and serves as a convenient tool for the structured analysis of TSPACE requirements and domain. The concepts that are identified and described in the framework are not only used to design the TSPACE reference architecture but also to provide a valuable insight on implementation of the reference architecture. For convenience, we have named Reference Architecture Development process Framework as RADeF.

As discussed in the introductory chapter, TSPACE aims at providing stakeholders involved in software architecture design and software engineering activities easy access to the tools that are required to perform the activities. RADeF provides a conceptual design framework for TSPACE and serves as a guiding tool for the detail specification of the activities that are to be performed not only during design and analysis of TSPACE reference architecture, but also during its implementation and execution phases. The

framework also discusses key elements of TSPACE and explains their relationships.

The trend of offering Software as a Service (SaaS) has significantly increased over the last few years and has been adopted in various domains [81-85]. Introduction of an *aaS model for software engineering tools, requires tailoring of the design strategies that are used for SaaS. One of the primary reason for this is that the software engineering tools often serve as part of tools ecosystem or a suite of tools, where multiple tools can be provided by different vendors and can be developed using different technologies [86]. To perform software engineering activities in general and software architecting activities in particular, four to five tools are used on average [55]. The tools can provide support for specific activities and can provide a certain number of features. The way in which artifacts are managed by the tools also varies. Some tools store the artifacts in the proprietary data structures, while others use standardized formats such as Unified Modeling Language (UML), which is supported by many architecture-modeling tools. Moreover, the tools are often developed using different technological paradigms such as desktop-based tools and web-based tools. Providing integration among different type of tools is challenging [87]. A paradigm that aims to offer the tools as services has to address the fundamental issue of providing the tools as part of the tool suite and supporting integration among the heterogeneous tools. Figure 8 presents a high-level overview of TSPACE context, in which tools are offered as *aaS model using cloud infrastructure and can be made accessible to end-users. In RADef, we consider technological and business aspects of all the above-mentioned issues. In the research that is presented in this dissertation, we have focused on the tools that are used in software architecting domain, which is a subdomain of software engineering. Although the findings from the research can be applied on other domains (disciplines), as per the focus of the dissertation we are only making the discussion with reference to software architecting domain. The concepts and the process that is described in this chapter are used for the detailed analysis of the requirements and detailed design of TSPACE reference architecture (which is further discussed in forthcoming chapters).

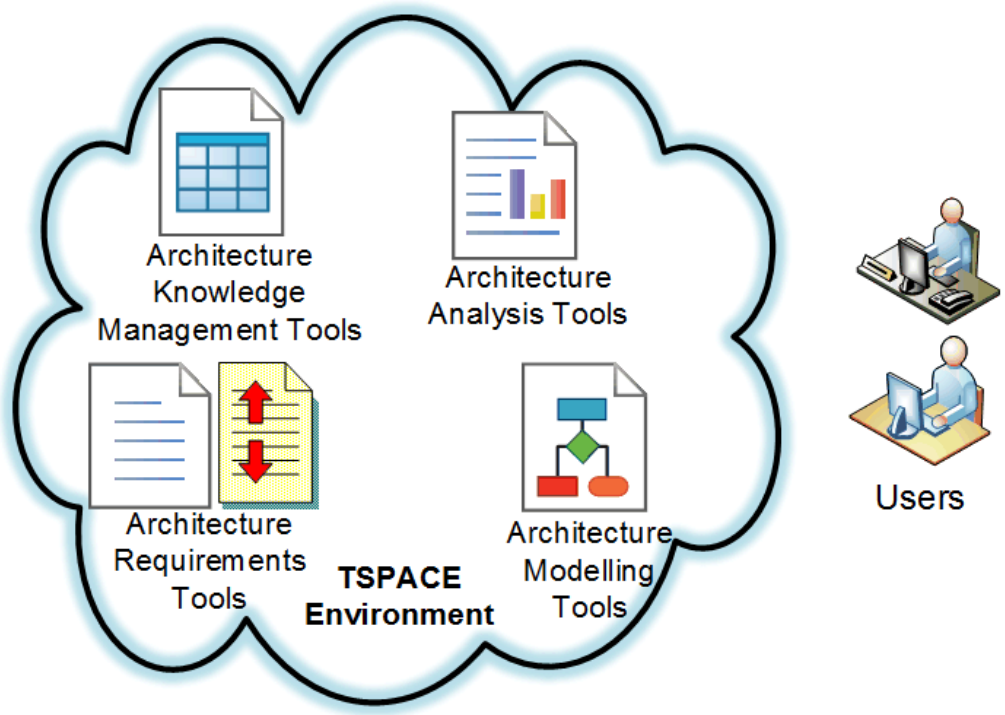


Figure 8: TSPACE Context

In the remainder of this chapter, we provide a high-level view of the framework. First, we define groups of concepts related to TSPACE, establish and elaborate the relationships among concepts and constituting elements of the concepts and discuss the concepts and constituting elements with respect to TSPACE domain. In second step, we elaborate the process that is used to define the reference architecture for TSPACE. The process framework that is discussed in this chapter is used to define TSPACE requirements (Chapter 4), formalization of TSPACE reference architecture using meta-models and ontologies [48] to address the semantic integration needs (Chapter 5) and detailed design of TSPACE reference architecture and its evaluation (Chapter 6).

3.2. High-level Overview of TSPACE Elements and their Relationships

The provisioning of the TaaS is at the center of TSPACE framework. As described in Chapter 1, TSPACE is defined as "*A platform that facilitates offering of a bundled suite of the desired tool to tenants as part of a workspace on demand*" [38]. This definition leads to the establishment of six groups of concepts. The involvement of bundled suite of tools leads to an integrated suite and the need for the tools to be provisioned in a way that is compliant with the specific needs of the tenants. The presence of multiple tenants shows that TSPACE should be configurable according to the specific needs of the users who are going to use the tools provisioned in a TSPACE

instance. The need for the tools to be a part of the workspace signifies that the tools provisioning and bundling should be according to the specific activities and tasks of the respective domain in which the tools are to be used. Table 28 describes high-level concepts encompassing TSPACE.

Table 28: High-level TSPACE Concepts

Concept	Description
Tool	Tools that are used to perform the activities in TSPACE.
Tenant	Users corresponding to a tenant for which a TSPACE instance is provisioned.
Context	Domains in which the tools as part of TSPACE are to be used.
Provisioning	Provisioning environment/infrastructure that is used to host and provision TSPACE.
Artifacts	Artifacts that can be produced or consumed by the tools in a TSPACE instance.
Integration	Tools integration needs that are necessary to bundle the tools together in a suite.

Figure 9 shows the relations among multiple concepts constituting TSPACE. All six elements of TSPACE, i.e. Tool, Tenant, Artifact, Provisioning, Context and Integration, are contained by a TSPACE via an aggregation relation. The users that can use the tools in a TSPACE instance are grouped into tenants. The users can work on multiple artifacts. The context of the domain for which TSPACE is designed, determines the integration needs of the tools that can eventually support their bundling as a tool suite. The tools integration approach needs to focus on two different aspects. First, it focuses on integration support for the artifacts that can exist as a stand-alone entity (are in standardized formats and can exist outside the scope of the tool) by facilitating exchange of artifacts among the tools. Second, it focuses on providing support for the artifacts that cannot exist as a stand-alone entity (are in tools proprietary format and cannot exist outside the scope of the tool). The tools are provisioned on the underlying infrastructure as per the selections of the tools in a TSPACE instance and are bundled together using the integration approach. TSPACE design approach should consider all the elements of TSPACE and their specific needs.

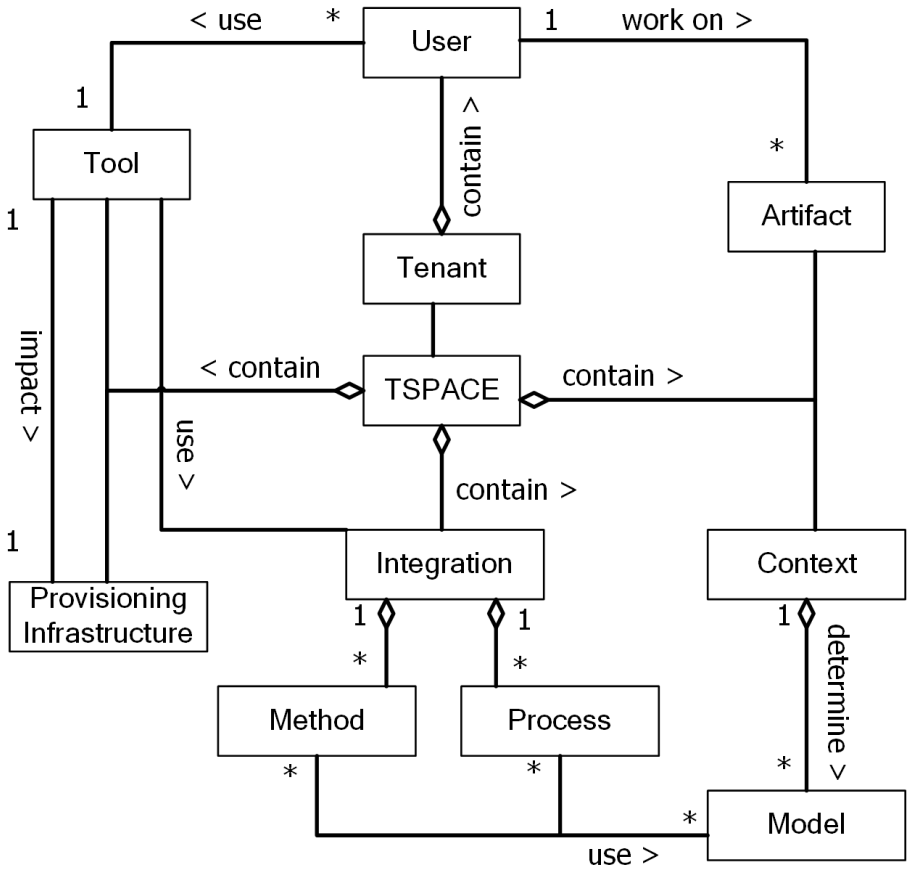


Figure 9: TSPACE Concepts Relations

Figure 10 shows details of the Tool element meta-model. A tool suite can consist of multiple tools or components that together enable the tool suite to perform certain features. In TSPACE, the tools and components can belong to different layers of the tools, e.g. presentation layer, business layer or data persistence layer. The components (and tools) can act as data sources for the components of the same tools or for the components of the other tools. Likewise the tools and components can also be the consumers of the data. Hence, each artifacts or data that is produced in a TSPACE instance have either a producer or a consumer associated with it. This configuration enables TSPACE to bundle the tools and components in a suite. Every tool and component has its deployment specifications that determine the runtime environment of the components and their deployment configurations. The tools and their constituting components can be implemented using object-oriented paradigms or service-oriented architecture (SOA) based technologies. The tools can belong to different types of technological paradigms such as desktop-based tools, web-based tools or cloud-based tools.

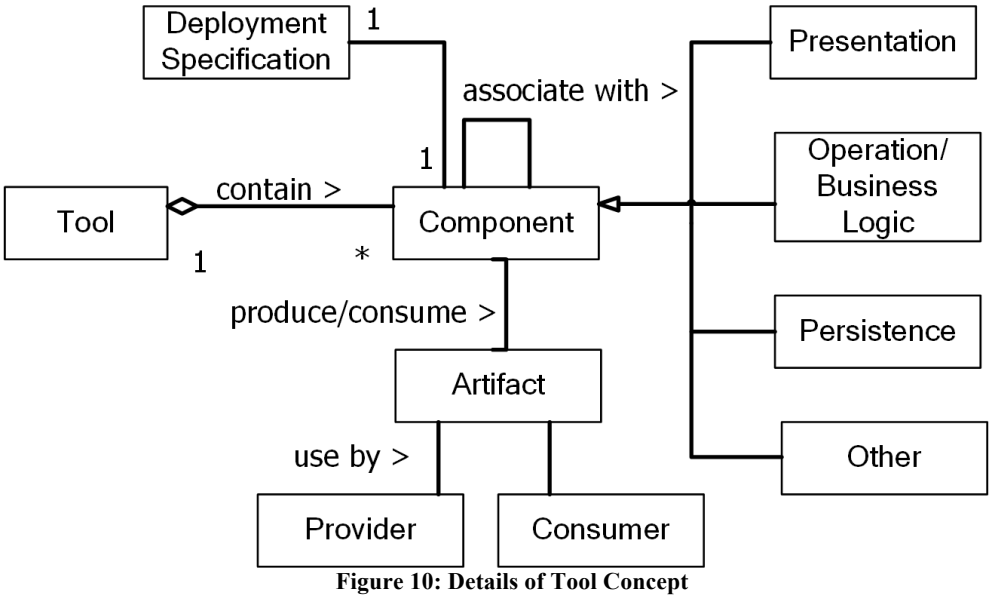


Figure 10: Details of Tool Concept

3.3. TSPACE Reference Architecture Design Process

A software reference architecture provides valuable guidelines for designing a concrete architecture. As a reference architecture for each domain has some unique characteristics and designing the reference architecture for the domain requires specific considerations, it is important to describe a reference architecture as comprehensively as possible and in an easy-to-understand way. It is also important to have a clearly described process that can be used to design and evaluate the reference architecture for a specific domain [15]. In this section, we describe the process that leads to TSPACE reference architecture development, evaluation and implementation. We also discuss important factors that should be considered at each stage of TSPACE reference architecture design. A pictorial representation of TSPACE reference architecture design process is presented in Figure 11. Information produced in the preceding stages is used as input for the proceeding stages of the process.

3.3.1. TSPACE Reference Architecture Design Process Stages

3.3.1.1. Stage 1 - Identification of TSPACE Concepts and Elements

First step in designing the reference architecture is to identify different concepts and elements that constitute TSPACE. At this stage, a high-level analysis of the domain in which the reference architecture is to be used is required. In Section 3.2, we have discussed basic concepts and elements of TSPACE for software architecting domain and the relationships between

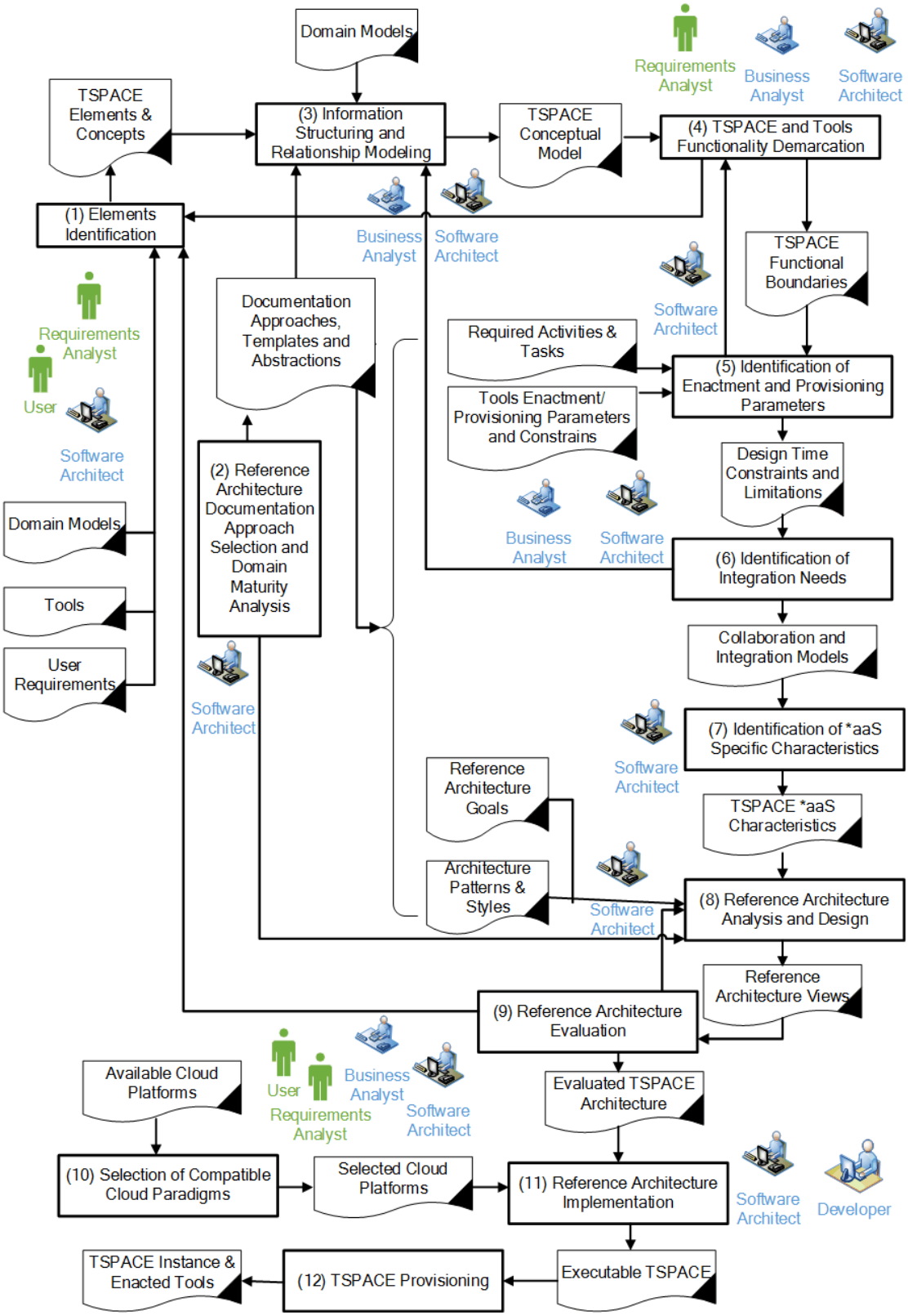


Figure 11: TSPACE Reference Architecture Design Process

them. As described earlier; *Tenants*, *Tools*, *Provisioning Infrastructure*, *Artifacts*, *Context* and *Integration Methods* are primary elements for software architecting TSPACE. Each of the primary elements is further explored in subsequent stages.

Participants' Roles: End users, Requirements Analysts and Software Architects.
Artifact(s) Consumed: N/A
Artifact(s) Produced: High level meta-models for TSPACE concepts and elements.

3.3.1.2. Stage 2 - Reference Architecture Documentation Approach

The activities that are performed at this stage focus on analyzing architecture documentation approaches and preliminary analysis of the maturity of the domain for which the reference architecture is being designed. The analysis of the documentation approaches determines the most appropriate strategies for capturing the architecture of the domain for which TSPACE is being designed. A comprehensive analysis of the software reference architecture documentation approaches is presented in [15, 16]. Angelov et al. have described that the reference architecture documentation should cover context, goals and design dimensions. The context dimension covers the purpose, the organization(s) who is (are) developing the reference architecture and maturity stage (e.g. preliminary or classic) of a reference architecture [15]. The goal dimension encompasses business goals and quality attributes as well as the purpose of defining a reference architecture (e.g. to standardize concrete architecture or to facilitate design of concrete architecture). The design dimension elaborates whether the reference architecture is concrete or abstract and whether the reference architecture has been described using formal, semiformal or informal approaches. Avgeriou et al. [16] propose that a reference architecture description should address three main constituents: (a) description of the approach used to document a reference architecture, (b) guidelines on instantiation of a reference architecture and (c) evaluation of a reference architecture corresponding to desired functional requirements and quality attributes. The outcome of this activity determines the reference architecture description approach, the level of abstractions to be covered in reference architecture documentation, the goals of the reference architecture in terms of its objectives and selection of approaches for evaluation and instantiation of the reference architecture. Outcome of this activity has impact on all the proceeding stages of the reference architecture design process. Reference architecture design dimensions are listed in Table 29.

Table 29: Software Reference Architecture Design Dimensions

Dimension	Sub-dimension
Description	How is the reference architecture documented?
Context	Who defines the reference architecture?
	Where will the reference architecture be used?
	What is the maturity stage of the domain?
Goal	Why is the reference architecture defined?
Design	What is described in the reference architecture design?
	How is the design described?
	How is the design represented?
Evaluation	How is the reference architecture evaluated?
Instantiation	How is the reference architecture instantiated?

Participants' Roles: Software Architect.
Artifact(s) Consumed: Architecture documentation templates.
Artifact(s) Produced: Reference architecture documentation approaches, templates and abstractions at which the architecture is to be documented, architecture evaluation and architecture initialization/instantiation approaches.

3.3.1.3. Stage 3 - Concepts and Elements Refinements, Structuring and Relationship Modeling

The activities that are performed at this stage aim at refining the concepts and elements that are identified in stage one, establishing the hierarchical structure of TSPACE elements and the relationships among the elements. The core sources of information for this stage are domain models. As we are focusing on software architecting TSPACE, architecture description standards such as IEEE 1471-2000 [47] and ISO/IEC/IEEE 42010:2011 [7] are used for refinements, structuring and relationship modeling of the concepts. These models provide a high-level standardized conceptual model of software and system architecture. IEEE 1471-2000 standard considers the mission that a system needs to fulfill and the environment that influence system's operations, whereas ISO/IEC/IEEE 42010:2011 only considers a system of interest, its environment and its architecture.

The domain models can provide standardizations for elements, their hierarchical structures and the relationships among the elements, however these models needs to be extended in order to cover all the dimensions of TSPACE including the tools, the processes which governs the provisioning and usage of the tools, data integration and exchange formats among the tools and additional functional aspects that are required by TSPACE in a specific domain. To incorporate ISO/IEC/IEEE 42010:2011 and IEEE 1471-2000 for software architecting TSPACE, the models need to be extended to incorporate

software architecting tools, processes, integration needs and workspace specific elements. The details on the extension process and extended models are discussed in Chapter 5 and Chapter 6. The artifacts that are produced at this stage, serve as a foundation for detailed requirements analysis and architecture design of the components that are responsible for tools bundling and integration of TSPACE.

Participants' Roles: Business Analyst and Software Architect.
Artifact(s) Consumed: Documentation approaches, documentation templates and architecture design abstractions.
Artifact(s) Produced: TSPACE conceptual models that consists of concepts and elements that encompass TSPACE and relationship among the concepts and models.

3.3.1.4. Stage 4 - Reference Architecture Functional Demarcation

The activities that are performed at this stage deal with demarcation of functional requirements that are to be taken care by TSPACE and functional requirements for which TSPACE can rely on the tools (that can be provisioned by TSPACE). For example, for software architecting TSPACE, the different activities associated with software architecting, such as architecture significant requirements documentation, architecture tradeoff analysis and architecture modeling can be supported by the provisioned tools. Whereas provisioning of the tools, providing integration among the tools so that the tools can exchange artifacts and data, and workspace-specific requirements such as collaboration and awareness support need to be supported by TSPACE. The artifacts that are produced at this stage provide a foundation for TSPACE functional requirements and high-level architecture design with specific focus on the identification of components responsible for TSPACE features.

Participants' Roles: Requirements Analyst, Business Analyst and Software Architect.
Artifact(s) Consumed: Domain models.
Artifact(s) Produced: Documents describing functional demarcation of TSPACE and encompassing tools.

3.3.1.5. Stage 5 - Identification of Provisioning and Enactment parameters

At this stage, the focus is on identifying tools bundling, provisioning and enactment needs and constraints. As one of the primary objectives for providing software architecting TSPACE is to provide the bundles suite of

tools on demand as part of the workspace, it is critical to determine bundling and provisioning constraints and parameters. The tools that are used to perform software architecting activities are of heterogeneous nature. The tools ranging from simple text-based tools to complex software architecture documentation, knowledge management and modeling tools are used [88, 89]. The tools that can be provisioned in software architecting TSPACE requires a bundling mechanism that can enable provisioning of heterogeneous tools (e.g. desktop-based and web-based). Moreover, for complex scenarios, some components of a tool may need to access information from components of another tool, e.g. a decision support tool can be used for different types of the artifacts and with different types of tools. As a result, the tools bundling mechanism should be flexible enough to cater such needs. In certain cases, there can also be some constraints with respect to the underlying virtualized infrastructure (e.g. IaaS cloud virtual machines) that can host the tools to enable their operations within acceptable runtime quality parameters (e.g. performance, scalability, reliability etc.). The activities that are performed at this stage also take care of identification of such constraints. The artifacts that are produced in this activity provide guidelines to identify integration needs of the tools in a TSPACE and guide the reference architecture analysis and design process.

Participants' Roles: Software Architect.
Artifact(s) Consumed: TSPACE functional boundaries, required activities and tasks, and tools enactment/provisioning parameters and constraints.
Artifact(s) Produced: Design time constrains, tools bundling constraints and tools' provisioning/enactment parameters.

3.3.1.6. Stage 6 - Identification of Integration Needs

The activities that are performed at this stage focus on identifying integration needs of the tools that can be provisioned in TSPACE. With reference to software architecting domain, the integration mechanism should be flexible enough to accommodate different proprietary and standardized formats as well as support integration among heterogeneous types of tools (e.g. desktop-based, web-based and cloud-based tools). The tools that are provisioned in a TSPACE instance can vary and the integration mechanism should be flexible enough to adapt to the integration requirements of the provisioned tools. As the reference architecture is aimed at providing TaaS in a workspace, the integration mechanism should also support workspace requirements, such as awareness of the operations that are performed on the artifacts as a result of the users' activities [90]. The artifacts that are produced at this stage guide the reference architecture design and analysis process of integration.

Participants' Roles: Business Analyst and Software Architect.

Artifact(s) Consumed: Design time constrains and tools bundling constraints.
Artifact(s) Produced: Integration and collaboration models.

3.3.1.7. Stage 7 - Identification of Architecture Quality Characteristics

TSPACE is aimed at providing a bundled suite of tools following service model. As a result, its reference architecture needs to incorporate architecture quality requirements of cloud-enables services based system such as scalability [91], multi-tenancy [92] and dynamic provisioning [93]. The activities that are performed at this stage aim to identify important quality characteristics with reference to the domain in which TSPACE is designed. For software architecting domain, scalability, multi-tenancy and dynamic provisioning are important. For another domain such as software testing, elasticity [94] and reliability [95] can also be important. The artifacts that are produced as a result of this activity provide a foundation for runtime architecture quality requirements of TSPACE.

Participants' Roles: Software Architect.
Artifact(s) Consumed: Collaboration and integration models.
Artifact(s) Produced: TSPACE runtime architecture quality requirements.

3.3.1.8. Stage 8 - Reference Architecture Analysis and Design

The activities that are performed at this stage focus on design and analysis of TSPACE reference architecture. As TSPACE is an emerging domain and there are no comprehensive solutions available to describe all the elements of TSPACE (as discussed in Section 3.2), different strategies are adopted to design the reference architecture. According to the classification scheme presented by Angelov et al. [15], the reference architecture should be classified as if it is a standardization effort or preliminary proposition. For example, the provisioning part of the reference architecture can be referred as standardization. Existing cloud-provisioning approaches (that are described in Chapter 2) need to be adopted and tailored with respect to the specific context of TSPACE to standardized tools selection, enactment and provisioning. To structure information and artifacts in TSPACE and to facilitate integration among the tools, we have proposed the ontologies and architecture constructs for process-centric integration and semantic integration (to be discussed in Chapter 5 and Chapter 6). As the tools are to be offered as services in TSPACE as part of the workspace, we have also proposed architecture constructs to facilitate workspace specific activities such as awareness of the operations and collaborative exchange of the artifacts using TSPACE. The

part of the architecture associated with integration and workspace-specific activities can be classified as a preliminary proposition (according to Angelov et al.'s classification scheme [15]) as to the best of our knowledge, no standardized architectures exist to address these challenges in *aaS context.

A reference architecture design should be based on reference models and architecture styles and patterns [5, 50]. As TSPACE reference architecture focuses on software architecting, we begin with software architecture description standardization models (IEEE 1471-2000 [47] and ISO/IEC/IEEE 42010:2011 [7]) as discussed in Section 3.3.1.3. We have leveraged cloud architecture styles and patterns along with standardized architecture patterns [5, 50] to design different parts of TSPACE reference architecture and to achieve design time and runtime qualities in the architecture. While designing TSPACE reference architecture, we have taken inspiration from the cloud architecting solutions that have been discussed in Chapter 2. If TSPACE is to be used for mission-critical and safety-critical tools, then it is also important to have metrics that can be used to measure runtime quality parameters of the architecture. Empirical investigation of the reference architectures have revealed that absence of important views [51] in the reference architecture and details of the supporting algorithms and formalization to achieve the functionality of the reference architecture [51] impact its adoption and applicability. Hence, the reference architecture should encompass all the important views necessary to describe the reference architecture. For example, we have taken following steps to address the challenges. We have described details of the TSPACE formalization and information structuring approaches using ontologies [48] (to be discussed in Chapter 5). We have elaborated TSPACE reference architecture requirements (to be discussed in Chapter 4) and have described details of the reference architecture using logical, process and deployment views (to be discussed in Chapter 6) of 4+1 view model [51].

Participants' Roles: Software Architect.
Artifact(s) Consumed: Reference architecture models.
Artifact(s) Produced: TSPACE reference architecture in terms of multiple views.

3.3.1.9. Stage 9 - Reference Architecture Evaluation

Evaluation of the reference architecture is an important step for analyzing its feasibility and applicability. Different considerations for reference architecture evaluation have been proposed [16, 17, 43]. Avgeriou et al. [16] have proposed to evaluate the reference architecture with the help of scenarios and prototype implementation of the reference architecture. The use of scenarios is proposed to have implementation-independent evaluation. The evaluation scenarios need to be focused on important design time and runtime

qualities of the architecture. The evaluation of the implemented prototype in terms of implementation decisions such as platform choices, programming languages etc. is suggested to evaluate implementation capabilities of the reference architecture. Angelov et al. [17, 43] have argued that straightforward adoption of architecture evaluation methods such as Architecture Tradeoff Analysis Methods (ATAM) [13] and Software Architecture Analysis Methods (SAAM) [12] is not feasible because: gathering all the stakeholders for the reference architecture is not possible and generation of concrete scenarios for reference architecture evaluation is not feasible because components are often abstract. To cater these issues, the authors [17, 43] have argued that it is important to identify most relevant architecture requirements with the help of domain experts or domain models and then prepare scenarios by involving reference architecture users (stakeholders who are potential users of the reference architecture) in the evaluation process to generate abstract evaluation scenarios.

Other than above-mentioned challenges, TSPACE reference architecture evaluation activity has additional complexities. First, TSPACE needs to provision the tools that are required to perform the different activities and to provide support for the integration and workspace specific functions in a *aaS model. Hence, it is important that the evaluation activity should be focused on the parts of the reference architecture that are embodied by TSPACE boundaries rather than by the tools to be provisioned. Some of the key quality characteristics are derived from *aaS domain because of TSPACE reference architecture focuses on on-demand tools provisioning. Therefore, the evaluation activity should focus on identifying and analyzing the relevant quality attributes for the given domain. Finally, to evaluate the tools bundling and integration approach, the role of prototype becomes more critical in evaluating the applicability and feasibility of the reference architecture. As a result, TSPACE prototype plays a critical role for its evaluation and the tools that are selected for provisioning using the prototype should cover critical evaluation scenarios. The outcome of evaluation activity can trigger modification in the artifacts that were generated in previous stages as depicted in Figure 11.

Participants' Roles: User, Requirement Analyst, Business Analyst and Software Architect.
Artifact(s) Consumed: TSPACE Reference Architecture.
Artifact(s) Produced: Evaluation results.

3.3.1.10. Stage 10, 11 and 12 – TSPACE Reference Architecture Implementation and TSPACE Provisioning

The activities that encompass the last three stages of TSPACE reference architecture design focus on TSPACE reference architecture implementation. An important step for implementation of the reference architecture is the identification and selection of an appropriate IaaS cloud platform that can be used to host the tools. Specific infrastructure needs of the tools as well the components of the tools should be considered for selecting the platforms. Once tools are registered and hosted by the TSPACE, these can be provisioned for the usage in a TSPACE instance. As discussed earlier, the implementation can also serve as mean to evaluate the applicability of the reference architecture.

Participants' Roles: Developer and Software Architect.
Documents Consumed: TSPACE reference architecture and potential cloud platforms analysis document.
Documents Produced: Executable TSPACE.

3.3.2. Discussion

By providing TSPACE reference architecture development process, we have attempted to identify key elements and important design stages that encompass TSPACE reference architecture development. The elements that are described in Section 3.2 provide a foundation for identification of different subsystems and components of the reference architecture. Multiple stages of the reference architecture design process identify sources of information and important design considerations. TSPACE reference architecture that is elaborated in detail in Chapter 4, Chapter 5 and Chapter 6 covers these stages and is designed following the design guidelines that have been discussed in the process. As it is clear from the TSPACE design process framework diagram (Figure 11), many of the activities of the process are of iterative nature and these cannot be distinctively segregated. Following is a brief description of the distribution of the activities over chapters of this dissertation. Chapter 4 elaborates TSPACE requirements with respect to the reference architecture elements that are discussed in this chapter and focuses on software architecting domain models, the nature of tools that can be provisioned in software architecting TSPACE and a high level overview of the functional and non-functional requirements. Chapter 4 also briefly covers requirements aspects of stage 3, 4, 5, 6 and 7. Chapter 5 presents TSPACE conceptual and information structuring model in terms of TSPACE ontology and provides details of the activities associated with stage 3, 4, 5 and 6. Chapter 6 provides details of the reference architecture and covers all activities of stage 8 and stage 9 and provides an overview of the prototype

implementation encompassing the activities associated with stage 10, 11 and 12.

3.4. Related Work

Researchers have attempted to establish standardized activities and frameworks that can be used for design and documentation of the reference architecture. The most comprehensive work in this regard is reported by Angelov et al. [15, 17, 43, 44]. Their work describes the classification techniques of the reference architecture based upon the maturity of the domain and how the reference architecture is designed. For mature domains, the aim of the reference architecture is to provide standardization of the architecture, whereas, for emerging domains, the purpose is to facilitate the design of concrete architectures in multiple individual organizations. Some of the problems that are associated with reference architecture design are missing design methods, problems with defining non-functional requirements, problems with selecting appropriate views, absence of software reference architecture documentation method and absence of software reference architecture evaluation [44]. In the proposed TSPACE reference architecture design process, we have explicitly catered all of the above-mentioned challenges to streamline the reference architecture design process and to have explicit stages for design and documentation methods, define non-functional requirements, select appropriate views and evaluation strategies.

Avgeriou [16] suggests representing a reference architecture using multiple viewpoints of Rational Unified Process (RUP) including logical viewpoint, deployment viewpoint, implementation viewpoint and data viewpoint. Avgeriou has emphasized that the reference architecture should be evaluated using both scenario-based and architecture prototype-based evaluation with respect to run-time and development qualities [16]. Another work by Nakagawa et al. [96] has proposed the use of ontologies to identify different components of the reference architecture. Fernandez et al. [97] have described the key documentation elements of a software reference architecture. The documentation elements include technical design, architecture knowledge and experiences and management documentation. In TSPACE reference architecture, we have described technical design and architecture knowledge in details. However, we have not attempted to address the management documentation (during applications of the reference architecture in different setting) as it is out of the scope of this dissertation.

3.5. Conclusions

In this chapter, we have described Reference Architecture Development process Framework (RADeF) that guides the development of TSPACE reference architecture. First, we have identified and elaborated the key elements of TSPACE, and have described the relationship among the elements. Then we have elaborated each step of the TSPACE reference architecture design process. We have identified the artifacts that are needed as a prerequisite of each stage and the artifacts that are produced as the result of the activities that are performed at each stage. We have also identified the role of the key stakeholders involved at each stage of TSPACE reference architecture. It is to be noted that although we have made clear demarcation among the stages, it can be hard to have a clear demarcation in many cases because of the highly iterative nature of the design process as indicated in Figure 11. The process provides the foundation for the TSPACE reference architecture design activities that are covered in forthcoming chapters of this dissertation. The process suggests that the TSPACE meta-models play a significant role in identifying architecture requirements and components of the reference architecture in a given domain. The documentation approaches that are adopted for the reference architecture design documentation also play a vital role for its elaboration and adoption. Last but not the least, evaluation of the reference architecture is an important activity and should be carefully conducted by considering the parameters and constraints under which the reference architecture is expected to be initialized and operated.

Chapter 4. Business Drivers and Requirements of TSPACE

In this chapter, we describe the business drivers and requirements of Tools as a service workSPACE (TSPACE). Business drivers are presented in terms of TSPACE value and benefits. TSPACE requirements are presented in terms of the reference architecture documentation requirements as well as functional and non-functional (quality) requirements that TSPACE reference architecture aims to achieve. This analysis allows the potential adopters of the reference architecture to identify the benefits that can be achieved by having TSPACE. The requirements also provide a foundation for the detailed analysis and design of TSPACE reference architecture that is to be presented in forthcoming chapters of this dissertation.

Parts of this chapter have been presented in [38].

4.1. Introduction

In this chapter, we provide a detailed description of the business values of TSPACE and TSPACE reference architecture requirements. We have taken following steps to achieve this objective. First, we have investigated benefits that can be achieved by providing TaaS as part of the cloud-enabled workspace that we are referring as TSPACE. Next, we have briefly discussed why it is necessary to treat TaaS differently than SaaS. After that, we focus on the reference architecture documentation requirements of TSPACE and the multiple levels of abstractions that TSPACE reference architecture needs to address. In subsequent steps, we provide a detailed analysis of TSPACE reference architecture's functional and non-functional (quality) requirements for software architecting domain. We also briefly indicate what additional requirements need to be incorporated for the adoption of TSPACE reference architecture in other engineering domains (however, it is not main focus of this chapter and the dissertation).

In addition, we describe how the discussion that is being reported in this chapter is related to the reference architecture design process framework RADeF (discussed in Chapter 3). With reference to the relevant stages of RADeF, functional and non-functional requirements and constrains under which TSPACE reference architecture is to be designed and operated are elaborated.

4.2. Value Analysis and Benefits of TSPACE

The benefits of cloud computing for addressing software engineering and development challenges have been reported in the literature [86, 98] and can facilitate both collocated as well as distributed teams. According to Hashmi et al. [98], geographical and temporal issues are among the major challenges faced by distributed teams. Geographical distances result in communication gaps, ambiguity in domain and product requirements, and challenges in transferring domain knowledge. Temporal issues result in reduced visibility of product artifacts and reduced product quality. The authors also discuss the utilization of cloud-based infrastructure for achieving business and technology alignment, interoperability and diversification of tools [98]. Maleej [55] has indicated that on average four to five tools are used to complete a single software engineering activity. In his position paper on future of software engineering, Whitehead [87] has emphasized the need to have an integration support among heterogeneous types of tools (including desktop-based tools and web-based tools).

The use of tools has been described in [86] along with different lifecycle phases of software development. In the study [86], the tools have been clustered into four groups. Each group is aimed at addressing specific software engineering and development challenges. An effort to optimize software development tools requires an integrated set of tools. The tools optimization effort addresses the technical issues related to software development and highlights the need for software engineering tools to provide solutions to one specific aspect of software engineering, e.g. software architecting tools, testing tools or requirements management tools. The project optimization effort includes the tools that are needed for project management and governance. The necessity of having optimized software development and engineering tools signifies the need of an *aaS approach to solve the engineering and development issues. We assert that the availability of TaaS can allow organizations and individual engineers to have easy alignment of processes and tools because there would not be any need to set up new (or optimize existing) tools and infrastructure for every new project. Following is the brief overview of what can be achieved by having TaaS in general and TSPACE in particular.

4.2.1. On-demand Access to the tools

Software development consists of multiple phases and each phase requires a specific set of tools, as the tools used in one phase may not be used much when a project is in another development phase, especially the tools that are used in the later development phases. For example, in traditional waterfall and spiral development process models [99], when a project is in the requirements

engineering phase, the tools required in design and development phases are not used. Similarly, when a project is in the testing phase, the requirements engineering and design tools are not used frequently by a large number of stakeholders. Organizations still need to acquire and maintain the tools for the whole span of a project. Since a cloud computing utility model can support the provision and acquisition of tools for a specific time frame and an organization can have the flexibility of acquiring a specific set of tools at the point when it is needed [98], it means the organizations can acquire tools when the tools are to be used. Some development phases may require more resources, for example, in order to simulate test cases associated with computing and data-intensive use cases, additional hardware and software resources are required. These resources are utilized only for a small fraction of time as compared to the overall system development life cycle. Using the computing as a utility model, organizations can acquire additional resources for the resource-hungry activities.

Furthermore, organizations need different tools in accordance with the requirement of each specific project. Configuring and maintaining a large variety of tools are resource and time intensive undertakings. When the projects are not in active stage, the tools and associated infrastructure still occupy the resources. These constraints may limit an organization's options to work in different business and application domains. The availability of TaaS through a cloud-based infrastructure can help organizations to acquire and release the desired set of tools according to the specific requirements of projects and only for the phases when the tools are really needed.

4.2.2. Tools Alignment with Organizational Processes

Organizations involved in software engineering and development activities have to align their software development processes with each other in order to effectively work on product artifacts [98]. As a consequence, organizations also require some tools for facilitating the processes management itself. Software design tools and software testing tools are common examples of tools that are closely tied with process. For a small sized organization, it is not cost effective to have a specific set of tools for each project (under development). If tools are available as TaaS, these can be acquired according to the process requirement of each single project and can be released afterwards when these are not needed.

4.2.3. Support for Awareness of the Operations and Collaboration

When multiple tools are used to perform different activities on the artifacts [55], providing support for collaboration and awareness is not trivial. Especially, when multiple users (or stakeholders) work on the related artifacts.

The magnitude of the complexity increases when multiple sites are involved in the activities [100]. Studies have shown that implicit information about a project and the communication that exists between collocated teams is not available to remote teams [101]. That is why emails, text chats, and instant messages are used as informal ways to increase awareness [101]. However, information that is exchanged through emails, texts and instant messages is detached from the actual artifacts and is not traceable. A number of tools have been designed to provide visual representation of participants' actions and allow them to view each other's work [101]. The traditional desktop-based tools require extensive exchange of information with each other for providing awareness to other software engineering tools because artifacts are scattered at multiple geographic locations. In case of TaaS hosted in a cloud, whenever users want to access artifacts, they will have the most updated view without any additional effort. Tools hosted in a cloud, minimize the burden of data exchange between sites, hence reducing latency delays. Moreover, incorporating additional features of awareness of the operations that are performed on the artifacts in TaaS can help to establish and maintain relations among the artifacts and collaborative exchange of information because these have access to the shared data structures provided by a cloud platform (infrastructure) hosting TaaS.

4.2.4. Working with Sensitive Artifacts and Data

Dealing with sensitive data in certain projects can be a challenging task. There are certain laws that restrict data movement outside a particular geographic location. Cloud-based infrastructure can help to address these challenges. For example, cloud-based design and development tools can have access to the real data that is of sensitive nature. These tools can deploy system components handling sensitive data in cloud inside the same geographic location where data are present to run test cases and collect the resulting metrics while being accessible from outside that region.

Cloud-based tools can also mitigate the overhead of data movements by providing access to data that is hosted closer to the tools. Moving a huge volume of data wastes a lot of time and network bandwidth. Cloud-based tools can eliminate the need to move the data from all sites of a project and allow the development an application inside the region where data can be easily and rapidly accessed.

4.2.5. Access to Sensitive Technology

Software artifacts developed in a distributed environment may be part of a very complex system requiring integration with expensive and sensitive hardware. Replication of expensive and technology sensitive hardware at each

location may not be possible because of cost or the sensitive nature of the hardware technology. If such hardware resources can be connected with software development tools on a private cloud, it can allow remote sites to participate in the development of such a project. Stakeholders in remote areas can perform their activities from remote locations using tools that are hosted on a cloud-based infrastructure.

4.2.6. Establishment of Knowledge Ecosystem

Organizations involved in software development maintain their in-house knowledge bases for internally sharing knowledge. This knowledge is often maintained outside the scope of the tools, which are used to perform the actual activities of software architecture design and development. If knowledge acquisitions and management infrastructure is provided by a TaaS hosting platform with integration support with the actual TaaS being used by organizations, knowledge maintenance and sharing can become convenient and transparent to the end users.

4.3. TSPACE Reference Architecture Requirements

This section presents a detailed analysis of the functional and non-functional requirements of the TSPACE reference architecture. Our research on TSPACE has been motivated by the need to provide a workspace where all the required tools can be bundled in a tools suite and provisioned as a service. The TSPACE purports to enable user(s) to have on-demand provisioning of tools and semantically integrated artifacts in a Just-in-Time (JIT) fashion. The functional requirements are the functionalities that should be supported and the non-functional requirements are the quality attributes that should be achieved by the design of TSPACE reference architecture. The reported requirements are based on our work on a TaaS infrastructure [38] and a review of the literature on important quality characteristics of cloud-based systems [42]. TSPACE functional and quality requirements (non-functional) are summarized in Table 31.

4.3.1. Reference Architecture Documentation Requirements

One of the initial stages of a TSPACE reference architecture development framework (described in Chapter 3) is to identify the scope of the TSPACE reference architecture documentation approach and to determine the maturity of the domain for which the reference architecture is to be designed. Since a reference architecture provides valuable guidelines for designing a concrete architecture, it is important to describe a reference architecture as comprehensively as possible and in an easy-to-understand way. We describe the proposed reference architecture using a systematic approach that

advocates the use of context, goal and design dimensions of a reference architecture [15, 16] as described in Chapter 3.

Table 30: TSPACE Reference Architecture Documentation

Dimension	Sub-dimension	Description
Context	Who defines it?	It determines the context in which the reference architecture is designed.
	Where will it be used?	It determines potential usage of the reference architecture.
	What is the maturity stage of the domain?	It describes the maturity stage of the domain for which reference architecture is being designed.
Goal	Why is it defined?	It describes the purpose of defining the reference architecture.
Design	What is described?	It describes what is described in the reference architecture.
	How is it described?	It describes how the reference architecture is described (e.g. using textual description or UML diagrams).
	How is it represented?	It describes whether the reference architecture is presented using informal, formal or semi-formal approaches.
Instantiation	How is it instantiated?	It describes the guidelines for adoption and instantiation of the reference architecture.
Evaluation	How is it evaluated?	It describes the steps for the evaluation of the reference architecture.

The context dimension covers the purpose, development organization and maturity stage (e.g. preliminary or classic) of a reference architecture [15]. The goal dimension encompasses business goals and quality attributes as well as the purpose of defining a reference architecture (e.g. to standardize concrete architecture or to facilitate design of concrete architecture). The design dimension elaborates whether the reference architecture is concrete or abstract and whether the reference architecture has been described using formal, semiformal or informal approaches. The reference architecture also needs to describe: (a) the approach used to document a reference architecture, (b) guidelines for instantiation of a reference architecture and (c) evaluation of a reference architecture corresponding to desired quality attributes [16]. Table 30 describes the reference architecture documentation and description approach (based on the reference architecture description model presented in Table 29 in Chapter 3).

4.3.2. Functional Requirements

We have identified the functional requirements based on the key features required by the reference architecture according to different lifecycle phases of TSPACE These are tools enactment and provisioning, integration of the tools and semantic integration among the artifacts associated with the tools after enactment and awareness of the operations that are performed on the artifacts as a result of the users' (stakeholders') activities during the tools' lifecycle. Following are the functional requirements that have been enhanced based on our work on TaaS infrastructure requirements [38].

4.3.2.1. FR1 - Enactment and Provisioning of TSPACE

Enactment and provisioning of a TSPACE and associated tools according to the requirements of different activities of a project is important, and while provisioning tools, the architectural support should also consider the specific enactment constraints (e.g. location and resource requirements parameter of the tools).

4.3.2.2. FR2 - Tools Management

There is a large variety of tools that are used in software engineering in general and software architecting in particular (e.g. architecture description tools, architecture knowledge management tools and architecture modeling tools). Each of these tools can have many different versions (and different features in each version). Each of the tenants can request a specific set of the tools as well as specific versions of the desired tools. So, in order to have a comprehensive support for hosting these tools, TSPACE should be able to maintain different tools and their versions (along with features that are supported in each version of the tools). It is evident that maintaining this kind of complex environment in a cloud increases the complexity of TSPACE reference architecture.

4.3.2.3. FR3 - Tools Bundling

In order to provide an end-to-end solution covering all the phases of software development life cycle, the tools hosted on clouds dealing with one phase should be able to integrate with tools serving preceding and proceeding development life cycle phases. As each activity of a project uses the artifacts produced by the preceding activity, there is a need to bundle the tools that can support different activities of the software architecting lifecycle. There is also a need to provide compatibility between tools that can be used for similar architecting activities. This will enable the users (tenant) to select tools of

their own choice, which are more suitable to projects tasks and the activities. TSPACE reference architecture should have the ability to provision TaaS to end-users as a bundle of tools to provide a comprehensive suite of tools for the activities to be undertaken.

4.3.2.4. FR4 - Tools Integration

TSPACE integration needs to facilitate on-demand provisioning of the tools in a suite. As the tools are to be provisioned as part of a suite of tools, on-demand provisioned require Just-in-Time integration, so that the tools can work in combination with other tools. The integration scheme needs to be compliant with the specific requirements of the domain in which the tools are to be used. Software architecting, like other software engineering activities, is carried out according to specific organizational processes. Common software architecting activities include architecture significant requirements analysis, scenario elicitation, architecture design analysis, architecture modeling and architecture evaluation [8]. As discussed earlier, there are different types of tools that can be used to perform the activities, ranging from simple text-based tools and web-based tools to more complex architecture modeling tools. TSPACE reference architecture needs to support two types of integration: (a) process-centric integration that can facilitate exchange of the artifacts among the tools according to the process that governs the artifacts' development and (b) semantic integration among the tools so that the artifacts that are generated and maintained by the tools can be related to other artifacts. TSPACE reference architecture integration needs are depicted in Figure 12.

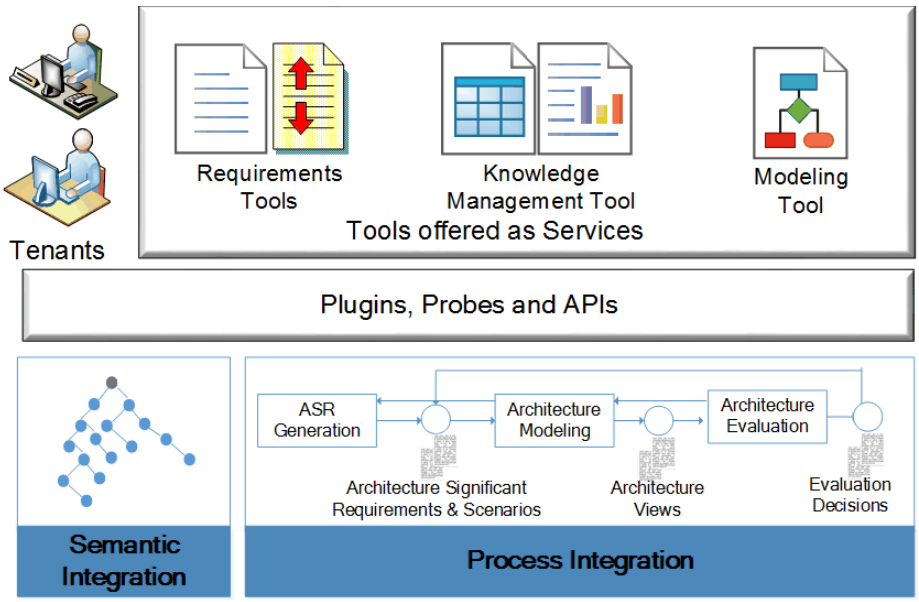


Figure 12: TSPACE Integration

4.3.2.4.1. Process-centric Integration among the Tools

TSPACE reference architecture should provide support to address the challenges associated with tools' alignment with the processes. Software architecting tools need to be smoothly integrated with each other in order to support end-to-end activities. One of the key characteristics that distinguishes TaaS from SaaS is the inherit need of software engineering and development tools to work as an ecosystem of applications. Maalej et al. [55] has reported following problems that traditional software integration mechanisms fail to cater: (a) identification of the changes in artifacts after a task is completed, (b) retrieving artifacts when they are made available by other stakeholders as a result of preceding process activities (is often done through manual approaches) and (c) synchronizing the artifact as they are developed in collaborative activities. TSPACE reference architecture should provide support to address the above-mentioned issues.

4.3.2.4.2. Semantic Integration among the Artifacts

Semantic integration among artifacts that are maintained by the tools constituting a TSPACE after enactment is vital. The TSPACE consists of multiple tools that may have proprietary formats to store artifacts. TSPACE reference architecture should support semantic integration among different types and formats of artifacts associated with the tools that constitute TSPACE.

4.3.2.5. FR5 - Support for Awareness of the Operations that are performed on the Artifacts

Awareness of (users') operations on the artifacts that are carried out during the lifecycle of a TSPACE using multiple tools is important when multiple tools are being used to perform the tasks associated with an activity. The tasks can be performed by the same users or by different users belonging to same tenant. Multiple artifacts are produced or consumed during the lifecycle of a specific project for which a suite of TaaS is provisioned by TSPACE. Hence, there is a need to raise awareness about what actions are performed on the artifacts, and the reference architecture should support such awareness.

4.3.3. Quality Requirements

The quality (i.e. non-functional) requirements of TSPACE are classified into two categories: (a) runtime qualities that are discernable once a system is operational [5, 43], referred to TSPACE quality requirements, and (b) design time qualities that are discernable while a system is being designed, referred

to reference architecture quality requirements. Following are the design time and runtime quality requirements for a TSPACE.

4.3.3.1. TSPACE Quality Requirements

4.3.3.1.1. QR1 - Automated Provisioning

TSPACE reference architecture needs to support automated provisioning of TSPACE so that the required tools can be acquired automatically for a project based on the specified constraints (e.g. constraints on the location of the tools).

4.3.3.1.2. QR2 - Multi-tenancy

Being a cloud-based platform, a TSPACE needs to be a multi-tenant [102] platform (with architectural support). Multi-tenancy is defined as a software system quality that enables it to provide logical isolation between the data of different tenant and enables a single instance of a software system to have different configurations as per the requirements of each tenant. Each TSPACE instance (a bundled suite of tools provisioned for a specific tenant) can have its own set of tools and rules for awareness of the operations that are performed on the tools. A particular tenant shall be able to specify and access all its specified features and configurations.

4.3.3.1.3. QR3 - Scalability

TSPACE shall scale up or down as the number of activities that are performed using the tools increases or decreases.

4.3.3.2. Reference Architecture Quality Requirements

4.3.3.2.1. QR4 - Flexibility

As the tools in a specific instance of a TSPACE depend upon the activities to be performed within a project, TSPACE reference architecture shall be flexible enough to provide semantic integration and awareness support for different tools.

4.3.3.2.2. QR5 - TSPACE Interoperability

There are large number of commercial and open source tools available that can support software engineering and software architecting activities. To have

plug and playable support for tools bundling, TSPACE reference architecture should provide support for the tools to interoperate on the artifacts that are maintained using standardized or non-standardized formats. Tools interoperability is important for following reasons. (a) For smooth functioning of organizations, it is important that the organizations do not have to change the existing software architecting tools for every new project. If TSPACE supports interoperability with existing tools, the tools can be used as an interface to the TSPACE while underlying data structures are maintained in clouds. It makes it easier to have access to shared artifacts. (b) It may not be possible to have alternates of every single tool available in clouds as TaaS. TSPACE interoperability support with other tools makes infrastructure adoption easier and smooth. TSPACE reference architecture shall provide semantic integration and awareness (of the operations) support for different types of artifacts (e.g. knowledge management, textual documentation and UML modeling tools that are used for software architecting related activities).

4.3.3.2.3. QR6 - Completeness, Feasibility and Applicability

Bass et al. [5] have presented a number of general quality requirements including completeness, feasibility and applicability. Completeness of TSPACE reference architecture is important so that it can serve as a guiding model for designing a specific instance of TSPACE. It should also be feasible to implement a reference architecture. The applicability quality characteristic is also important so that a reference architecture can be used to design and evaluate a concrete architecture of TSPACE.

4.3.3.2.4. QR7 - Modifiability

The tools associated with a TSPACE may come from different vendors. Those tools can be provisioned using public, private or hybrid deployment models. Hence, TSPACE reference architecture shall support modifiability for seamless integration among different modules of a TSPACE with the provisioned tools that can be provided by different vendors.

Table 31: TSPACE Requirements Summary

	ID	Requirement
Functional Requirements	FR1	*Provisioning:* TSPACE should support provisioning of TSPACE components and associated tools according to the requirements of different activities that are to be carried out using the tools and constraints on tools enactment.
	FR2	*Tools Repository:* TSPACE should provide support for repository management of different types of tools available for provisioning.

		FR3	*Tools Bundling:* TSPACE should provide support for tools bundling according to the required activities.
		FR4	*Integration:* TSPACE should support semantic integration among the artifacts of different types as a TSPACE instance consists of multiple tools that can use different formats to store the artifacts. TSPACE should provide support for process centric integration (and collaboration) among the tools that are used to perform process centric activities.
		FR5	*Support for Awareness of Operations:* Multiple artifacts are produced or consumed during the lifecycle of a specific project for which a TSPACE is to be instantiated. Hence, TSPACE should provide support for awareness of the users' activities associated with the operations that are performed on the artifacts using the tools.
Quality (Non-functional) Requirements	***TSPACE Quality***	QR1	*Automated Provisioning:* TSPACE should support automated provisioning and bundling of the tools.
		QR2	*Multi-tenancy:* Being a cloud-based platform, TSPACE needs to be a multi-tenant **[102]** platform. Each TSPACE instance shall have its own set of tools and rules for awareness. A particular tenant shall be able to access all its specified features and configurations.
		QR3	*Scalability:* TSPACE should scale as the tools that are provisioned and the number of activities that are performed using the tools increases.
	Reference Architecture Quality	QR4	*Flexibility:* As the tools in a specific instance of a TSPACE depend upon the activities to be performed within a project, the reference architecture shall be flexible enough to provide semantic integration and awareness support for different types of tools (desktop-based, web-based etc.).
		QR5	*Interoperability:* reference architecture shall provide interoperability, integration and awareness support for different types of the tools and artifacts (e.g. textual documentation and UML models).
		QR6	*Completeness, Feasibility and Applicability:* TSPACE reference architecture shall positively address completeness, feasibility and applicability. Completeness of the reference architecture is important so that it can serve as a guiding model for designing a specific instance of TSPACE. The applicability quality characteristic is important so that the reference architecture can be used to design and evaluate a concrete architecture.
		QR7	*Modifiability:* TSPACE reference architecture shall positively address modifiability so that it can easily be adopted in different organizational contexts.

4.4. Conclusions

The adoption of new ways of provisioning tools in cloud-enabled workspaces should improve the performance of the organizations and improve the chances of more smooth operations of the software development projects. Tools as a Service (TaaS) in cloud-enabled workspaces should reduce tools infrastructure and maintenance costs and provide easy access to the tools on demand. Furthermore, TSPACE offers new possibilities for establishing an ecosystem of tools that are inline with organizations' software engineering and development processes.

In this chapter, we have described the business foundation for on-demand provisioning of TaaS in the context of a cloud-enabled workspace (which is referred as TSPACE). We have elaborated detailed requirements for TSPACE reference architecture documentation, its functional requirements and its quality requirements (quality requirements of TSPACE as well as quality requirements of TSPACE reference architecture). The work that has been presented in this chapter aims to facilitate a conceptual design of TSPACE reference architecture that is described in forthcoming chapters.

Chapter 5. Ontologies for Structuring and Formalization of TSPACE

To capture the relationship between different entities constituting Tools as a service workSPACE (TSPACE) and to incorporate requirements associated with semantic integration and awareness in TSPACE, as discussed in Chapter 4, formalization of TSPACE concepts is required. This chapter describes an ontology-based approach to formalize TSPACE and tools selection and provisioning in TSPACE. The approach consists of a suite of ontologies to characterize activities, tasks, artifacts, tools' features and stakeholders' requirements of the tools, and to support semantic integration among artifacts in TSPACE. The ontologies provide a structured mechanism to support semantic integration and to raise awareness (artifacts' addition, modification sharing and conflicts) of the stakeholders' activities corresponding to the artifacts. This chapter elaborates the details of the developed ontologies, how the ontologies are instantiated, populated and used during lifecycle of the tools in TSPACE.

Parts of this chapter have been presented in [49].

5.1. Introduction

Provisioning of the tools in a TSPACE instance and providing support for the different activities and tasks during lifecycle of a TSPACE instance is not trivial. TSPACE can consist of a number of tools that can be used to perform various activities related to software architecting. To provision the tools for the end users, TSPACE not only requires facilitation of the selection and provisioning of the tools but must also provide seamless operations of the tools in terms of distribution of the activities over various tools and integration among the artifacts maintained by the tools. Multiple vendors using different technology paradigms and using different programming languages can provide the tools to be provisioned by TSPACE. For example, the majority of the tools that are used for architecture modeling such as Microsoft Visio [103] and ArgoUML [104] are developed on top of a desktop-based paradigm. The desktop and cloud-based word processing tools (e.g. Microsoft Office Suite [105] and Google Docs [106]) and specialized web-based applications (e.g. PakMe [107]) can be used for architecture documentation (architecture scenario description, architecture significant requirements elicitation and architecture design decisions documentation). Heterogeneous technological paradigms and involvement of multiple vendors highlight the importance of having a gluing mechanism that facilitates the

selection of the appropriate tools from the pool of available tools and a seamless integration among the tools. Involvement of the heterogeneous tools requires a solution that is applicable and extendable for various types of the tools, irrespective of the technological paradigm and a tool's vendor.

We have leveraged semantic integration technologies to address the above-mentioned challenges of hosting and provisioning tools as services. We have proposed ontologies for TSPACE. The use of ontologies in a specific domain can provide a powerful mechanism to semantically relate unstructured information [108]. Ontologies also facilitate communication, integration and reasoning [108]. Our ontologies-based solution enables the provisioning of Tools as a Service (TaaS) for performing different activities using appropriate tools hosted on clouds, without the individual tools focusing on how to relate the artifacts and data across multiple tools. The users (stakeholders) can choose a set of tools to perform specific activities using the tools. The selection of the relevant tools can be based on a number of reasons, including but not limited to organizational policies, stakeholders' preferences for the tools, the tasks and the activities related to the project and that are to be performed using the tools and process requirements of the projects. Restricting stakeholders to a specific set of tools is not a viable solution for performing complex activities. If the projects' stakeholders have the flexibility to choose from a set of tools, the provisioning mechanism needs to provide a flexible way to support tools selection from the set of tools according to the desired needs as well as to provide inter tool integration, so that the artifacts that are produced or consumed in one tool can be related/integrated with the artifacts that are being maintained in other tools. Moreover, the integration mechanism should also provide support for additional collaboration and awareness activities among the users who perform the activities using different tools.

Our proposed ontologies provide solutions to three main lifecycle phases of the TSPACE. (a) The solution supports selection of the tools that are to be provisioned as part of the workspace. (b) Once TSPACE is enacted, the solution provides support for semantic integration among the heterogeneous artifacts that are produced and maintained using different tools. (c) The solution provides support for awareness of the activities that are performed by the stakeholders using different tools. The awareness mechanism encompasses the activities that are performed on the semantically related artifacts and any conflicts that can occur as a result of the activities. However, as software architecting is highly complex domain, our proposed approach can only partially automate the conflict identification mechanisms by identifying the potential areas of conflicts. The stakeholders working on the artifacts using different tools make the final decisions.

The main contributions of the research that is being presented in this chapter are:

- TSPACE ontologies that can be used to capture concepts of TSPACE and include Capability Ontology, Tools and Artifacts Ontology, Change Ontology and Annotation Ontology.
- Identification of the abstract concepts (content elements), relationships between different types of content elements (CEs) of the TSPACE and the definition of the rules based on ontologies to raise awareness of the operations that are performed on the tools (e.g. addition, modification and deletion) and to provide a foundation for conflict identification. The identification of the relations among different concepts and elements of TSPACE is important for a number of reasons. The tools that can be provisioned as a part of TSPACE can be associated with complex domains (such as in our case it is software architecting). There is a need to define abstract concepts and relations, because discovery of all the concepts and elements of TSPACE and the complex relations between the concepts and elements at runtime [109] without a predefined relationship meta-model is not possible. The abstract CEs and their relations are extended at runtime as the artifacts are produced and linked to each other in TSPACE. The abstract relationships also facilitate use of the dynamic rules for information extraction for awareness and conflict identification.
- Rules and algorithms that are used for tools selection and identification of the conflicts.

5.2. Solution Approach

As described in the introductory part of this dissertation, our research on TSPACE has been motivated by the need to have easy and on-demand access to tools required for performing specific activities associated with software architecting (e.g., architecture documentation and architecture modeling). Some of the advantages of TSPACE include provisioning of tools for specific needs of a project, tools alignment to organizational processes, support for organization-wide collaboration and awareness of the operations that are performed on the tools, and finally provision to work with decentralized artifacts using the tools [38]. The key quality requirements of TSPACE require support for bundling multiple tools together in a suite because different stakeholders may have different requirements for tools to perform specific activities. In order to provision TaaS, TSPACE should provide support for mechanisms through which (a) capabilities (functional and non-functional characteristics) supported by the tools and required by the stakeholders can be captured, (b) related artifacts and data elements

maintained among the different tools can be associated with each other and (c) an awareness mechanism through which notifications of operations and changes can be propagated across tools provisioned as a part of TSPACE.

As previously stated, a project's stakeholders usually work with multiple tools provided by commercial vendors or Open Source communities. These tools need architectural level support for interoperability so that the artifacts produced in different formats (texts, diagrams, standardized formats and proprietary formats) can be integrated with each other. We have proposed to leverage semantic technologies for tools integration; however, our solution needs to be complemented by appropriate architecture abstractions for information discovery from tools. The details of architecture and integration meta-models are described in Chapter 6. TSPACE also needs to have a set of rules to support information discovery and awareness.

An ontology is defined as "a formal and explicit specification of a shared conceptualization" [48] and consists of a shared vocabulary that can be used to model a specific domain [48]. Ontologies are widely used to define semantic relationships among data and to maintain knowledge of semantic relationships. The knowledge is often maintained using a web-based application such as Semantic Wikis [110]. Annotating digital documents is a common strategy to adapt digital documents to the Web [111]. Ontologies are an effective way of modeling, sharing and reusing organizational knowledge [112]. We have proposed TSPACE ontologies to achieve following objectives:

- Identify all the concepts that constitute TSPACE to provide TaaS in general and software architecting TaaS in particular.
- Provide a structure of all the elements constituting TSPACE and relationships among the elements.
- Provide a mechanism to manage and organize different artifacts that are produced or consumed by the tools in a TSPACE instance.
- Provide a dynamic information retrieval mechanism that can provide a backbone for different forms of information extraction. The extracted information can be used to raise the awareness, provide support to identify conflicts that can emerge as a result of activities that are performed using different tools in a TSPACE instance.

In our work, the proposed TSPACE ontology model consists of four specializations and it is represented with 4-tuple elements:

$$\text{TspaceOnt} = < \text{CapOnt}, \text{ArtToolOnt}, \text{ChaOnt}, \text{AnnOnt} >$$

A brief description of each specialization is as follows:

- *Capability Ontology (CapOnt)* is used to capture the capability of individual tools (functional and non-functional features) that can be provisioned in a TSPACE instance and to capture users requirements of the functionality from a TSPACE instance. *CapOnt* is also used to instantiate the underlying ontology model with respect to the tools that are provisioned in a TSPACE instance. The tools bundling is achieved by matching tools' supported capability with the stakeholders' (end users) required capabilities.
- *Artifacts and Tools Ontology (ArtToolOnt)* is used to establish relationships among the artifacts, activities, tasks, and the tools that are used to perform activities and tasks.
- *Change Ontology (ChaOnt)* complements *ArtToolOnt* and monitors and tracks changes on a single content element (CE) in a TSPACE instance.
- *Annotation Ontology (AnnOnt)* also complements *ArtToolOnt*. *AnnOnt* acts in context of multiple artifacts. *AnnOnt* is used to annotate artifacts that are produced or consumed in a TSPACE instance, establishes relationships between multiple artifacts, monitor changes that are performed and analyze impact of changes among the artifacts (that are triggered as a result of the stakeholders' activities and operations on the artifacts).

While *ArtToolOnt* establishes and manages relationship among the artifacts, activities and tools, *ChaOnt* and *AnnOnt* take care of the activities that are performed on the artifacts using the tools in a TSPACE instance. Figure 13 shows TSPACE ontology meta-model and the relationships among TSPACE ontology specializations. Figure 13 shows four specializations of TSPACE ontologies including Capability (CapO*nt*), Artifact and Tool (*ArtToolOnt*), Change (*ChaOnt*) and Annotation (*AnnOnt*) ontology. Association between *ChaOnt* and *ArtToolOnt* indicates that change ontology monitors the operations that are performed on the artifacts using the tool. *AnnOnt* uses the *Relation* element of the *ArtToolOnt* to annotate the artifacts. The details of the ontologies and their constituting elements are described in following sections.

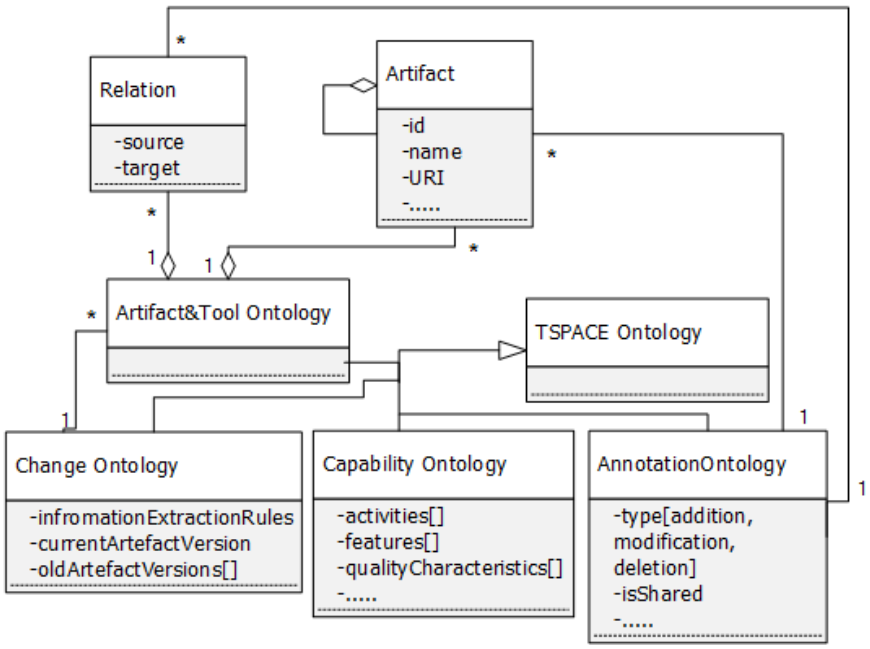


Figure 13: TSPACE Ontologies Relation

The strategy to build ontologies for a specific domain is a critical step. Two different approaches are used to build ontologies: manual and automated [113]. The manual approach is based on expert knowledge, whilst the automated approach is based on information extraction and natural language processing techniques [113]. The automated ontology generation approach is used to extract concepts from the data and structure the concepts in hierarchical order [109], however the automated techniques cannot be used to identify the complex relations between the concepts associated with a particular domain. Hence, because of complex nature of the activities involved in software architecting and the relationship between the artifacts and different elements of the artifacts, we have to adopt a combination of manual and semi automated ontology building approaches. We have identified high-level core concepts and relationships between the concepts with the help of software architecture documentation domain model. We have also leveraged our experiences with designing architectures of the software systems to refine the concepts extracted from domain model. The specializations of the high-level core ontology concepts are populated using semi-automated techniques as artifacts are produced in a specific instance of TSPACE using respective tools. The relationship between the specialization of a dynamically identified concept or content element (CE) is same as of its abstract parent with other concepts or CEs. As in this dissertation, we have focused on software architecting domain, our abstract ontology model is based on conceptual architecture documentation meta-models IEEE 1471-2000 [47] and ISO/IEC/IEEE 42010:2011 [7]. We have followed a bottom-up

approach to develop ontologies for the TSPACE. We have analyzed Software Architecting domain using the conceptual meta-models of architecture description. We have tailored and extended the conceptual models for TSPACE by incorporating TSPACE's specific functional and *aaS model requirements.

A general TSPACE meta-model is shown in Figure 14. The meta-model shows concepts and CEs of an abstract TSPACE and selected concepts of its specialization for the software architecting domain. The meta-model is further refined while describing TSPACE reference architecture in Chapter 6. TSPACE meta-model has following main elements:

- The tools that can be provisioned by TSPACE.
- Ontologies that are used for the provisioning of a TSPACE instance and management of the artifacts during the instance's lifecycle. The ontologies are complemented by information discovery and correspondence rules. These rules are used to raise awareness among the stakeholders working on related activities and tasks in a TSPACE instance.
- Different types of artifacts that are generated and maintained by the tools associated with a specific instance of a TSPACE. The activities and tasks that are performed by the stakeholders on the artifacts as part of software architecting process are captures with the help of *ChaOnt* and *AnnOnt*.

The meta-model presented in Figure 14 also shows details of different elements of TSPACE and relationship among them. Activities and tasks that can be performed on the artifacts depend upon the number of tools bundled together in a specific TSPACE instance as per the requirements of software architecting process. Each activity or task can have multiple artifacts associated with it that are produced and maintained by the tools. The relationship between the artifacts and corresponding activities and tasks is established using APIs of TSPACE while the artifacts are being maintained inside the tools (to be described in Chapter 6). The activities and tasks are part of the process. A process can consist of multiple activities and an activity can consist of multiple tasks.

TSPACE provides semantic integration for two types of artifacts.

- ***Type 1:*** The artifacts that are maintained by the tools in proprietary data structures (for example, using database tables). These artifacts cannot exist outside the scope of the tools; hence additional measures are required if these are to be made available outside the tools.

- ***Type 2:*** The artifacts that can be exported or imported by the tools as stand-alone entities.

Type 2 artifacts can be retrieved from the tools and can be stored by TSPACE as stand-alone entities, whereas Type 1 artifacts can only be accessed via the APIs of the tools or via plug-ins/add-ins. TSPACE can contain a number of artifacts that are produced or consumed in a specific TSPACE instance and may consist of multiple attributes. Each attribute associated with the artifacts is an atomic entity that cannot be subdivided. The artifacts can be part of different abstractions. TSPACE meta-model shown in Figure 14 also shows TSPACE instantiation for software architecting activities. In software architecting TSPACE, the artifacts are part of different views. The views correspond to different viewpoints [7]. The detail of hierarchy and different types of relationship among the artifacts along with details of attributes is elaborated in Section 5.3.2. As it is visible in Figure 14, an instantiation of a TSPACE for a specific domain may require additional concepts such as architecture viewpoints and architecture views as in the case of software architecting TSPACE. Hence, the proposed TSPACE meta-model provides flexibility to incorporate additional concepts by supporting dynamic composition and aggregation of different concepts in a TSPACE.

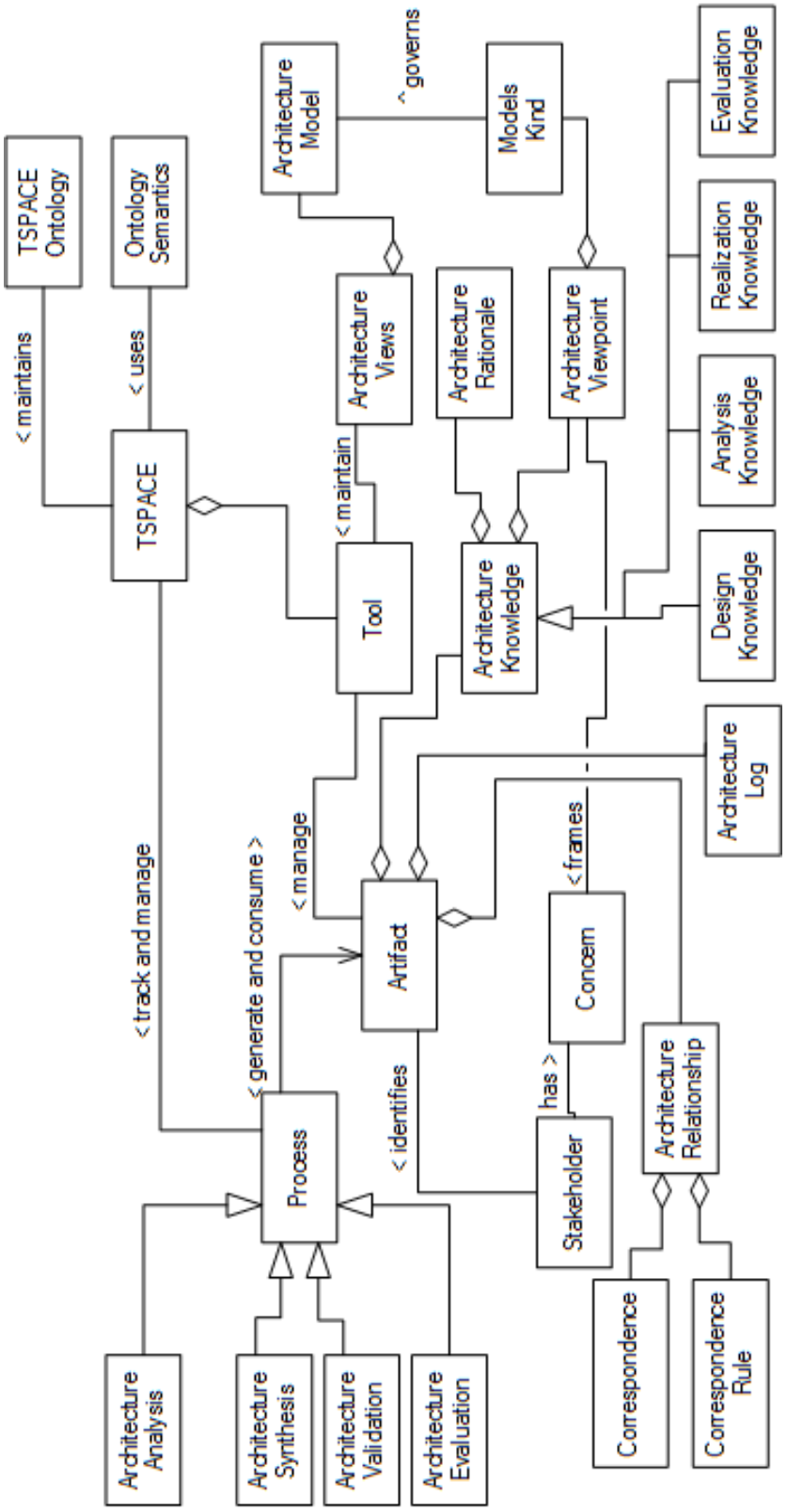


Figure 14: TSPACE Elements and Relationships Meta-model for Software Architecting

5.3. TSPACE Ontologies' Details

This section describes the details of the proposed TSPACE ontologies and elaborates the context in which the ontologies can be used. We also describe the algorithms that are proposed to complement the ontologies and to raise awareness of the activities and operations that are performed on the artifacts by the users in a TSPACE instance. An activity may be performed by using several tools, whose selection depends upon a number of factors including project and organizational requirements. It is vital to establish semantic relations between artifacts consumed or created by different tools to support users performing different tasks associated with an activity using multiple tools. For example, software architecture design and documentation activity requires the use of tools to document and design different aspects of the software architecture such as documentation of architecture design decisions [5], tradeoffs made during the design, architecture patterns and styles [4] that are chosen to implement the design decisions and models of the architecture using different views [10]. It is important to have a consolidated view of different activities carried out using different tools. The activities, tasks and artifacts should be linked in a TSPACE instance for establishing and maintaining relations among them.

Some important aspects of the activities and the processes must be considered while defining ontologies and annotation. Artifacts and process reuse, and management should be treated as a process, not as an event [114]. There is also a need to record and track actions and events throughout a software engineering process [114]. Process, task and product knowledge are considered key elements to reuse system design [115]. An ontology to support artifacts and information (knowledge) discovery should track different activities performed and should support on demand information discovery according to desired parameters. Structuring information at different levels of abstraction using ontology concepts and relationships between them using ontology annotation is also an important factor to consider [116]. It facilitates information discovery and analysis. In the following subsections, we describe in details of 4-tuple elements of TspaceOnt.

5.3.1. Capability Ontology (CapOnt)

The capability ontology captures the capabilities of individual tools and users' (stakeholders') requirements of a specific TSPACE instance. Attributes of the capability ontology are presented in Figure 15. The capability ontology provides a map between the stakeholders' requirements of the features required from a TSPACE instance and the features supported by the tools available for provisioning by TSPACE. If an exact match cannot be found, the capability ontology can be used to provide the closest match to the desired

requirements and provision TSPACE accordingly. Capability ontology corresponding to each tool consists of two constituents. Functional capability captures activities, tasks and artifact types supported by the tools or required by the users. Non-functional capability deals with quality requirements and deployment preferences of the tool. Roots of functional and non-functional capabilities are associated with TSPACE via a *capableOf* relationship. The members of functional and non-functional capabilities are associated with root elements with the *support* relationship.

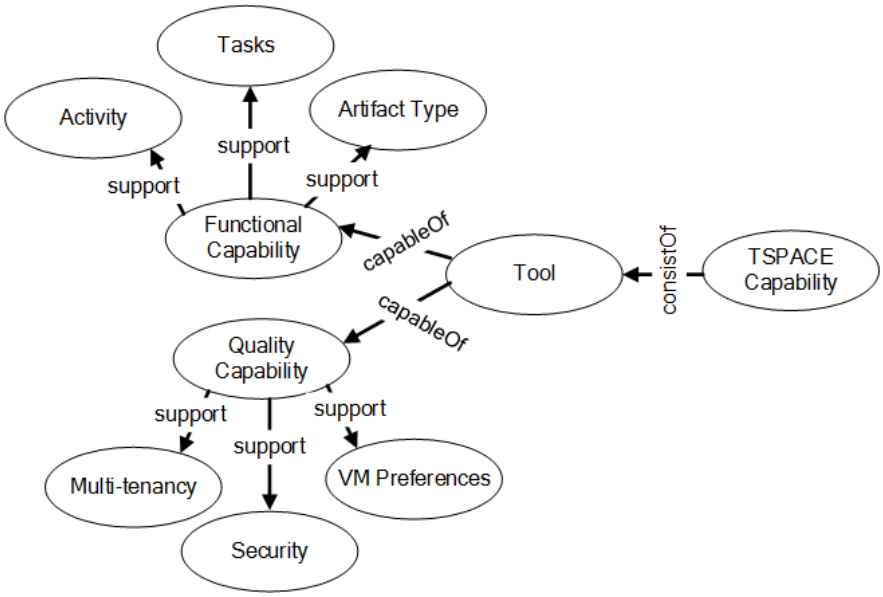
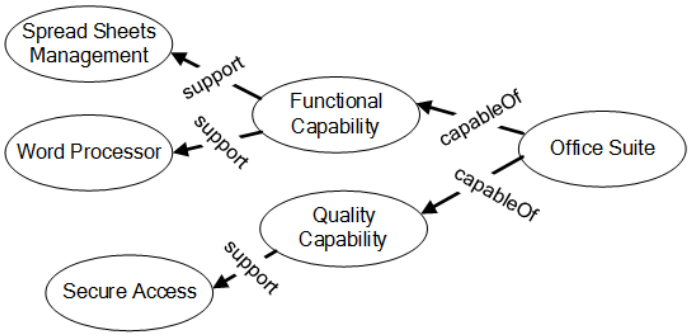


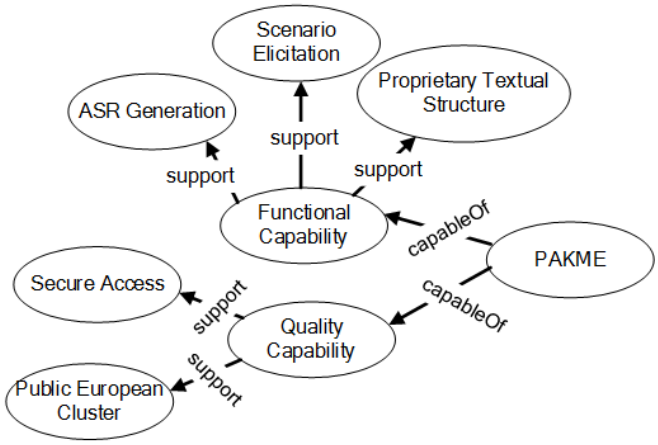
Figure 15: Capability Ontology Structure

Figure 15 shows a meta-structure of the capability ontology. TSPACE consists of multiple tools that are available for provisioning. Each tool has the capability to provide a number of features (e.g. support for specific types of activities and tasks, such as architecture documentation and providing support for certain types of artifacts such as Unified Modeling Language diagrams) that is represented as *Functional Capability* and capable of providing a number of runtime quality attributes (such as secured access, support for multi-tenancy and location specific enactment) that is represented as *Quality Capability*.

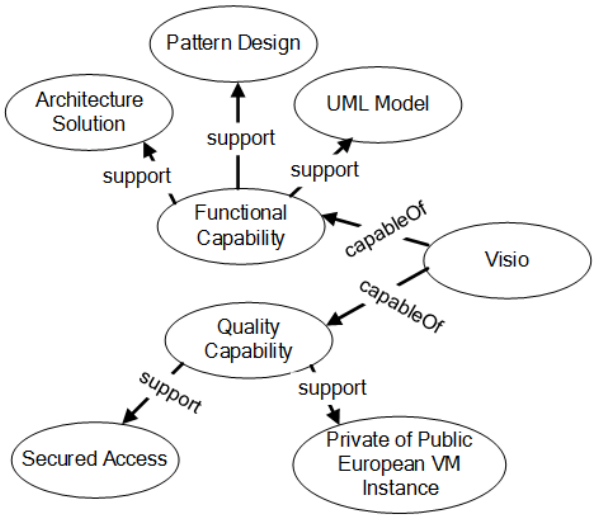
Figure 16 (a), Figure 16 (b) and Figure 16 (c) show capabilities of three examples of tools used for commonly performed software architecting activities, i.e. architecture documentation (word processing tools and spreadsheet), architecture knowledge management tools (PAKME [107]) and architecture modeling tools (Microsoft Visio). In the diagrams, only some of the functional and quality capabilities are presented.



(a) Text Processing Tool



(b) Architecture Knowledge Management Tool



(c) Architecture Modeling Tool

Figure 16: Capability Ontology Examples

Individual capabilities of the tools are combined to formulate the aggregated capability of TSPACE, as shown in Figure 17. The aggregated capability

ontology shows the overall capability of the tools (including the features and quality characteristics) that can be provisioned in a specific TSPACE instance. In the diagrams, we have only shown one tool of each kind. However, there can be multiple tools of the same type that support different features and can operate under desired runtime quality parameters (non functional requirement).

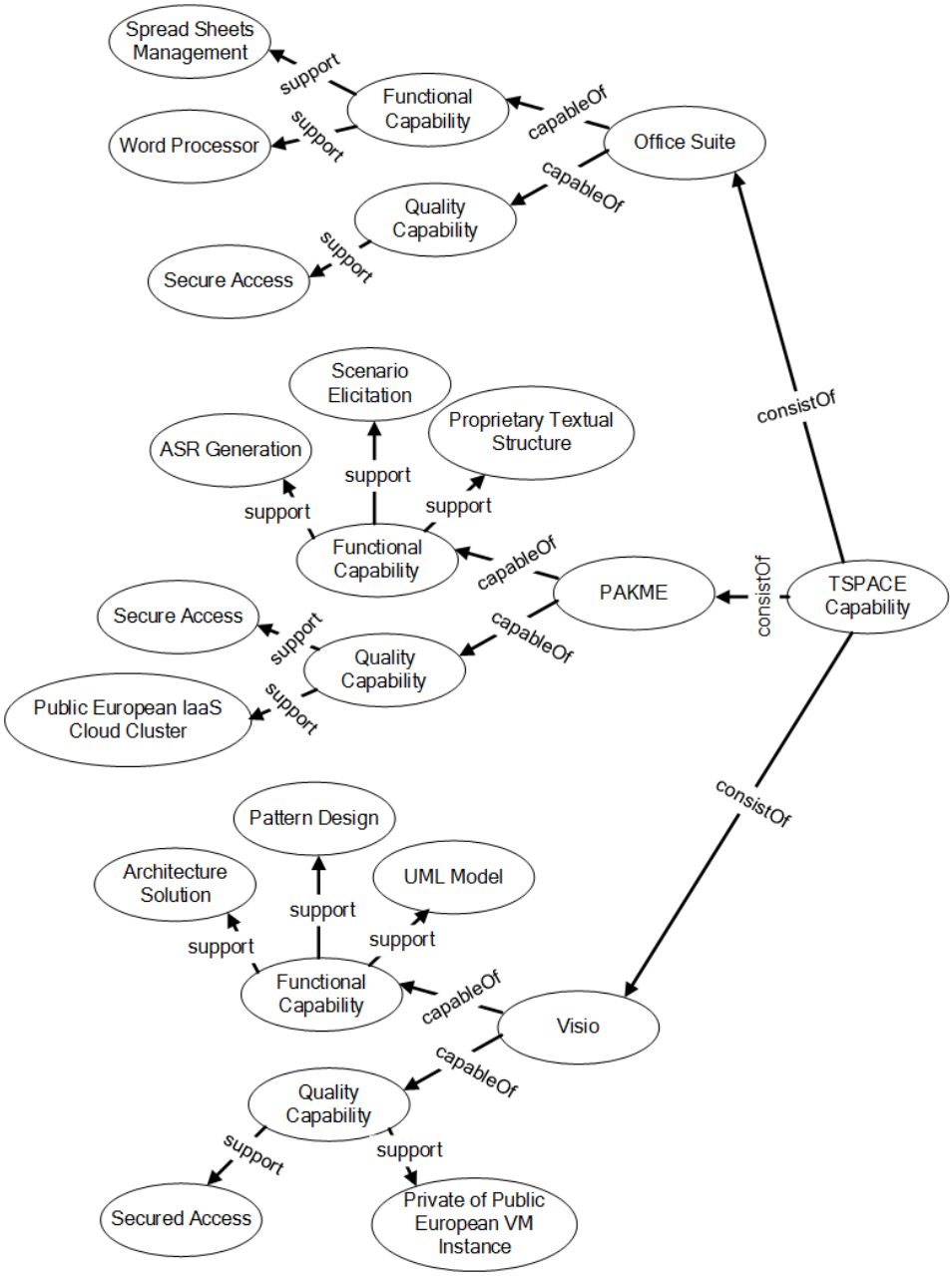


Figure 17: Aggregated Capability Ontology

The capability ontology structure presented in Figure 15 can also be used to seek input of the users required capabilities in a TSPACE instance. Figure 18 shows an example of an end user's (stakeholder's) requirement of a TSPACE instance. The aggregated capability ontology is used to find out the match of the tools available for provisioning with the required tools using the corresponding capability ontologies.

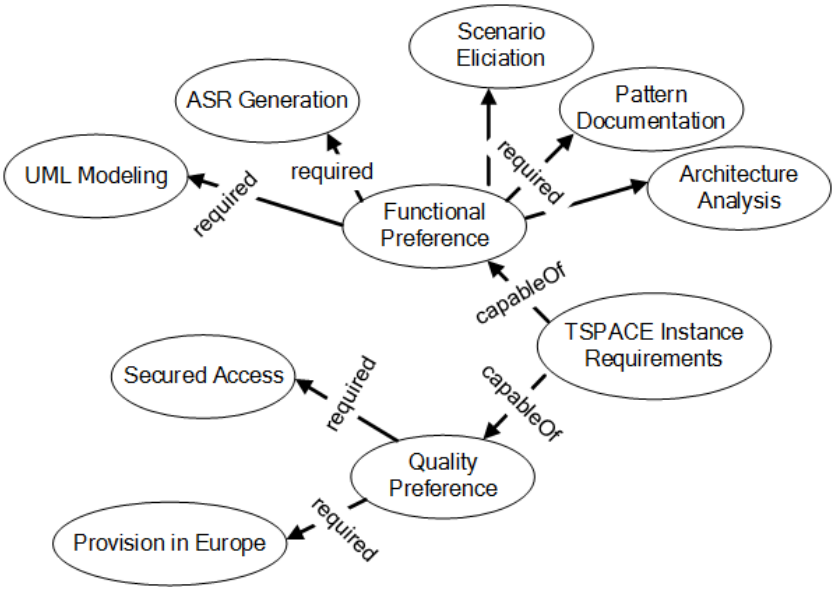


Figure 18: Capability Ontology for Tools Selection

The approach for matching stakeholders' tools requirements with the tools available for provisioning is described in Algorithm 1. The match is established by taking intersection of the required capabilities with tools' (that are available for provisioning) supported capabilities. Capability ontology can be used to find tools match for two categories of tools. (a) The tools that are enacted and provisioned by TSPACE as part of a TSPACE instance on a public or private IaaS cloud and (b) the tools that are enacted by third party tool providers and are integrated with a TSPACE instance by providing support for data integration using a TSPACE semantic model that is based on the ontologies.

Algorithm_ToolsSelection

matching_Tools_List ← null

i ← TSPACE aggregated ontology
j ← TSPACE required capability ontology
for each *k* ← tool available for provisioning registered in *i*
    l ← set of functional capability of tool *k*
    m ← set of functional capability specified in *j*
    if *l* ∩ *m* is not null then
        add *k* in *matching_Tools_List*

    n ← set of quality capability (non-functional features) of tool *k*
    o ← set of quality capability (non-functional features) specified in *j*
    if *n* ∩ *o* is not null then
        add *k* in *matching_Tools_List*

return *matching_Tools_List*

Algorithm 1: Tools Selection using Capability Ontology

To rank the tools according to their suitability with a desired capability of the tools, Analytical Hierarchical Process (AHP) [117] is applied as shown in following formula.

$$\text{Rank Score of Tool}_i = \sum_{j=1}^{N} W_{ij} * \text{Property}_{ij}$$

Property_i represents a set of features or a quality that a Tool *i* supports. Value of *j* ranges from 1 to N representing indexes of the properties set (Property_i). Value at index *j* of Property_i set is 1 if a feature corresponding to index *i* is supported by the tool, otherwise its value is zero. W_i is a set of weights for a Tool *i*. Weight value at index *i* of set W_i can be one of 0, 3, 5, 7 and 9 where zero indicates not important and nine indicates very important. Tools providing a closer match have a higher rank score.

Registering the tools with the platform can be a challenging task because of the possibility to provision a large number of locally enacted and third party provided tools. Manual registration of the tools with TSPACE may not be feasible to offer it as a service model, especially when the third-party *aaS model tools are to be integrated with TSPACE. The capability ontology of each tool (functional/quality feature set) can be populated manually by looking into features and quality characteristics that are supported by the tool or with the help of an automated crawler using term frequency and inverse document frequency technique (TF/IDF) [109].

5.3.2. Ontologies to manage Relations among Artifacts and Relations among Artifacts and Tools (ArtToolOnt)

The ontologies to manage the tools and the artifacts formally describe the semantic model of tools and artifacts in a TSPACE by defining possible types of TSAPCE elements (TE), content elements (CE) and relation elements (RE). TEs describe the concepts associated with TSPACE, tools that constitute a TSPACE, and activities and tasks that are performed using tools. CEs describe the concepts that determine elements of artifacts' logical structure with respect to different types of the tools used in a TSPACE instance. REs describe relationships among TEs and CEs in a logical structure.

Figure 19 represents an abstract description of the TSPACE ontologies and shows the relationship among the main constituents of TSPACE including activities, sub tasks within the activities, artifacts that are associated with the activities, different parts of the artifacts and relationships among the artifacts. In the diagram, the dark nodes represent TEs and the light nodes represent CEs. Aggregation Content (AC) and Aggregation Item (AI) are two core TSPACE elements. AC is root node of the TSPACE ontology. AC defines the logical structure of the elements of the TSPACE (e.g. architecture design space) and establishes a relationship between AC and different instances of AI with a *contain* property. AC represents a common root of TSPACE that all instances of TSPACE belong to, whereas AI represents a specific TSPACE instance. Content Unit (CU) is a representation of a specific process that encompasses multiple activities that are to be performed within that process, e.g. software architecture design process or software architecture evaluation process. Each activity can consists of a number of tasks, and each task can involve users working on at least one artifact. The artifacts are organized in a hierarchy according to their specialized type and are linked with the root artifact element. The artifacts can be related with other artifacts. Each artifact has at least two elements associated with it: a unique identifier that identifies an artifact in TSAPCE and contents of the artifact. The artifact contents can have multiple sub attributes. Description of the contents of the artifacts includes artifacts contents and structure, e.g. in the case of a textual artifact it contains its textual contents, and in case of a diagram e.g. UML class diagrams it can contain ontologies generated from UML or XMI of the corresponding UML diagram. If the artifact has a metric used for its description, measurement for the metric and its measurement unit, it can also be specified using *ArtToolOnt*. Depending upon the nature of the artifact, the contents of the artifacts can have additional attributes associated with them. The relationships that can exist between different elements of TSPACE are

listed in Table 32 and are explained in remainder of this section with the help of *ArtToolOnt* in software architecting domain.

The tools that are available for provisioning in TSPACE have capabilities, as described in Section 5.3.1, and can be based on different paradigms (e.g. desktop-based stand alone tools, web based tools or the tools built using service-based principles in which different components of the tools can be dynamically composed and provisioned). The tools have associated virtual machines that can be used to provision the tools. Semantic integration among the artifacts in a TSPACE is also required. Figure 19 shows common concepts and interaction among the concepts in TSPACE. As per the requirements of a specific domain, there can be more concepts added in *ArtToolOnt*. For example, in software architecting domain, architecture views and architecture viewpoints are used [5] and Representation Class concept that is shown in Figure 19 has two specializations including Architecture Views and Architecture Viewpoints.

The abstract ontologies and the relationship between different elements are explained with the help of TSAPCE ontology instance for software architecting domain and are shown in Figure 20. Containment relationship between different types of elements of TSPACE and the tools that contain the elements is established via *maintainedBy* property. Specializations of tools are represented via *specializationOf* property. Aggregation and specialization relationships between TEs enable structuring of content elements in the form of tree structures. The relationships also enable the establishment of a link between content elements according to a given criterion. AI maintains a reference to ContentUnits (CU) of a TSPACE via *consistOf* property. A CU is a container for multiple activities that are performed in a TSPACE. An activity may consist of multiple tasks. In Figure 20, *Modeling*, *Knowledge*, and *Requirements and Scenarios* are examples of CUs. The relationship of an activity or a task with CE is captured by the *contain* property.

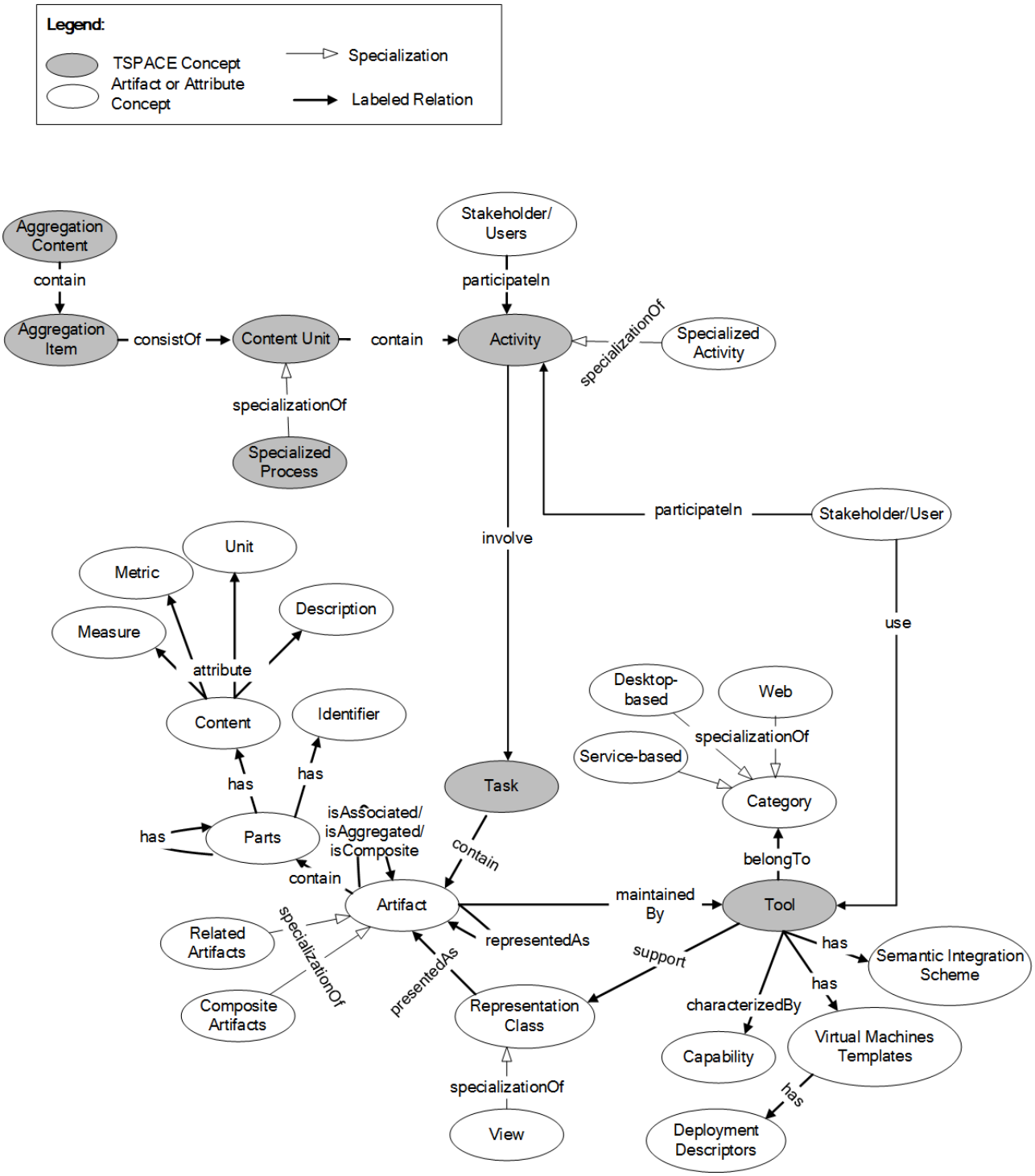


Figure 19: TSPACE Abstract Tool and Artifact Ontology

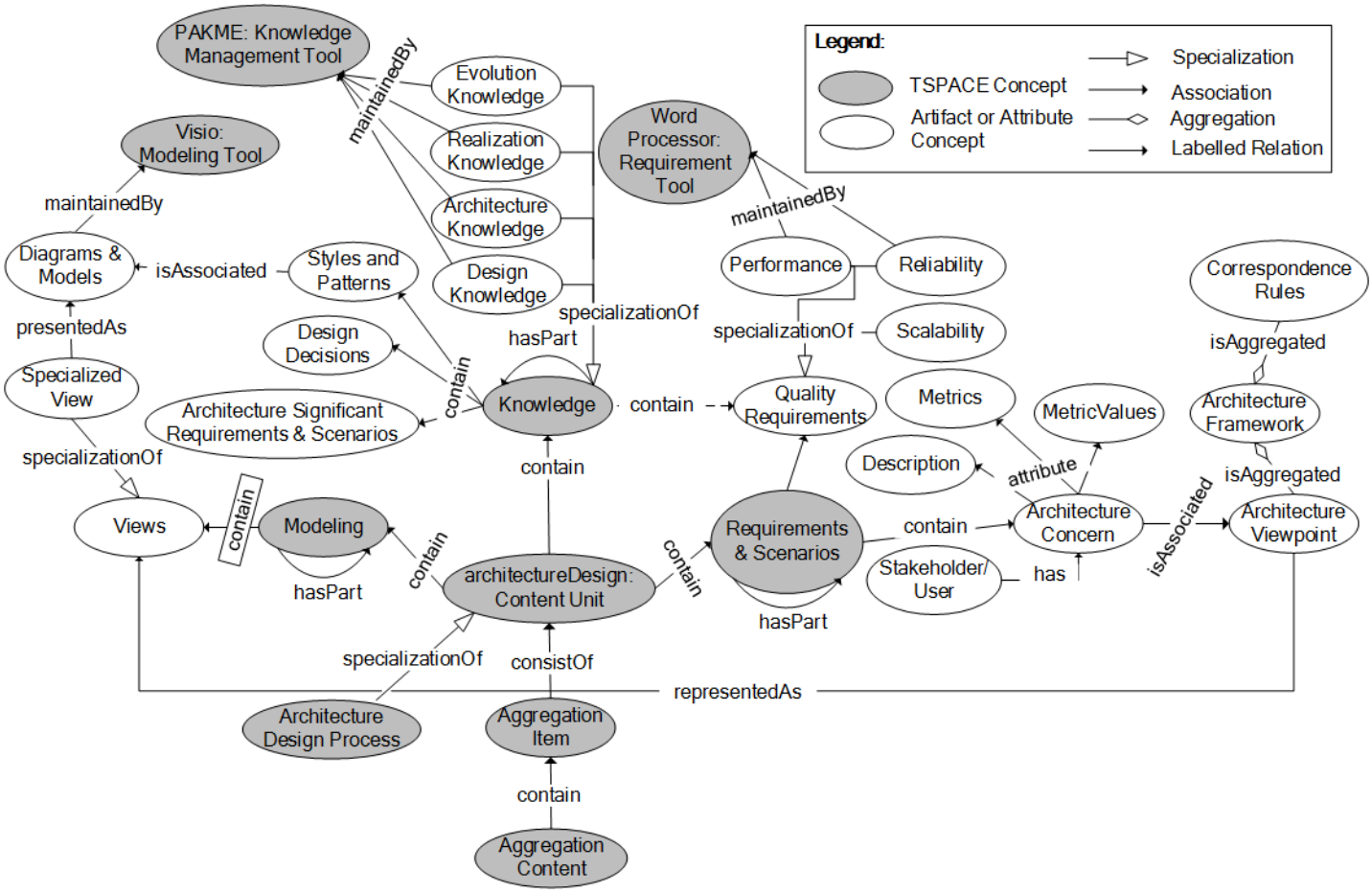


Figure 20: TSPACE Tool and Artifact Ontology Instance Example

Each CE describes a uniquely identifiable resource in a TSPACE instance. The resources represent elements of TSPACE. These can be extracted from a specific TSPACE instance and can be reused in other TSPACE instances. In Figure 19, *Artifact*, its specializations and its sub-concepts are the examples of CEs. Figure 20 shows an instance of CEs with references to architecture design of the TSPACE. In Figure 20, each of the sub-concepts represent content elements at high level of abstraction with reference to architecture design of the TSPACE and consists of multiple sub-elements. The relationships between CU, main concepts and sub-concepts are established with the *contain* property. A nested relationship among the main concepts of the same type is established with the *hasPart* property. The relationship between abstract concepts and their specializations is established with *specializationOf* property.

The details of relations and specializations of CEs with reference to software architecture design of the TSPACE are shown in Figure 20. *Requirements and Scenarios* contains two primary sub-concepts: quality attributes and

architecture concerns. Quality attributes may have many specializations. The specialization of a quality attribute is represented via *specializationOf* relationship. Nested specializations are also possible and can be represented via *specializationOf* relationship. Architecture concerns consist of description, metrics and metrics' values. Architecture concerns are framed by architecture viewpoint and are represented by *framedBy* relationship. An architecture framework aggregates architecture viewpoints. The aggregation relationship of architecture framework with architecture viewpoint and correspondence rules is represented via *isAggregated* relationship. Architecture knowledge contains architecture significant requirements and scenarios, quality attributes, design decisions, and styles and patterns. Architecture knowledge can be classified into four specializations: design knowledge, architecture knowledge, realization knowledge and evolution knowledge. The specialization is represented as *specializationOf* property. There can be more specializations of architecture knowledge, though they are not depicted in the figures. An architecture is modeled using different views and is a representation of different viewpoints. This relation is depicted as *representedAs* property. Views can be further specialized as process view, logical view, physical view, deployment view and scenarios, as depicted in 4+1 view model [51]. We only represent the details of scenarios that are indirectly linked via architecture viewpoints. Every specialized view can be presented with one or more diagrams, and this relationship is represented via *presentedAs* property. A general association is possible among CEs or different elements of CEs, such as association of architecture styles and patterns with models and diagrams. This type of general association is represented by *associatedWith* property. The high-level relationships of SpaT ontologies discussed above are listed in Table 32.

Table 32: TSPACE Relations to manage the Tools and Artifacts

Relationship	Description
hasPart	Relationship between a child and parent content unit (CU) such that only one of the children CU of its type can exist.
consistOf	Relationship between a parents CU and a child CU.
contain	Containment relationship between parent content element (CE) and child CE such that parents and child are at different levels of abstraction.
containedBy	Containment relationship between child CE and parent CE. It is inverse of the *contain* relation.
specializationOf	Specialization of a generalized CE into a specialized CE.
has	Association between an actor (stakeholder) and a

	CE.
framedBy	Containment relationship between a child CE and a parent CE such that the parent CE consists of one or more child CE and the parent CE is not valid until it has all of its children CEs.
isAssociated	Association relationship between two CEs that are at same level of abstraction.
isAggregated	Aggregation (composition) relationship between two CEs such that the one being aggregated can exist without the aggregator.
isComposite	Composition relationship between two CEs such that the one being composed cannot exist without the composer.
presentedAs	Diagrammatic or textual representation of a CE by another CE such that both are of different forms. E.g., one in textual and other one in diagrammatic form.
representedAs	Representation of one type of CE with another type of CE with same form. E.g., using textual description.
attribute	An attribute of a CE that represents its property. For example, if a non-functional requirement asks for 95% availability, value 95 is an attribute of the availability requirement.
support	A particular view that is supported by the respective tool e.g. a scenario view or a use case view.

5.3.3. Change Ontology (ChaOnt)

The change ontology tracks changes in the TSPACE's content elements (CE) and relationship between CEs. We extended the change ontology of the semantic document model reported in [113] for the elements of TSPACE. Pictorial representation of the root-level change ontology is presented in Figure 21. The change ontology consists of three main concepts: AggregationContentChange, AggregationItemChange and ContentUnitChange corresponding to *AggregationContext*, *AggregationItem* and *ContentUnit* respectively. Every change creates a new version of the content element. Both old and new versions of the changed content elements are stored, and *oldVersion* and *newVersion* properties are used to capture the changes in CEs. The properties are also used to link the old and new versions of the content elements. The changes in the content elements are determined by comparing old and new versions of the elements.

In order to capture modifications in a CE, *addedUnit* and *updatedUnit* properties are used. Addition of a new content element in the TSPACE is captured by *addedUnit* property. Any change in the contents after first time addition is captured using *updatedUnit* property. The changes that emerge in the structure of the architecture design space are managed by linking instances of rdf:Graph data structure with the changed content element. The property *hasAllChanges* links AggregationContentChanges with AggregationItemChanges. The property *referTo* links AggregationItemChanges to ContentUnitChanges.

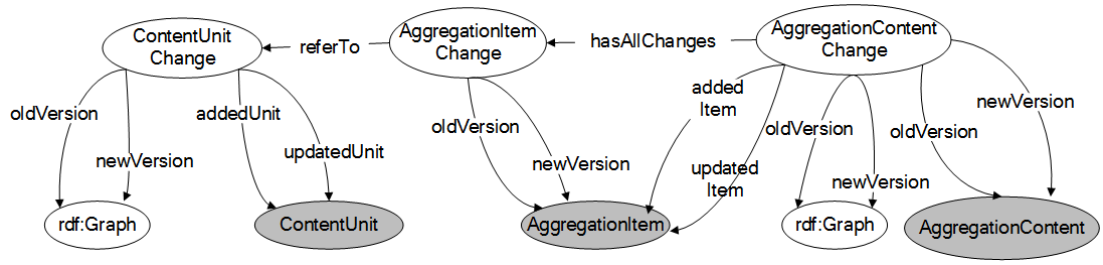


Figure 21: TSPACE Change Ontology

5.3.4. Annotation Ontology (AnnOnt)

One of the main objectives of our semantic model for TSPACE is to enable discovery and access to artifacts corresponding to the activity and to reuse of CEs. In order to enable discovery, access and reuse of artifacts and their elements in TSPACE, we have developed an annotation ontology that is presented in Figure 22. We extended the annotation ontology of the semantic document model reported in [113] for the TSPACE.

Our approach to enabling TaaS leverages annotation ontology, for semantic integration among artifacts maintained by different type of tools used in a specific instance of the TSPACE. The annotation ontology supports pulg-ins and data collection probes. The plug-ins, add-ins and probes discover CEs at different levels of abstraction with the help of the annotation ontology. By introducing the annotation ontology, we aim to provide common high-level concepts in terms of classes and provide methods for adding annotations to CEs. Both classes and properties can be evolved and extended dynamically at runtime to support multiple types of tools in the design space. The ontologies for annotation provide a mechanism to semantically relate data and artifacts. Considering CEs of TSPACE and tools, we have identified two types of annotations:

- Context independent annotations corresponding to the content elements that are independent of the artifacts and the tools.

- Context dependent annotations corresponding to the content units that are part of artifacts and the tool that is maintaining the artifact.

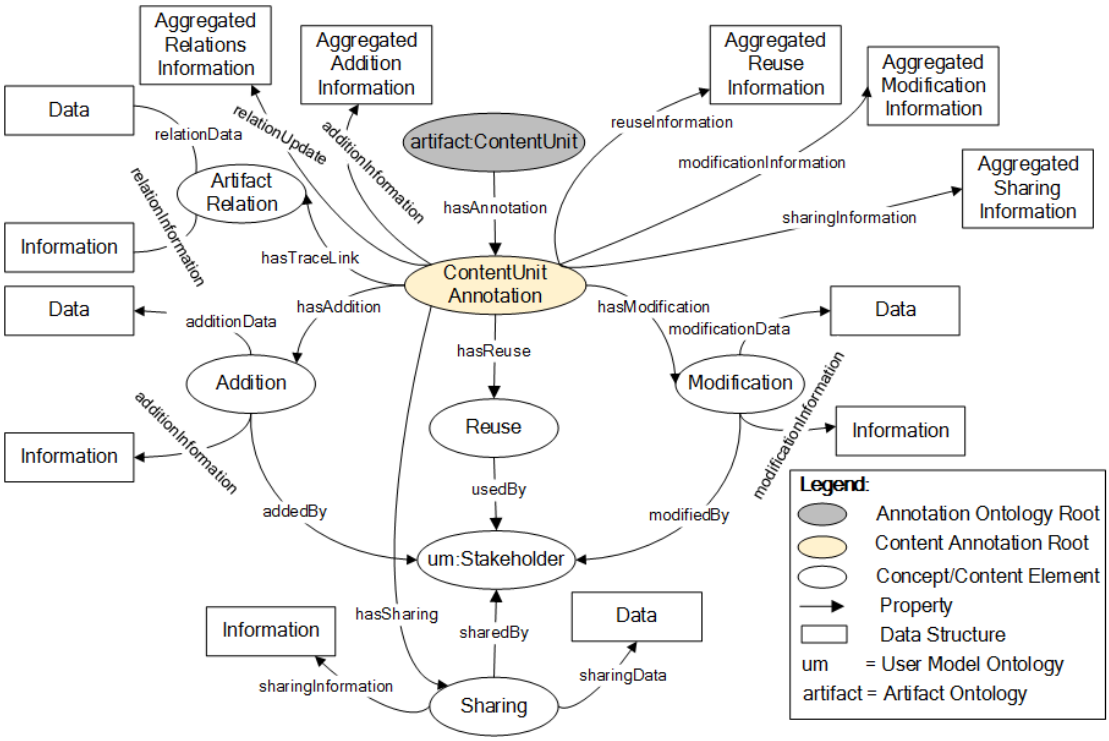


Figure 22: TSPACE Annotation Ontology

5.3.4.1. Context Independent Annotation

Artifacts and tools ontology discussed in Section 5.3.2 complements the annotation ontology, which relates context-free annotations to the instances of content elements. ContentUnitAnnotation and hasAnnotation rules are introduced in the ontology to bind metadata to content elements. The semantic document model (SDM) to enhance data and knowledge interoperability for text documents [113] has identified three categories of context independent annotations (for annotating the data and tracing the activities that are performed on the data): standardized metadata, usage metadata and subject metadata. Standardization depends upon the specific domain in which the TSPACE is to be used; hence, in this chapter we are only dealing with usage metadata and subject specific metadata.

The *Usage meta-data* tracks operations on the artifact's content elements in the TSPACE. One of the goals with TSPACE is to provide a customizable and semantically integrated suite of tools by bundling multiple tools together [38]. To support tools bundling, it is important to capture information about interaction of stakeholders (users) with artifacts in terms of data and

operations that are performed on the data. To achieve this objective, we have extended our proposed ontology with a set of concepts and properties. There are four main concepts and two properties associated with each concept. Main concepts are *Addition*, *Modification*, *Sharing*, *Reuse* and *TraceLink*. The concepts are associated with the type of interaction, the stakeholder who is participating in the interaction and the trace links among artifacts and data that are affected as a result of interaction. *Addition* concept represents addition of new artifacts and data associated with the activity (e.g., architecture design activity). *Modification*, *Reuse* and *Sharing* concepts track information of interactive activities and applications through which stakeholders are performing the activities. Every time a stakeholder interacts with a CE associated with the artifacts, the metadata is added to the CE. With the help of *TraceLink* concept, the metadata is used to determine how the CEs are linked to each other. Each concept has two properties: *data* and *information*. *ContentUnitAnnotation* also has five properties corresponding to aggregated information of each concept. Aggregated information is maintained for a consolidated view of stakeholders' activities on the artifacts that are being used as part of an activity (e.g., software architecture design). Figure 22 shows a pictorial representation of annotation ontology and usage metadata.

The *subject specific meta-data* is an ontological metadata that conceptualizes the subjects that are described by content elements. It is conceptualization of knowledge modeled by content elements [113, 118]. It is represented by @* property associated with content element in the proposed annotation ontology. The goal of our proposed ontology is to facilitate processing of human readable artifacts and data through machine with help of plug-ins and probes, and the proposed semantically integrated TSPACE.

The *standardization meta-data* is used to describe the identifiable resource. Vocabularies standards such as Dublin Core [119] and IEEE Learning Object Metadata (LOM) [120] are designed to describe identifiable resources [113]. The models are well suited for a document with simple structure but are not fully compliant with artifacts having complex structure. We have selected a tailored subset of meta-data to make it fit for architecture design workspace. Following are the elements of the tailored metadata that is incorporated in annotation ontology.

- @creator refers to creators or authors (stakeholders) of the artifacts.
- @datetime corresponds to creation or modification date and time of the artifact.
- @format refers to the format or standardization that is used to describe the artifact. It can contain reference to the standard, for example, an XMI representation of UML documents. For the artifacts that are maintained by the tools using proprietary standards or maintained by the tools using

complex data structures, corresponding information is specified against the format element.

- @representation refers to whether the artifact is represented using textual description, xml format or binary format.
- @language refers to the language that describes the representation. This property holds values only if the artifacts are described using natural language or XML structure. If the artifact is represented in natural language then an identifier for natural language (e.g., ENG for English) is specified. If the artifact is represented using XML then mapping between different elements of the XML structure and contents of the artifacts is specified.
- @identifier represents the unique identifier of the artifact or data corresponding to the artifact.
- @PLACEHOLDER[VALUE] represents annotation for a specific portion of an artifact. PLACEHOLDER represents type of annotation whereas VALUE represents value of the annotation. E.g. parts of the architecture significant requirement using @PLACEHOLDER[VALUE] can be annotated as "*Application should be @quality[available] @value[90%] of a @metricunit[day]*".

5.3.4.2. Context Dependent Annotation

This annotation is a representation of the content elements when these are parts of a specific TSPACE. The annotation ontology relates the context dependent annotations to AggregationItem (Figure 19) concept defined by the TSPACE ontology. We introduced the DesignSpaceAnnotation concept in our annotation ontology that acts as a metadata binder for AggregationItem (Figure 22). In order to facilitate binding operations, we introduce two new concepts in the annotation ontology: (i) SemanticElement to extract the relationship between sub concepts of AggregationItem according to defined queries and procedures. (ii) TraceElement to identify dependencies and trace links between content elements of the TSPACE.

5.4. Use of Ontologies for Notification and Quality of Software Architecting Activities

Annotation ontology along with change ontology also support notifications for collaborative activities that are being performed using multiple tools in a TSPACE. The rules use elements of annotation and change ontology to raise awareness about the activities performed using the tools and to send notifications across the tools corresponding to actions associated with the activities. Let x and y be content elements of the artifacts that are produced in the TSPACE, $Relation_{x,y}$ be a relationship that exists between x and y (e.g., $Parent_{x,y}$ represents x is parents of y), $<Action>_x$ is an action triggered for x,

and <Select><U>$_x$ as selection of a content element to be used in a particular activity by a user U. Table 33 shows sample notifications rules corresponding to the addition, modification, conflict identification and sharing of the content elements. These rules can be implemented using SPARQL [121] queries in combination with complimentary algorithms.

Table 33: Sample Rules for TSPACE Notifications

Notification	Formation
R1: Addition	$\forall_{x,y}$: <NotifyAddition>$_y$ $\Longrightarrow$ Parent$_{x,y}$ $\wedge$ <Addition>$_x$
R2: Modification	$\forall_{x,y}$: <NotifyUpdate>$_y$ $\Longrightarrow$ Parent$_{x,y}$ $\wedge$ <Modification>$_x$
R3: Sharing	$\forall_x$: <NotifySharing>$_x$ $\Longrightarrow$ <Select><User1>$_x$ $\wedge$ <Select><User2>$_x$
R4: Conflict	$\forall_{x,y}$: <NotifyConflict>$_y$ $\Longrightarrow$ Parent$_{z,x}$ $\wedge$ Parent$_{z,y}$ $\wedge$ <Modification>$_x$ $\rightarrow$ <Conflict>$_y$

Algorithm 2 is used to fire addition and modification notification, when a content element of any of the parent of the content element that is under investigation is modified or additional attributes are added to it.

```
Algorithm_Notification (TSPACE lookupContentElement)
    user_notification_List ← null

    i ← TSPACE instance aggregated RDF
    for each j ← ancestor of lookupContentElements in i
        if NotificationAddition(j) OR NotificationUpdate(j) is true Then
            User u ← getUser(k)
            append(user_notification_List, u, NotificationType(j))
        end if
    end for

    Fire_Notifications(user_notification_List)
```

Algorithm 2: Tools Selection using Capability Ontology

5.5. Discussion

The ontologies that have been presented in this chapter are collectively used to provide semantic integration among the artifacts of different types and different abstraction levels. In this section, we present a sample scenario that depicts the process of instantiation of the ontology structure when tools are

provisioned by TSPACE. After a specific set of tools is selected and TSPACE provisions the tools, the baseline for semantic integration is configured using artifacts and tools ontology (Section 5.3.2). E.g. if an architecture significant requirements documentation tool, an architecture knowledge management tool and an architecture modeling tool are provisioned by TSPACE, separate content units (CU) for each of these tools are created (as depicted in Figure 20). Hence, TSPACE root CU has three child CUs: one for architecture significant requirements tool, one for knowledge tool and one for modeling tool. In case multiple tools are provisioned to perform specific activities, nested CUs are possible. Once the data of the artifacts is posted on TSPACE, the data is added under the corresponding CU. E.g. data of the artifacts corresponding to architecture significant requirements tool is added under architecture significant requirements documentation CU. In this manner, data inside CU of each tool is populated. Annotation ontologies are used to relate an artifacts or parts of an artifact that is produced from one tool with artifacts of other tools. In ontologies, it is reflected as a relation of the artifact managed within one CU with another artifact in another CU. E.g. if an architecture significant requirement artifact is related with a design decision corresponding to that requirement in a knowledge management tool, the relation of the artifact managed under the requirements CU is established with an artifact managed under the knowledge CU, and annotations from the annotation ontologies (Section 5.3.4) are used to establish the nature of the relationship. The relations to manage artifacts (Table 32) are used by annotation ontology. The changes on these artifacts that occur as results of the operations that are performed on the artifacts are recorded and managed with the help of change ontology (Section 5.3.3). The detail on the ontologies implementation is elaborated in Chapter 6 using TSPACE reference architecture and a prototype of the reference architecture.

TSPACE can consist of two types of the tools: (a) the tools that are enacted as part of TSPACE instance and (b) third-party cloud-based tools that are available on the cloud and are integrated with TSPACE using its APIs. Although in this dissertation the research is only focused on providing semantic integration among the tools that are provisioned by TSPACE, the proposed ontology model can also be used to integrate externally provisioned tools with TSPACE.

5.6. Related Work

A number of studies have reported adoption of ontology-based approaches to address specific software engineering challenges. The most comprehensive study is conducted by Zhao et al., [110], in which authors have presented a classification of ontologies and semantic web-based approaches proposed for

software engineering domain. Software process ontologies capture activities, process phases and process models. Knowledge management in design process, work process and project management, design traces, design documentation and tools are primary constituents of the software engineering ontologies [110].

Process ontologies: Ontologies are also used to model features of a specific application domain [110]. Boskovic et al. [122] have presented an ontology for configurable business processes following software product line engineering approaches. Their approach is based on Business Process Family Models (BPFMs), which consists of problem space (artifacts), mapping between problem and solution space, and solution space (business process model template). First step of the approach [122] is to identify the relationship between features of independent families. In next step, verification and validation of the relationships between target customers and developers of different families are performed. It is also verified whether the relations are specified properly. In next step, appropriate integration choices are made. In the final step, selected integrated patterns and initial feature models are transformed into integrated families. Feature models are modeled using semantic annotations. The proposed approach provides a semantic-based mechanism to compose service families, but it does not provide integration among the services with respect to business logic and operations.

Valiente et al. [123] have proposed an ontology-based approach to integrate software development and information technology service management processes and corresponding support tools. The proposed approach links the business (organizational) view to the engineering view with the aim of integrating business information early in the software development lifecycle. The integration framework consists of the workflow ontology, based on roles of the resources taking part in the business activity, the responsibilities of individual activities and the routes that specify flow of information between different business activities. This approach is primarily focused on workflow and information processing but does not deal with issues specific to tool integration and information consistency, especially when multiple tools generate information that cannot be transferred among tools and services as it is handled in a workflow-base system.

Software maintenance process ontologies provide concepts and their relationships corresponding to the software maintenance and related activities. Other than activities, this type of ontologies also contains person, procedures and resources [110]. Most significant work in this regard is by April et al. [124], in which the authors have presented a formalization of a software process ontology by combining primary processes that are used to carry out actual maintenance with support processes (documentation, version

management, verification, validation, review and audits etc.) and organizational processes (maintenance planning, measurement and analysis, innovation and deployment, process definition and improvement etc.). Their main contribution is to provide formalization of the maintenance ontology presented by Kitchenham [125].

Liao et al. [126] have provided a brief description of ontology to describe software processes. Two main components of their process ontology are organizational *processes* and *practices*. They have not described the details of a specific type of process or domain in which their model can be applied. Their approach is abstract and cannot be adopted into any real application domain.

Modeling ontologies: Software modeling ontologies model interactions and relations among architecture concepts, styles and patterns [110]. The most prominent work in this regard is proposed by Athanasiadis et al. [127], in which they have propose a technique for object to relational mapping based upon semantics. Their approach facilitates enterprise application development using OWL based formal domain specifications. At the core of the proposed approach is an ontology that maps OWL data type properties onto java entity classes. Pattern oriented ontologies describe patterns using OWL and RDF properties [110]. Patterns are defined and OWL concepts and the relation between concepts and sub concepts is established using properties (e.g. hasParticipant) [110]. Ameller and Franch [128] have presented an ontology to describe relationship between architecture view, their framework, architecture styles, variant of architecture styles and their implementation in context of a web-based application.

Artifacts' ontologies: Software artifact ontologies provide a set of concepts to classify different artifacts according to their format, internal structure and related concepts such as actors who create the artifacts and projects in which artifacts are created [110]. The work on artifacts ontologies is limited to providing support for searching relevant knowledge and reusing it. Antunes et al. [129] have presented a semantic web based approach to facilitate developers to search knowledge repository and to suggest knowledge relevant to a current task that a user is performing. The aim of their approach is to facilitate reuse of software development knowledge using semantic web. Happel et al. [130] have also presented a software reuse methodology based on ontologies. Their work focuses on facilitating reuse libraries by providing background knowledge. Semantic integration of implicit and explicit metadata facilitates deriving new facts from the existing knowledge. An object-oriented ontology provides a relationship among different concepts of object oriented programming languages and version control ontologies facilitate formalization of versioning of the software development artifacts [110].

Documentation ontologies: Documentation ontologies provide relationships between concepts of software documentation. A number of studies have been reported to formalize software documentation approaches. Witte et al. [131] have proposed a semantic web-based approach to automatically integrate source code and source code documentation by populating corresponding ontologies using code analysis and text mining. Their approach facilitates to perform maintenance tasks including traceability recovery between code and documents easily. Zhand et al. [132] have also proposed a traceability recovery approach based on ontologies. Their approach establishes a relationship between source code and corresponding source code documents at semantic level. Although the authors have described the notion of design patterns in the documentation and their relationship with code, their approach is limited to simple relationships between design documents describing different elements of the source code and source code. Decker et al. [108] have presented an overview of self-organized reuse of software engineering knowledge using semantic wikis. The authors have given examples from the requirements engineering domain to show advantages of semantic wikis in reusable software engineering knowledge.

There are a number of studies reporting use of ontologies for software architecture documentation. Boer et al. [133] have presented use of ontologies to visualize architecture design decisions. The authors have presented an ontology for architecture design decisions named "QuOnt" [134]. The presented approach establishes a relationship between quality criterion and quality attribute, and the effect of quality attribute on quality criteria. Effect can either be positive or negative. Criterion is further subdivided into subclasses according to specific requirements. They have presented the use of ontology to visualize architecture design decisions in the domain of software architecture audits. The rules and constrains to specify relations among different quality attribute and quality criterion are also presented. Tang et al. [135] have presented a lightweight ontology to establish a relationship between different elements of architecture documentation including requirements, architecture structure, architecture components and architecture design decisions. Graaf et al. [136, 137] have presented an ontology for software documentation using a semantic Wiki named ArchiMind. The main contribution of their approach is the evaluation that shows ontology-based approach is better, time efficient and more effective than document-based approach.

The studies on architecture documentation show the significance of using ontologies to structure and relate concepts involved in software architecture documentation activities. However, the studies do not address the root cause of the issue; i.e. how to provide a common place for multiple types of artifacts

maintained by heterogeneous tools that are used to perform activities associated with software architecting.

Patterns ontology: Pattern ontology provides a catalogues of design, usability and application patterns [110]. Henninger and Ashokkumar [138] have described a basic meta-model and ontology to describe the presence of different elements of a design pattern (context, forces, problems, solutions etc.), their classifications and the relationship between different classifications. However, their work does not provide ontologies to assign and evaluate each property of pattern at atomic level that is measureable and can be used for further automated analysis and reasoning on software architecture.

Cloud-based software engineering ontologies: Zhou et al. [139] have presented an approach for reengineering software for cloud-based systems using ontologies. Ontologies from enterprise application are build using reverse engineering and model transformation techniques, and a system for the cloud is reengineered using model transformation techniques. Link between original enterprise system and transformed system is established using ontology relations. Once requirements are mapped onto concepts and relationship between difference concepts is established, inconsistency is detected by applying inference rules.

Requirements analysis: Kaiya and Saeki have presented an ontology-based semantic processing approach for requirements analysis [140]. Their work is based on mapping requirements to domain ontology using transformation rules. Domain model consists of specialization of concepts and relationships.

Integration ontologies: Brandt et al. [141] have presented a flexible ontology-based schema for knowledge management and integration platform to integrate process and product information in a chemical engineering domain. They have described their ontology in four broad categories: *product area* describes the type and version history of electronic sources, *storage area* describes location and version of a particular artifact, *descriptive area* describes high-level semantics for content and role of the product objects or artifacts, and *process area* describes concepts to represent process objects. Rilling et al. [142] have described a meta-model and ontology to link documents with source code. Their meta-model captures the relationship between tools, tasks and the artifacts involved with the tasks. Text mining is used to build the relationship between the entities. However, their approach does not provide a detail of the ontology associated with tools, tasks and the artifacts. It remains vague how the information is structured and how different types of rules are applied to link document with code.

Besides specialized work on ontologies, there are also studies that have reported generalized technology ontologies for software development environments and tools [110]. Wongthongtham et al. [143] have also presented different dimensions of software engineering ontologies and have specified properties of the class diagrams and the relationship that can exists between different elements of the class diagram.

Table 34: Existing architecture design ontologies with respect to TaaS

Software Engineering Ontologies	Focus	Comparison with TaaS
Process	-Semantic mechanism to compose service families. -Workflow information processing.	-TaaS integration, artifacts relationship management, completeness, consistency is missing.
Modeling	-Objects to rational mapping. -Relation among patterns and their variants.	-Integration among different abstraction of models is not considered.
Artifacts	Artifacts indexing to facilitate search and reuse.	-Distributed artifacts are not considered.
Architecture Documentation	-Integration between source code and its documentation. -Traceability recovery. -Reusing software engineering knowledge. -Visualize architecture design decision. -Relationship between quality criteria and quality attributes. -Wiki-based architecture documentation.	-Does not address root cause of multiple artifacts being generated and consumed using multiple tools.
Pattern	Elements of patterns (intent, context, design)	-Ontology does not present patterns elements so that they are distinguishable from other patterns and can be measured.
Cloud-based	Map requirements onto concepts to	N/A

Software Engineering	determine inconsistencies.	
Requirements	Requirements consistency, completeness, ambiguity and conflicts.	N/A
Integration	-Integrate process and product. -Integrating source code with its documentation. -Ontologies for UML class diagrams.	N/A

5.7. Conclusions

In this chapter, we have presented an ontology-based framework for provisioning of the tools in TSPACE. The main contribution of our approach is development of a suite of ontologies to support tools' selection and to provide semantic integration among the tools and artifacts provisioned by TSPACE. The suite of ontologies enables building of structures of multiple content elements (CEs) of the TSPACE, and respective annotation and change ontology. Our approach emphasizes and supports the relations between content elements of TSPACE. The ontologies are used to identify all the constituents of TSPACE including CEs, relationships among CEs, attributes of CEs, labels of the relationships and the tools that constitute a TSPACE instance. The ontologies are used as a foundation to design a detailed TSPACE reference architecture that is to be presented in Chapter 6. Although in this chapter and in the dissertation we have focused on TSPACE application on software architecting domain, we foresee that the concepts can be adopted in other engineering disciplines. We also believe that leveraging formal approaches such as ontologies can facilitate the adoption of cloud-based tools in other sophisticated domains.

Chapter 6. Reference Architecture Models and Components

The role of Software Architecture (SA) is critical in developing and evolving a cloud-based workspace for hosting and provisioning TaaS. In this chapter, we present details of a Reference Architecture (RA) for designing a cloud-based TaaS workspace (TSPACE) - a platform for hosting and providing Tools as a Service. The TSPACE reference architecture has been designed by leveraging well-known architecture design principles and patterns and has been documented using views-based approach. The reference architecture has been presented in terms of its context, goals and design elements with respect to the requirements (as described in Chapter 4), design tactics and different components of the reference architecture at multiple levels of abstraction. We also report evaluation of the reference architecture for different functional and non-functional requirements for hosting and integrating tools using IaaS cloud. The reference architecture can provide valuable guidance and insights for designing and implementing concrete software architectures of TSPACE.

Parts of this chapter have been presented in [1, 38, 49, 52].

6.1. Introduction

Software Engineering (SE) needs to be supported by several tools to perform different activities such as Requirements Engineering (RE), Software Architecture (SA) design and testing. Traditionally SE tools (i.e. individual tools or integrated environments) are either deployed on individual desktops or provisioned from a centralized server via Web browsers. The desktop and web-based provisioning of SE tools make it quite difficult (or sometimes impossible) to easily and freely share tools, data, and artifacts across projects, teams or organizations. With the increasing adoption of Cloud Computing as a flexible and reliable technology for acquiring and releasing Information Communication Technology (ICT) Infrastructure for real and perceived benefits as reported in [20, 144], several commercial and research efforts are focused on provisioning of cloud-based or cloud-hosted Tools as a Service (TaaS) such as cloud-based IDE Cloud9 [145], online diagramming tool Gliffy [146], and several other efforts reported in [38, 147].

Whilst Cloud Computing provides a viable and flexible technological infrastructure to provision SE tools, building and leveraging cloud-based platform for providing tools as a service (TaaS) presents several unique challenges that need appropriate architectural support [38]. For example, a

user (or a team) should be able to bundle and acquire a diverse set of tools that can interoperate so that the user(s) working on a SE activity can have access to different artifacts and data, even those artifacts and data are maintained within different tools in non-standardized or proprietary formats [38]. Moreover, users also need to be aware of the activities and actions that are being performed on the data using multiple tools [38]. Provisioning of SE tools from an Infrastructure as a Service (IaaS) cloud according to specific project and organizational constraints is also a vital requirement [52]. To address these challenges, there is a need for a reference architecture of TSPACE. TSPACE reference architecture is *architecture of an aggregated platform that facilitates activity or task specific tools selection and provisioning, provides integration among heterogeneous types of the artifacts managed by the tools in a TSPACE and raises awareness of the stakeholders' operations that are performed on the artifacts using the provisioned tools*.

As discussed in previous chapters, our research effort has been motivated by the need to provide the key specifications and architectural guidance in terms of a reference architecture for designing and evaluating a TSPACE for provisioning software architecting TaaS. A software reference architecture "maps division of functionality together with data flow between the pieces onto software elements and data flow between the elements" [5]. A software reference architecture also provides a standardization and an abstraction of a concrete software architectures for a specific domain, facilitates the reuse of design knowledge and reduces the cost of creating new design solutions for the domain [15].

This chapter deals with architecture components of TSAPCE and presents a set of key specifications, a design process and a description of TSPACE reference architecture that can support a modular and highly configurable TSPACE. Different vendors can provide specific modules of the proposed TSPACE reference architecture and individual software architecture tools can be hosted and provisioned using the TSPACE. We foresee that the proposed reference architecture will make it easier to design new cloud-based workspaces for TaaS, to analyze and evaluate existing ones as well as significantly facilitate the software development process of TaaS workspaces. The description of the TSPACE reference architecture details the functionalities to be supported, architecture design decisions [148], and different abstractions and views of the reference architecture. Whilst the architectural concepts and design decisions presented in this chapter are generic enough to be applied to design and evaluating a TSPACE for any other engineering domain, we have focused on software architecting domain. The main research contributions that are discussed in this chapter are:

- We present a meta-model to characterize TSPACE and to design concrete architecture for providing TSPACE. We also present a structure of a set of ontologies that formalize the tools selection, tools provisioning and semantic integration among the artifacts consumed or generated by the hosted tools.

- We provide a detailed description of the TSPACE RA by using multiple levels of abstractions [10] and rationalizing the incorporation of different modules and components in the RA. The RA is described in terms of development view, logical view, process view and deployment view, as recommended by view-based approaches [10]. We also identify different solutions that can be used to implement the RA.

- We demonstrate the use of well-known design principles and architectural patterns [148] for designing and reasoning architectures for TSPACE. The description of the used patterns and their pros and cons can provide guidance for implementing the RA for different engineering domains.

The organization of the remainder of the chapter is as follows. Section 6.2 explains TSPACE architecture design and description strategy. Section 6.3 describes TSPACE design tactics. Section 6.4 describes detailed TSPACE reference architecture design using multiple views. Section 6.5 provides an overview of prototype implementation of the reference architecture, and Section 6.6 provides insights on TSPACE reference architecture evaluation.

6.2. Reference Architecture Design and Description Strategy

While designing TSPACE reference architecture, we have addressed reference architecture documentation and TSPACE functional and quality requirements (elaborated in Chapter 4). We have leveraged findings from the synthesis of cloud software architecture solutions (discussed in Chapter 2) for incorporating *aaS quality characteristics in TSPACE reference architecture. TSPACE reference architecture also encompasses TSPACE ontologies (discussed in Chapter 5). Since a reference architecture provides valuable guidelines for designing a concrete architecture, it is important to describe a reference architecture as comprehensively as possible and in an easy-to-understand way. We have described TSPACE reference architecture using a systematic approach that advocates the use of context, goal and design dimensions of a reference architecture [15, 16], as discussed in Section 3.3.1.2 in Chapter 4. Table 35 lists different dimensions of the reference architecture documentation and the proposed TSPACE reference architecture.

Table 35: TSPACE Reference Architecture Documentation

Dimension	Sub-dimension	TSPACE RA Solution
Context	Who defines it?	It is defined as a part of a research project.
	Where will it be used?	It aims to facilitate implementation and evaluation of a TSPACE for industrial trials.
	What is the maturity stage of the domain?	The corresponding architecture domain is considered as preliminary because to the best of our knowledge, comprehensive solutions are not yet available.
Goal	Why is it defined?	It aims to facilitate the design of a concrete TSPACE by providing the development, logical, process and deployment views of the RA.
Design	What is described?	The RA is described in terms of high-level modules, connectors, details of the modules in terms of components using multiple views and design principles of the RA.
	How is it described?	It is described using textual description and diagrams.
	How is it represented?	We have shown high-level representations using semi-formal approaches with the help of lines and boxes, and have described details using UML diagrams.
Instantiation	How is it instantiated?	We have evaluated the instantiation of the RA by building a prototype.
Evaluation	How is it evaluated?	We have evaluated the RA using scenarios for functional requirements and quality parameters, and assessed its feasibility by having a prototype.

6.3. TSPACE Reference Architecture Design Tactics

We have designed TSPACE reference architecture for supporting software architecting activities such as architecture analysis and design. TSPACE reference architecture is generic enough to be adopted for supporting engineering efforts in other domains. We have developed the presented reference architecture experimentally and iteratively. For designing the

reference architecture, we have followed the functional decomposition and part-whole principles [5] and several architectural styles. TSPACE reference architecture consists of four abstraction layers; several components and sub-components on each layer have been structured based on the part-whole principle to achieve functional and non-functional requirements.

Functional decomposition and part-whole principles help to achieve a number of quality characteristics such as modifiability and integratability. Functional decomposition also makes it easy for practitioners and researchers to understand different components of the reference architecture and to tailor it for their specific needs. We have used an ontology-based semantic integration approach to support flexibility and interoperability. Ontology-based semantic integration enables the reference architecture to accommodate different types of artifacts produced or consumed by different tools using standardized or proprietary formats. We have defined a clear connection between the interfaces of semantic integration layers. We have also defined explicit components that manage semantic models of a TSPACE at different levels of abstraction.

One of the core elements of the proposed reference architecture is a meta-model to characterize the elements of a TSPACE and the relations among the elements (Figure 23). Since we intend to concretize the TSPACE reference architecture for software architecting domain, we have decided to develop TSPACE meta-model by following and extending the conceptual meta-models of architecture description provided by IEEE 1471-2000 [47] and ISO/IEC/IEEE 42010:2011 [7]. The extended TSPACE meta-model is shown in Figure 23, which is a detailed view of the meta-model that has been briefly discussed in Section 5.2 in Chapter 5. The meta-model shows an abstract TSPACE and its specialization of architecting TSPACE (i.e. TSPACE instance for the tools that are used for software architecting activities). The proposed meta-model is explained in following paragraphs.

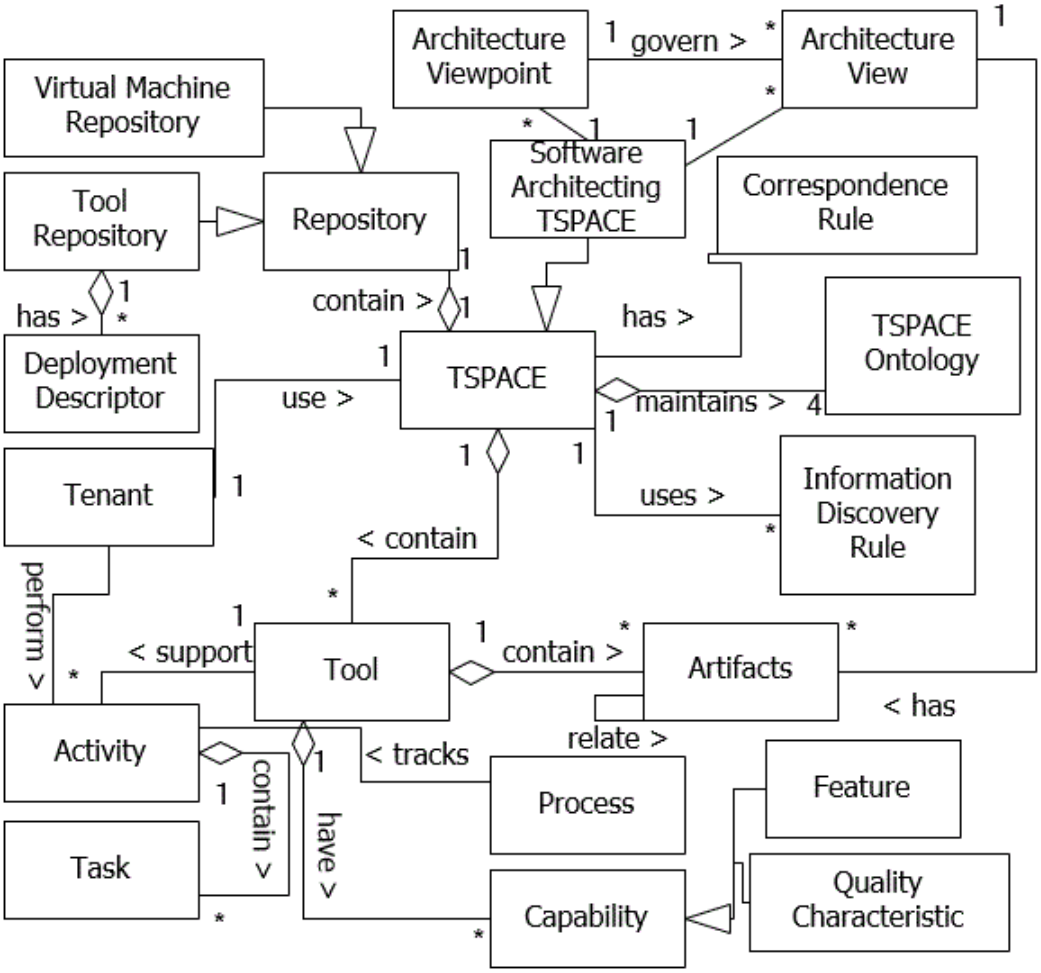


Figure 23: TSPACE Meta-model

A user (tenant) can associate the required tools with a TSPACE in two ways: (a) the tools can be provisioned by third party vendors and are integrated with TSPACE via plug-ins, and (b) the TSPACE enacts the required tools and hosts them on the virtual machines. As a result, TSPACE consists of two types of repositories, namely tools repositories and virtual machine repositories on which tools can be hosted. The hosted tools provide different types of features and support different types of quality characteristics (e.g., scalability and availability). In the meta-model, the features and quality characteristics of the tools are represented as tools' capability. The hosted tools provide support for different types of activities and sub-tasks of those activities. Each tool can enable a user to work on the required artifacts that may be in a standard format such as UML models or a tool's proprietary format.

As previously stated, a project's stakeholders usually work with multiple tools provided by commercial vendors or an Open Source community. These tools need architectural level support for interoperability so that the artifacts produced in different formats (texts, diagrams, standardized formats and proprietary formats) can be integrated with each other. We have proposed to leverage semantic technologies for tools integration; however, our solution needs to be complemented by appropriate architecture abstractions for information discovery from tools. The architecture of a TSPACE also needs to have a set of rules to support collaboration, awareness of the operations on the artifacts and information discovery of the related artifacts as a project's stakeholders usually perform different activities using multiple tools. The meta-model in Figure 23 shows a specialization of a TSPACE for the software architecting tools. As shown in the figure, an instantiation of a TSPACE for a specific domain may require additional concepts such as architecture viewpoints and architecture views, as in the case of software architecting TSPACE. Hence, the TSPACE reference architecture meta-model also provides flexibility to incorporate additional concepts by supporting dynamic composition and aggregation of different concepts in a TSPACE. In following subsections, we describe TSPACE architecture design tactics to address TSPACE requirements that have been reported in Chapter 4.

6.3.1. Use of Ontologies to Formalize TSPACE

We propose the use of ontologies to formalize TSPACE [49] because ontologies provide shared conceptualization and vocabulary that can be used to model a specific domain [48]. Figure 24 and Figure 25 show an aggregated view of the TSPACE ontologies that have been described in detail in Chapter 5. The TSPACE ontologies consist of four specializations, namely Artifact Ontology, Capability or Tool Ontology, Annotation Ontology and Change Ontology. The TSPACE meta-model that is presented in Figure 23 shows an overall structure of the TSPACE elements whereas the TSPACE ontologies provide the basis for formalizing the tools selection process, establish the relationship among the artifacts that are produced or consumed in a TSPACE instance, and capture the operations that are performed on artifacts. The presented ontologies are based on the TSPACE meta-model and have been inspired by the data interoperability semantic model [113].

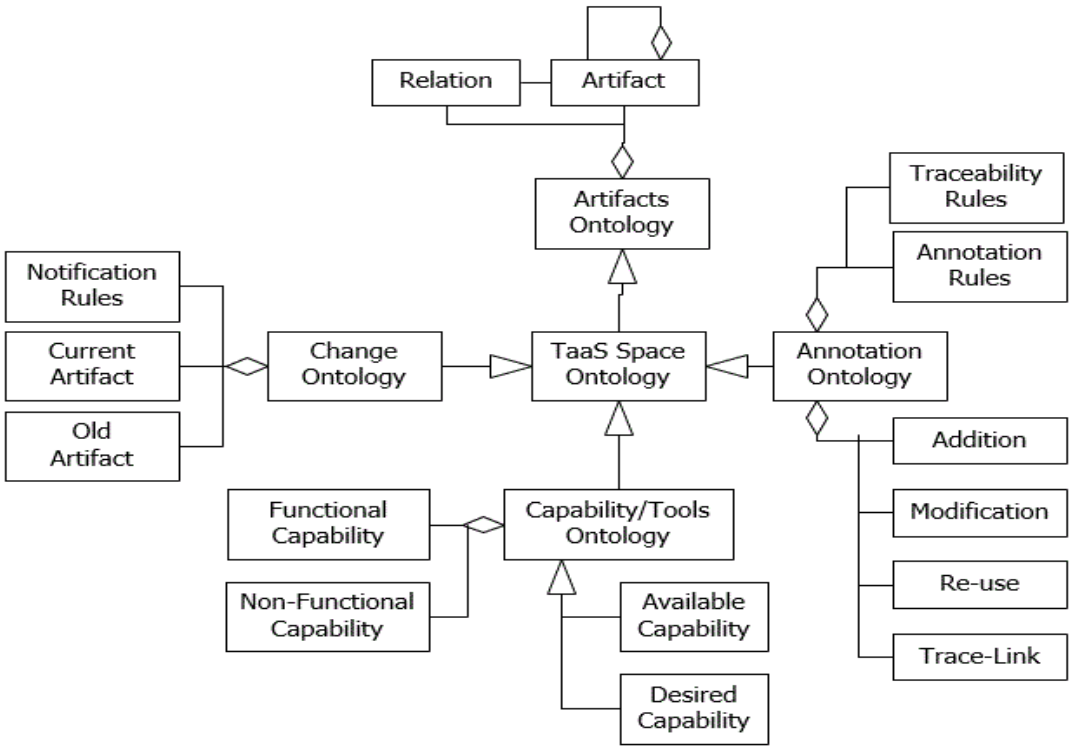


Figure 24: TSPACE Ontology Meta-model Structure

Different components of TSPACE reference architecture (to be discussed in Section 6.4) use one or more specialization of TSPACE ontologies that have been elaborated in detail in Chapter 5. *Artifact Ontology* consists of multiple abstractions of TSPACE. Each abstraction can have nested abstraction and can consist of one or more artifacts. *Artifact Ontology* also contains the relationships among abstractions and the contained artifacts. *Tool Ontology* captures the functional and non-functional requirements supported by the tools associated with a TSPACE instance. *Tools Ontology* can be used to capture the capability of tools as well as the functionality and quality characteristics users require from a specific instance of a TSPACE. *Annotation Ontology* provides support for annotating the artifacts that are produced or consumed by the tools constituting a TSPACE. *Annotation Ontology* provides support for annotating artifacts for addition, modification and re-use within the same instance of TSPACE. It also provides support for manual definition or automatic recovery of trace links between artifacts based on the relationships supported by *Artifact Ontology*. This ontology consists of rules for both annotation and traceability relations. The rules govern valid annotation and traceability relationship. Finally, *Change Ontology* keeps track of the old and new versions of the artifacts in a TSPACE and raises awareness of the operations that are performed on the artifacts among users by generating notification according to defined *Notification Rules*.

Benefits: Our decision to use ontologies at the core of the reference architecture appropriately formalizes the concepts about a TSPACE. It also makes the reference architecture flexible and dynamic enough to accommodate different types of tools.

Challenges: Building ontologies for complex domains is not a trivial undertaking. The process of building such ontologies requires expertise in domain knowledge for defining the high-level concepts and relationships between different artifacts. The meta-model presented in Figure 23 shows high-level relationships between different concepts and artifacts of a Software Architecting TSPACE. The meta-model needs to be extended for other domains in the same manner we have followed for the Software Architecting TSPACE.

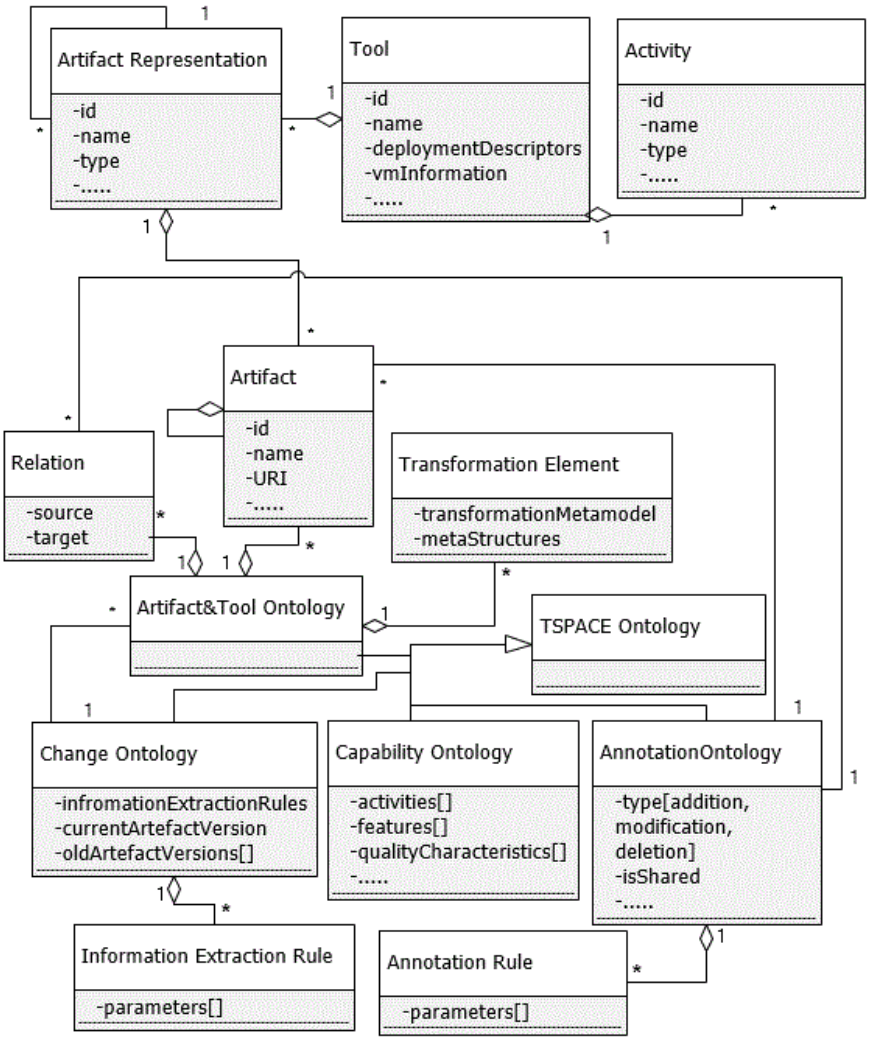


Figure 25: TSPACE Ontology Meta-model Detail

6.3.2. Using SOA and REST as TSPACE Façade

For designing the façade of the reference architecture, we used Service Oriented Architecture (SOA) [149] and REST architecture styles [150]. The tools associated with a TSPACE interact with the reference architecture via its façade.

Benefits: The use of SOA and REST makes it easy to modify the reference architecture's components and supports seamless integration of heterogeneous tools to be hosted. The tools hosted in a TSPACE can be traditional desktop-based tools hosted in Virtual Machines (VMs), web-based application enacted by a platform or cloud-based tools provided by third party vendors and integrated with the platform. Different technologies are used for implementing different types of tools and allowing a platform-neutral façade based on SOA and REST styles paves the way for seamless integration.

Challenges: For certain tools, it may not be possible to write plug-in or probes to have direct interaction with a platform using SOA or REST interfaces. In such cases, intermediate glue code components can be required.

6.3.3. Using Centralized (Shared) Repository Pattern

We used shared repository pattern [50] to provide a common Global Ontology Knowledgebase to TSPACE instances for multiple domains. A centralized ontology repository hosts standard abstract Artifacts Ontology, Annotation Ontology, Change Ontology and Capability Ontology for different domains.

Benefits: A centralized global ontology repository provides a single point of access to different ontologies of a TSPACE. It also positively addresses the modifiability characteristic (QR7) of TSPACE reference architecture.

Challenges: A centralized repository pattern can become a performance bottleneck if there are multiple instances of a TaaS accessing the repository. This risk can be mitigated by having replications of the repository and a load balancer.

6.3.4. Using Pipes and Filters Pattern

There can be a number of tools in a specific instance of TSPACE, and the reference architecture needs to support multiple TSPACE instances. There needs to be architectural support to handle an increasing amount of data generated by multiple tools associated with each instance of a TSPACE. That

is why we used *pipes and filters pattern* [4] in the reference architecture to meet the performance requirements of the platform.

Benefits: The adoption of the two staged pipes and filter architecture style provides a queuing mechanism. In the first stage, there is a common queue pipeline at which data from all the tools belonging to different instances of a TaaS are received. In the second stage, there are multiple queue pipelines corresponding to an instance (for a specific tenant) of TSPACE. The input data are sent to the queue of the corresponding tenant with the help of a monitoring filter.

Challenges: If the input data streams scale rapidly, having only one monitoring filter may become a performance bottleneck. Multiple monitoring components can be attached to the first queue pipeline to address the scalability issue.

6.3.5. Loosely Coupled Layers

The layered architecture style [4] is widely used to provide loose coupling and separation of concerns in a system. We used the layered architecture at multiple levels of abstraction in TSPACE reference architecture.

Benefits: The layered architecture style makes it easy to implement and evolve different components of the reference architecture independently of each other, and plug in third party tools.

Challenges: The layered architecture style requires explicit interfaces for components in each layer via which other layers can utilize its functionality. This may result in more effort while materializing the reference architecture. Layered architecture can also have negative impact on performance. However, the potential negative affects of the layered approach can be mitigated by incorporating performance improvement techniques for data retrieval (such as data caching).

6.3.6. Using Resource Description Framework for Information Extraction

The requirements associated with semantic integration demand a mechanism that can be used to extract different types of knowledge and information from a TSPACE instance to provide support for awareness of the operations that are performed on the artifacts (users activities) and collaboration between the users. We have used Resource Description Framework (RDF) [151] to extract and structure information from TSPACE ontologies and maintain the information in a query-able manner. Figure 26 (a) and (b) shows different

stages of RDF construction from the annotated data of content elements (CE) of TSPACE that is sent to a TSPACE instance via its APIs (e.g., via plug-ins, add-ins and probes on each of the provisioned tool). At the first stage, the annotated data is used to construct RDF corresponding to artifact ontology for each tool. At the second stage, tools-specific RDFs are merged together to generate an aggregated RDF for a TSPACE instance. Being a query able data structure, RDF enables dynamic definition of information extraction rules. Centralized repository pattern [50] is used to provide root ontology templates for all TSPACE instances, as shown in Figure 26(a). Ontology templates are used as a baseline by each TSPACE instance and are populated according to specific configurations of the provisioned tools.

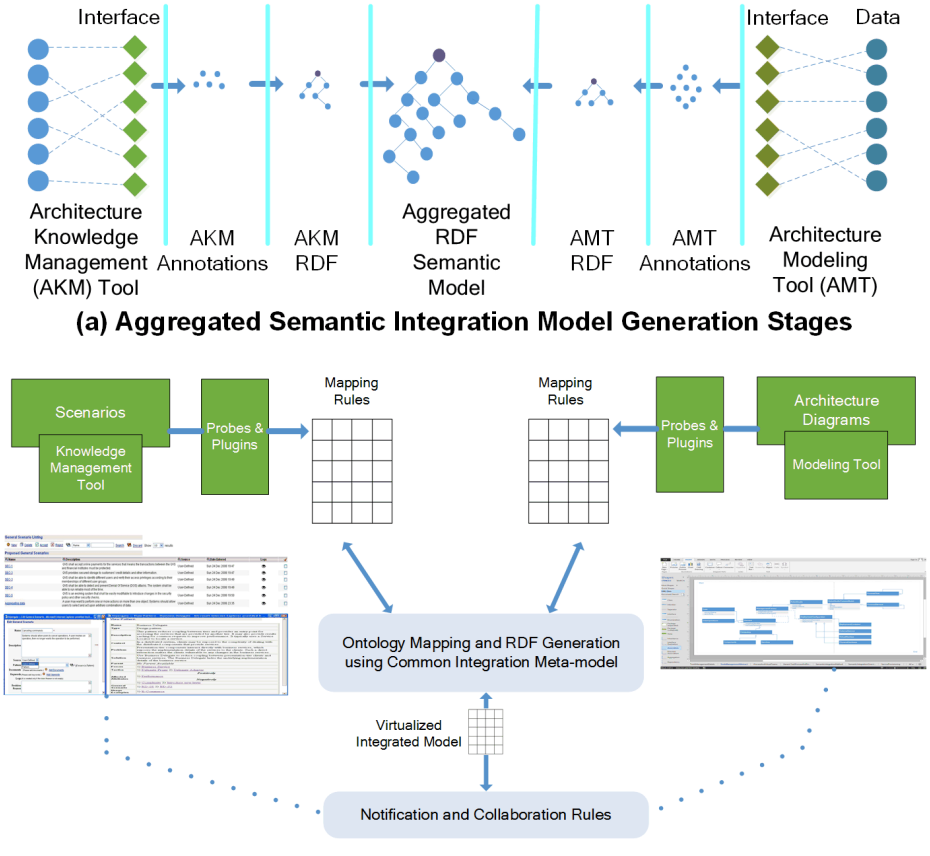


Figure 26: Semantic Integration Stages

Benefits: The query able nature of RDF provides an extendable and easily modifiable approach to define information extraction and collaboration rules that have been briefly discussed in Chapter 5.

Challenges: Generating RDF for all the data in a TSPACE instance can become a performance bottleneck when a large amount of semantically related data is being maintained in a TSPACE instance. To avoid this bottleneck, an incremental RDF generation approach is adopted. Therefore, instead of generating an RDF from the ontology structure when the data needs to be queried, RDF is updated whenever new data is added in the TSPACE instance, and a query on the data can be run at anytime without the need to regenerate RDF. (A code snippet of the incremental RDF generation method is shown in Section 6.5 while describing an overview of the prototype.)

6.3.7. Use of SPARQL for supporting Dynamic Rules

The requirement QR4 emphasizes the need to support dynamic update of awareness and collaboration rules. For this purpose we have used SPARQL [121] based information extraction rules in TSPACE. SPARQL is a query language for RDF based data structures.

Benefits: Using SPARQL provides flexibility to define rules according to specific needs of the domain in which TSPACE is to be adopted.

Challenges: Defining rules using SPARQL can be challenging. For this purpose, a set of reference queries and methods are provided in the prototype implementation of TSPACE to facilitate incorporation of new rules.

6.3.8. Using Redundancy of Pipes and Filters to Support Scalability

The requirement QR3 identifies the scalability needs of TSPACE as the number of users and the activities that are performed using the tools grows. Scalability architecture is shown in Figure 27.

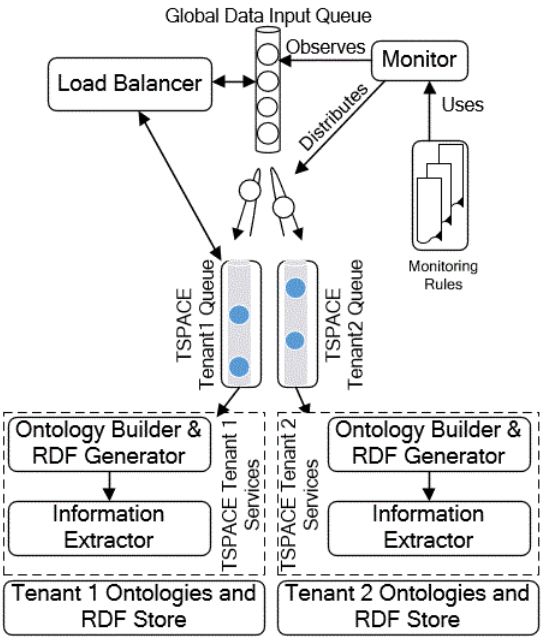


Figure 27: TSPACE Scalability Pattern

Scalability in TSPACE is achieved in three different ways. (i) First, inclusion of three levels of HashMap-based indexing technique allows us to retrieve information with constant access time even though the artifacts and CEs grow exponentially. When a TSAPCE instance services multiple tenants, the first HashMap returns a reference to ontology map of that specific tenant using tenant identifier as a key. The second HashMap returns reference to a specific ontology map, e.g. tool and artifact ontology and third hash map returns a reference to the specific data element (e.g. an artifact), which needs to be accessed. As a result, TSPACE provides access to a specific data element in a constant access time even though the data grows rapidly. Every ontology and data element in a TSPACE instance has a unique identifier that is used as a key for the third level hash map. A HashMap structure is shown in Figure 39(b). (ii) Second, we have incorporated two FIFO (first in first out) queues attached with the APIs that receive the data from the plug-ins. The first queue is tenant-independent queue that received data from all the tools. The second set of queues consists of tenant-specific queues. A monitoring pattern is applied [152] to monitor and assign the data to corresponding queues. A Monitor component fetches the data from the first queue and puts it at the tail of the corresponding tenant-specific queue. The monitoring is performed according to the monitoring rules that define how to extract tenant information from the input stream of data. The queues ensure that TSPACE architecture is able to handle growth of data without impacting TSPACE operations. (iii) Third, we have introduced a load balancer component in case the first level queue is replicated. The load balancer fetches the data from the head of the replicated

first level queues and passes it to replicated Monitor components in round robin manner. The monitoring component then places the data on the tail of tenant specific queue. The information from the head of tenant-specific queues is fetched by the corresponding TSPACE tenant's components instances.

Benefits: Providing scalability points at three different levels guarantees to provide TSPACE provisioning as *aaS model.

Challenges: Autonomous replication of TSPACE components on cloud can be challenging. However, incorporating scalability features of underlying IaaS cloud can easily mitigate this risk.

6.3.9. Using Location-Specific Provisioning to Satisfy Location Constraints

The requirement FR1 highlights the importance of TSPACE capability to provision tools according to specific location constrains on the underlying IaaS cloud. Location specific provisioning can be achieved by using enactment APIs (e.g. Amazon Provisioning APIs[21]) of underlying IaaS cloud provider.

Benefit: Using IaaS's location-specific enactment features provides an easy mechanism to guarantee adoption of TSPACE for the projects that involves working on artifacts and data of sensitive nature.

Challenges: Unavailability of location-specific provisioning features in an IaaS cloud can pose a challenge. However, more robust solutions for hybrid cloud provisioning models, e.g. frameworks such as IBM Altocumulus [74] can be adopted to address this risk.

6.3.10. Multi-tenancy

Multi-tenancy is an important characteristic of *aaS model [102]. The proposed reference architecture fulfills the multi-tenancy characteristic to provide proper isolation of the tools and data of one tenant of a TSPACE instance from other tenants (QR2).

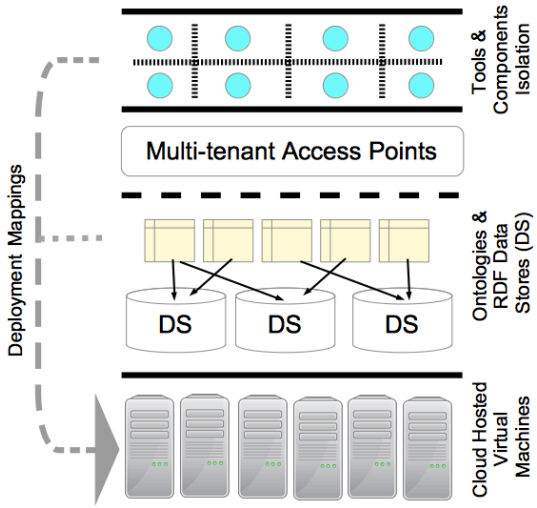


Figure 28: Multi-tenancy Layers

Figure 28 shows a layered multi-tenancy architecture pattern used in TSPACE reference architecture. The isolation between the architectural elements has been provided at three different levels of abstractions. At the first level, the isolation between the tools of a TSPACE and different components of the reference architecture is supported. At the second level, the isolation between the ontology instances and the RDF data stores is provided corresponding to each instance of a TSPACE. That means there has to be a multi-tenant access point that can act as a bridge between the plug-ins (probes) of the tools, and ontologies and the corresponding RDF data stores. Examples of such access points from the proposed RA are *Data Monitor*, *Tenant Independent DC Queue* and *Tenant Specific DC Queues* (to be explained in Section 6.4.2.2.1 and Section 6.4.2.4). At the third level, the isolation is provided at Virtual Machine (VMs) level where the tools that are provisioned by TSPACE reference architecture are hosted on separate cloud-based virtual machines.

Benefits: Incorporating multi-tenancy in TSPACE allows the reference architecture to serve multiple tenants and have tenant specific configurations of collaboration and information discovery mechanisms.

Challenges: Compartmentalization of data stores and components for all possible scenarios can be challenging. For such cases, more sophisticated multi-tenancy solutions can be adopted. The WSO2 carbon platform [73] can be used to provide isolation between components of a TSPACE instance to complement the architecture design decisions of combining a multi-tenant access point (to be described in Section 6.4.2.4) with pipes and filters pattern. The information flow authentication model based on security policy [153] and role based authorization mechanism [154] can be incorporated to implement

security in multi-tenant access points. The multi-tenant access and indexing techniques [92] can be used for multi-tenant persistence of ontologies and corresponding RDF data stores.

6.4. TSPACE Architecture Design and Decomposition of Architecture Elements

We present TSPACE reference architecture at four levels of abstractions. First we describe the top-level modules; then we decompose those modules into components and sub-components. There are some components that provide abstraction of the external systems (e.g. provisioning components) whereas other components are described in detail as part of the reference architecture. The legend presented in Figure 29 shows the notations that are used in the diagrams of the reference architecture.

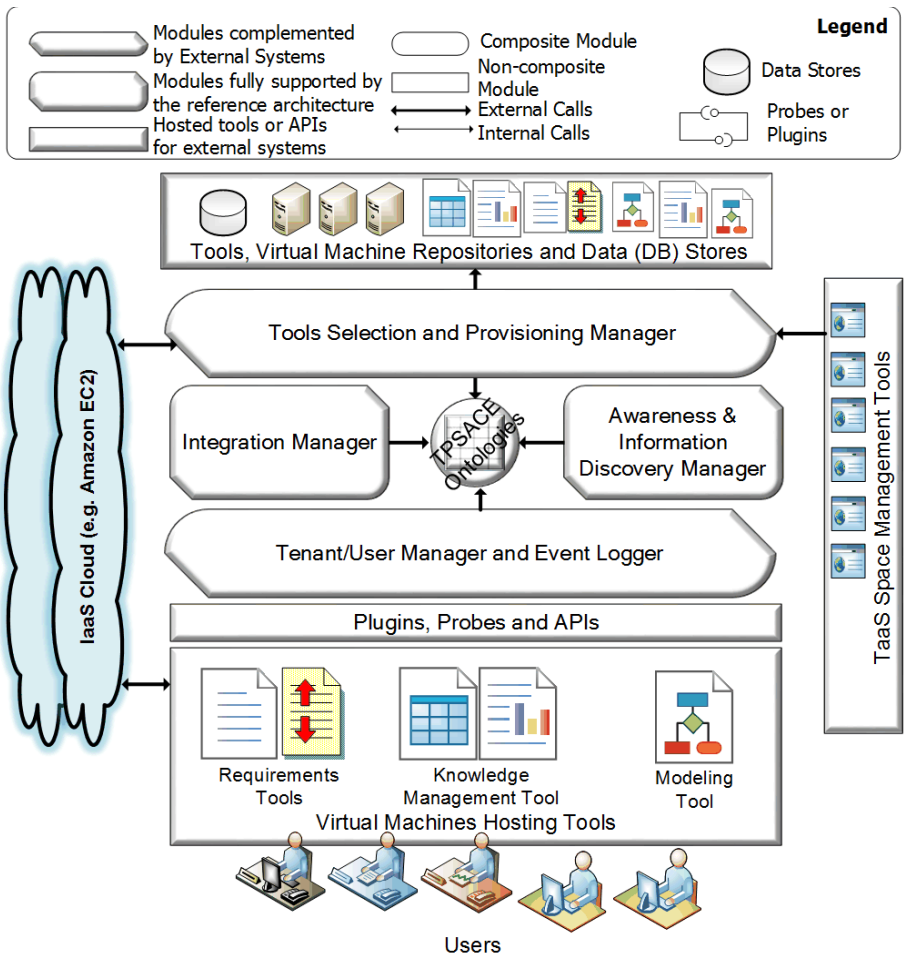


Figure 29: TSPACE Architecture - First Level Decomposition

6.4.1. First Level Decomposition

According to the functional requirements (Chapter 4), three lifecycle phases of tools (enactment and provisioning, semantic integration and awareness of activities and operations on the artifacts) constituting TSPACE are supported by TSPACE reference architecture. Figure 29 provides an overall representation of the reference architecture (development view). The modules at the first level of decomposition are organized following the layered architecture style [5]. The TSPACE reference architecture conceptually consists of four modules: (i) Tools Selection and Provisioning Manager, (ii) Integration Manager, (iii) Collaboration, Awareness and Information Discovery Manager, and (iv) Tenant (User) Manager and Event Logger.

Tools Selection and Provisioning Manager enables users to select the tools that are suitable for the activities to be performed. *Integration Manager* supports process centric and semantic integration among the tools and the artifacts that are maintained by the provisioned tools. *Awareness and Information Discovery Manager* helps extract the information that can be used to notify users about different events that are triggered in a TSPACE. The events are triggered according to the rules defined in an instance of TSPACE with respect to the corresponding domain in which the reference architecture is used. *Tenant Manager and Event Logger* manages the tenants' authentication and identification. It also logs operations that are performed on the artifacts using the tools. At the core of the reference architecture, there is an ontology-based semantic integration model (Section 5.3.2 and Section 6.3.1). All the tools constituting a TSPACE and the relevant artifacts are annotated using the Annotation Ontology of the semantic integration model (Section 5.3.4 and Section 6.3.1).

Each module is further divided into multiple components and sub-components. Each component provides methods that can be invoked by components in other modules. We have used façade pattern [155] to support integration among components and modifiability (QR7). The decomposition at the first level fulfills the functional requirements that have been discussed in Chapter 4. We have also described the collaboration (using collaboration diagrams) between the components of each module to show the data exchange between the components.

6.4.2. Second and Third Level Decomposition

The decomposition of *Tools Selection and Provisioning Manager* is based on requirements FR1, FR2, FR3 and QR1. FR1, FR2 and FR3 deal with enactment of TSPACE based on the tools' needs for the activities of a specific project and with respect to the location and resource constraints. QR1 deals

with automation of the provisioning process. Decomposition of *Integration Manager* is based on providing support for process-centric integration among the tools (FR4) and semantic integration among heterogeneous artifacts (FR4), awareness of the users' operations and activities that are performed on the artifacts (FR4), interoperability (QR5), modifiability (QR7) and integration (QR5). Decomposition of *Collaboration, Awareness and Information Discovery Manager* provides awareness to users about different actions performed on the artifacts using different tools constituting a TSPACE (FR5). *Tenant Manager and Event Logger* facilitates to incorporate multi-tenancy features in TSPACE reference architecture (QR2). We have also considered the interactions among different components to describe the behavioral model (process view) of the reference architecture and have presented the collaboration diagrams corresponding to main components of the RA.

6.4.2.1. Tools Selection and Provisioning

The components constituting this module provide support for tools selection and provisioning. The high-level views of the architectures from [74-76, 156] inspire the RA and have been extended for TSPACE by incorporating the tools selection ontology (Section 5.3.1 of Chapter 5) that formalizes tools' capability and users' requirements for tools.

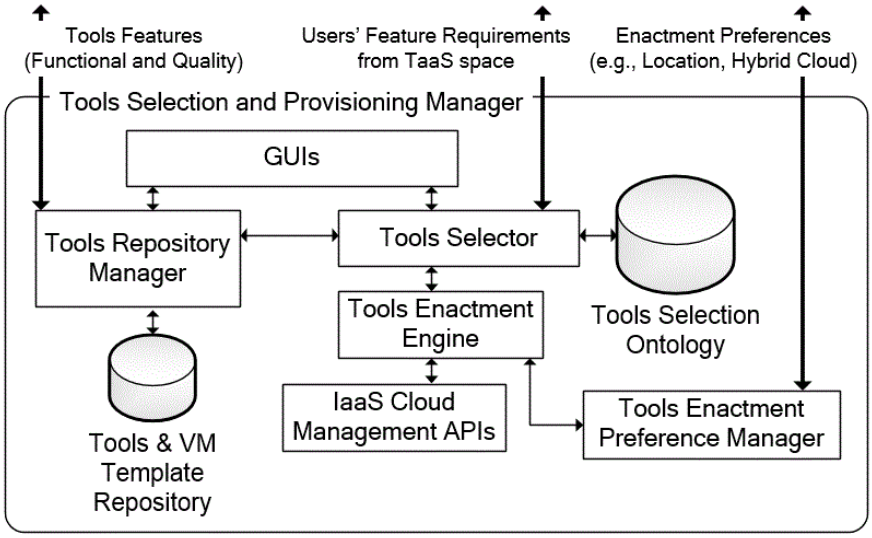


Figure 30: Tools Selection and Provisioning – Logical View

Figure 30 shows decomposition of *Tools Selection and Provisioning Manager*. The *Graphical User Interfaces (GUIs)* provides an interface that supports users interaction and allows administrators to register tools with an

instance of the RA, allows stakeholders to specify their tools' requirements and supports administration activities. The *Tools Repository Manager* component maintains a repository of tools that are registered with the system, the Capability Ontology model of each tool and the VMs that are to be used to host the tools. For example, the tools versioning management strategy proposed in [156] can be used to maintain and provision different versions of tools in a specific instance of a TSPACE. *Tool Selector* transforms a user's tools' requirements into a relevant ontology and compares it with the Capability Ontology of all the tools registered by the *Tools Repository Manager* to provide the best possible match to the desired tools requirements. *Tools Enactment Preference Manager* takes care of the constraints associated with the enactment of the tools. For example, location constraints require that all the tools for a specific instance of TSPACE shall be provisioned from a public or private Infrastructure as a Service (IaaS) clouds hosted in European Union territory. *Cloud Enactment Engine* enacts tools on an underlying IaaS cloud using *IaaS Cloud Management APIs*. The APIs of the public IaaS cloud such as Amazon EC2 APIs [157] can be used in case the tools are to be deployed on a public cloud. If a private or hybrid IaaS is to be deployed, then a cloud management framework such as IBM Altocumulus Framework [74] can be used.

Behavioral Model: Figure 31 shows the collaboration between components of *Tools Selection and Provisioning Module*. Tools Selector component receives a TSPACE instantiation request via GUIs. The desired tools are selected using *Tools Selection and Capability Ontologies*. *Tools Enactment Engine* provisions and enacts tools using *IaaS Cloud Management APIs* by seeking input from *Tools Enactment Preference Manager,* and *Tools and Virtual Machine Template Repositories* component.

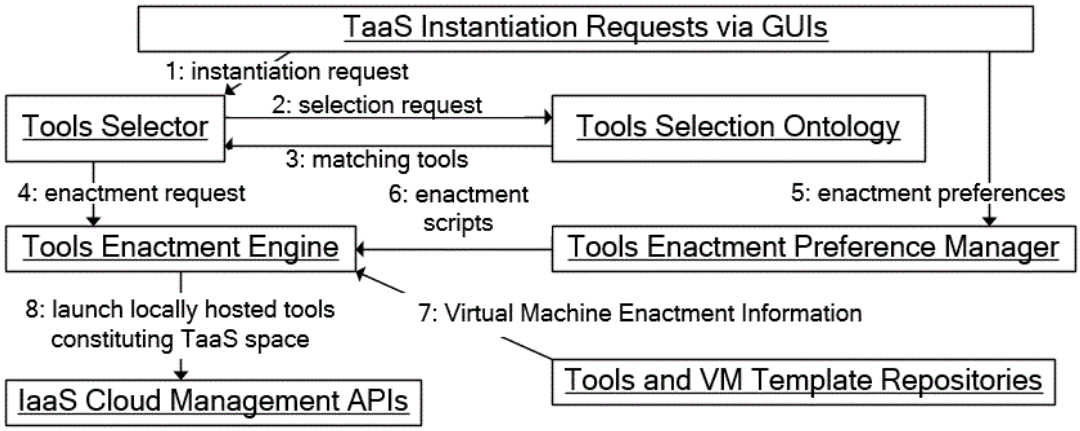


Figure 31: Tools Selection and Provisioning - Process View

6.4.2.2. Integration support in TSPACE

Integration Manager facilitates integration among the tools in two different ways: (i) It provides support for semantic integration (FR4) using the ontologies that have been described in Chapter 5, and (ii) it provides process-centric integration (FR4) among the tools so that the tools can exchange artifacts when the tools are provisioned in accordance with a specific process. Integration Manager is subdivided into two modules: *Semantic Integration Manager* and *Process-centric Integration Manager*. Moreover some tools can require importing and exporting files, therefore support for simple exchange of the artifacts is also provided in TSPACE reference architecture with the help of a wrapper for cloud storage services.

6.4.2.2.1. Semantic Integration Manager

The components that are included in this module support semantic integration among the artifacts produced or consumed by the tools that constitute a TSPACE and provide a foundation for artifacts' traceability (FR4). There is an ontology-based semantic integration model at the core of this module.

Figure 32 shows the Semantic Integration Manager's decomposition. Plug-ins and Probes that are installed on the provisioned tools to provide Semantic Integration Manager a point of access to the tools. The designed RA can support the implementation of multiple instances of the TSPACE. The data sent from Plug-ins and data collection probes are received at a tenant-independent data collection queue. A Data Monitor component monitors all received data elements and filters for forwarding to a tenant-specific data collection queue. The monitoring and filtering rules to identify tenants from the incoming data stream are maintained by Tenant Identification Rules.

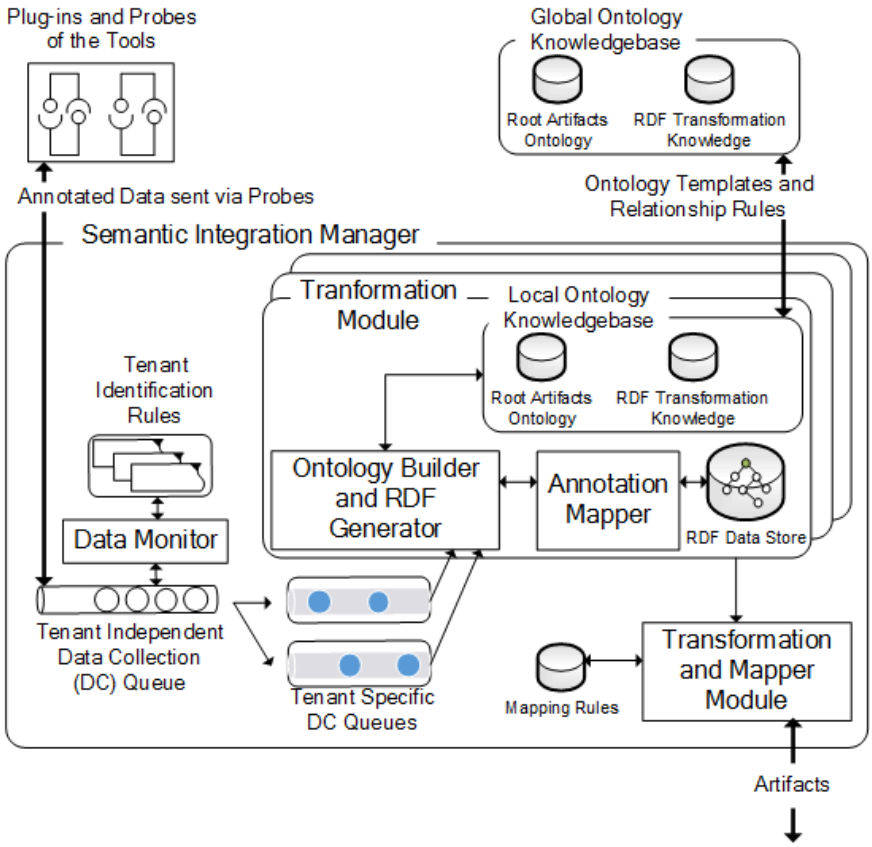


Figure 32: Semantic Integration Manager - Logical View

We have designed a dedicated *Transformation Module* for each instance of the TSPACE. This module handles the data sent by *Tenant Specific Data Collection Queues*. The *Transformation Module* is further subdivided into multiple components. There are two types of ontology knowledge bases in the RA: the *Global Ontology Knowledgebase* maintains the tool Capability Ontology and Artifact Ontology that establishes the relationships among all the possible concepts (the artifacts and their types) that can exist in a specific domain. The *Local Ontology Knowledgebase* maintains the relation between the concepts for a specific instance of a TSPACE corresponding to the tools included in the instance. *Ontology Builder and RDF Generator* populates the root Artifacts Ontology based on the data inputs from *Tenant Specific Data Collection Queue*.

Behavioral Model: Tenant independent and tenant specific data collection queues collect data that is sent by the plug-ins. The Data Monitor component monitors the data according to the tenant identification rules and sends the data to the data collection queue of the corresponding tenant. There is a separate instance of the *Ontology Builder and RDF Generator* component for each tenant. This component fetches the data from the data collection queue of

the corresponding tenant, annotate the data and populate the RDF data store for each tenant. Figure 33 shows the detailed collaboration between the components.

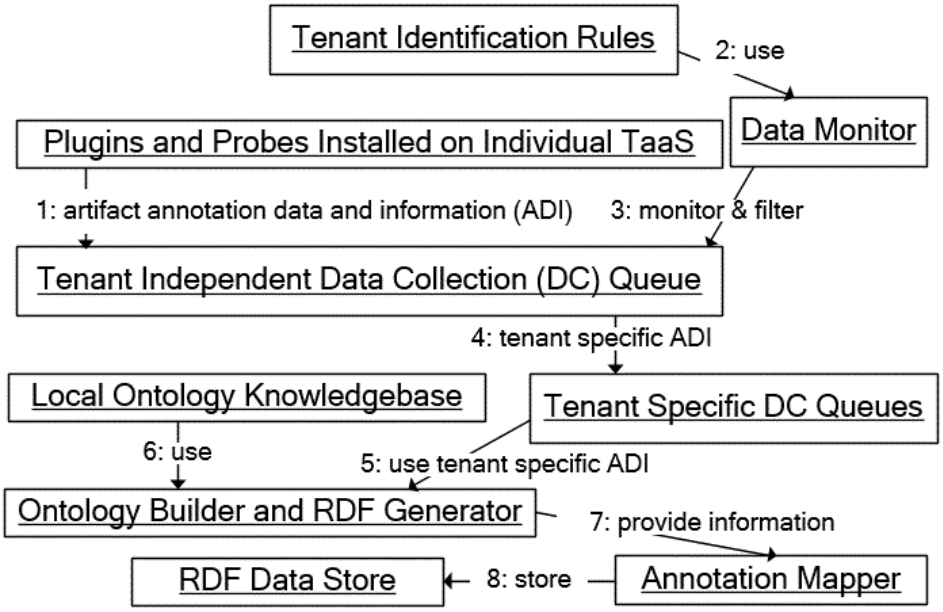


Figure 33: Semantic Integration Manager - Process View

6.4.2.2.2. Process-centric Integration Manager

The components that are included in this module provide process-centric integration among the tools. Figure 34 shows details of the components constituting Process-centric Integration Manager. *Process Engine* is core component of this module. A process can be created and different tools can be assigned to the process with the help of GUIs and is enacted by *Process Engine*. A process consists of multiple nodes that are connected to each other according to a specific workflow. A tool can be attached with a specific node along with the tenant who has access to the tool. A tenant can consist of multiple users. Tenants are assigned to nodes with the help of *Access Manager* component. Between every two interacting nodes of the process, additional services can be attached. These services are used to perform specific operations on the artifacts when the artifacts are propagated from one node to another (e.g. compiling the source code into executable files or encrypting/decrypting of the artifacts).

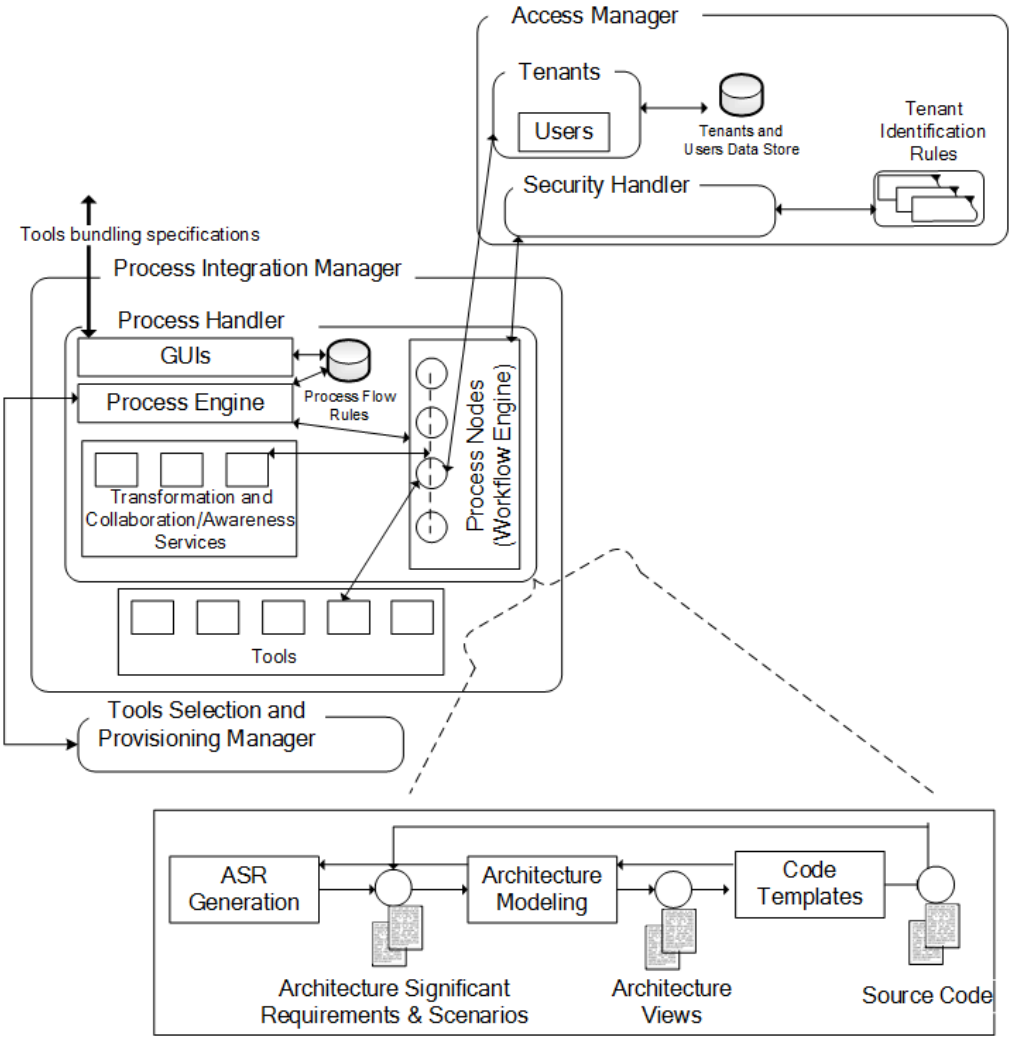


Figure 34: Process-Centric Integration Manager - Logical View

Figure 34 shows pictorial representation of process-centric integration for tools that are used for software architecting. The boxes in the figure depict the activities that are performed by the tools attached to the node, and the circles represent different nodes of the process. The artifacts that are generated by the tools attached with a preceding node of the process are used as an input of the tools that are attached with the proceeding node of the process. *Process Engine* provides APIs that can be used to retrieve different kinds of notifications by the tools. Further detail of the APIs is to be elaborated in Section 6.4.3 while describing fourth level decomposition of TSPACE reference architecture.

Behavioral Model: Figure 35 shows the interaction between the components of *Process-centric Integration Manager*. *Process Engine* prepares workflow script (in form of BPMN XML specification) and tools enactment scripts

based on the process-specific tools enactment parameters that are received by GUIs. Workflow scripts are passed to *Process Workflow Engine*, which enacts the workflow. Tools enactment scripts are passes to *Tools Selection and Provisioning Manager* (Section 6.4.2.1) to enact the tools on an underlying IaaS cloud. Once the tools are enacted, *Process Engine* assigns the tools and tenants (group of users which can access the tools) to different nodes of the workflow.

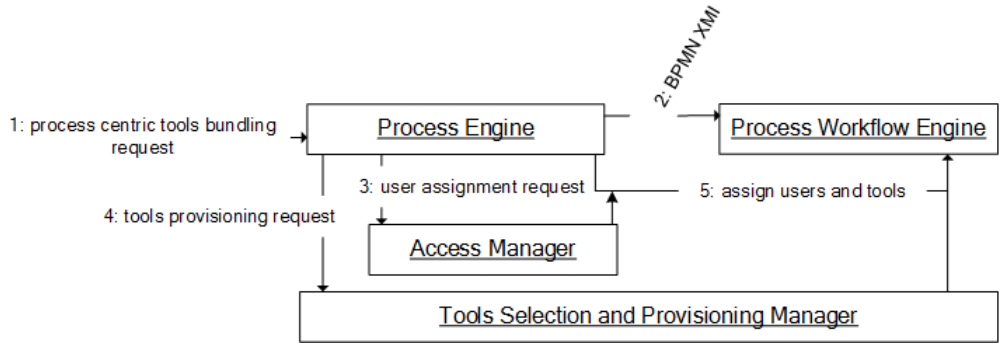


Figure 35: Process-centric Integration Manager - Process View

6.4.2.2.3. Plain Artifacts Exchange

Simple Storage Manager that is depicted in Figure 36 provides a wrapper around the storage services of cloud storage providers. Plain storage service provides an option for the tools to import and export artifacts. This component is provided to complement semantic and process centric integration.

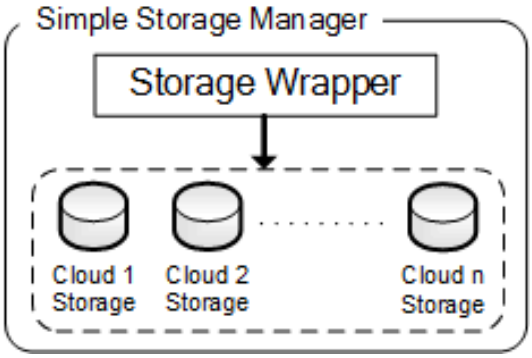


Figure 36: Simple Storage Wrapper

6.4.2.3. Awareness and Information Discovery Manager

This module provides support to raise awareness about users' operations on the artifacts (FR5) and provides support to trace the changes and the sources of the changes to the artifacts during the lifecycle of a TSPACE (FR5). This module leverages the RDF data store that is populated by *Semantic*

Integration Manager and applies information discovery rules for different types of change and trace notifications.

Figure 37 shows a decomposition of *Collaboration and Awareness Manager*. RDF Data Store is at the core of this module. *Change Handler* and *Annotation Handler* respectively manage the structure of the artifacts for changes and annotate the artifacts using global ontology templates and global ontology relationships fetched from the *Global Ontology Knowledgebase*. *Annotation Manager* acts as a data-input source for *Information Discovery Manager* and *Notification Manager*. *Information Discovery Manager* uses predefined information discovery rules that are stored in the information discovery data store. *Information Extractor* uses *SPARQL Query Generator* to generate executable SPARQL [121] queries using information discovery rules and executes the queries on the RDF Data Store that is managed by *Annotation Manager*. SPARQL provides a configurable and dynamic mechanism to query RDF data structures. *Information Provider* acts as a façade between *Information Discovery Manager* and *Notification Manager*. *Notification Manager* generates the change and trace notification for the users using the tools according to the notification rules. The notification rules primarily provide a guide for what information needs to be sent to users for trace and change notification, whether the users have subscribed for pull or push notification and what criteria and frequency exist for push notifications.

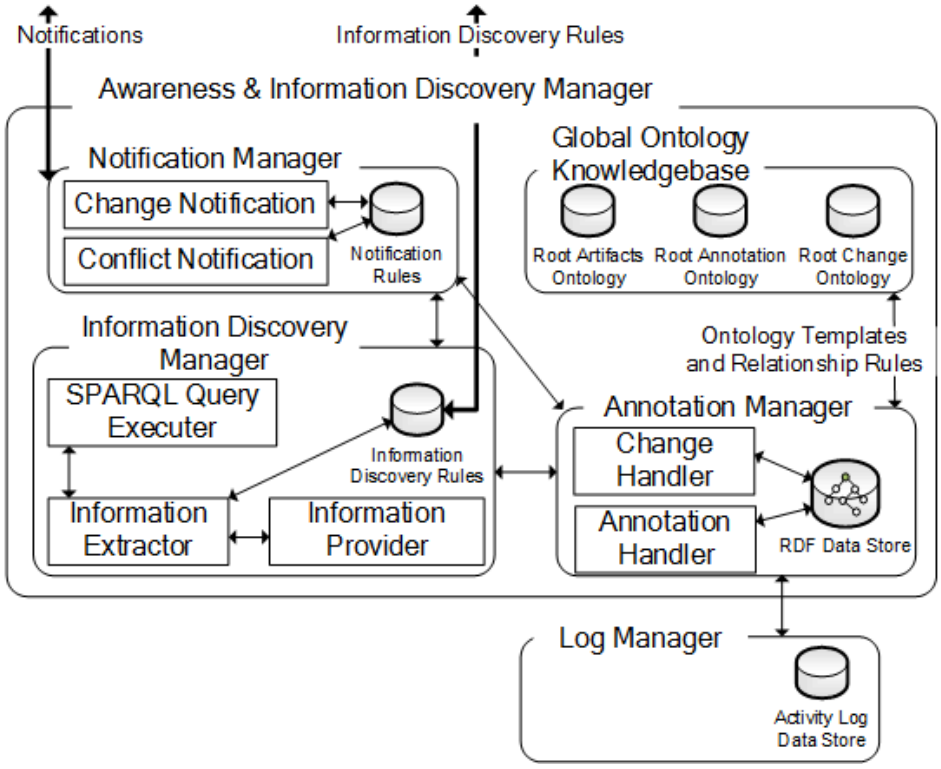


Figure 37: Awareness and Information Discovery Manager - Logical View

Behavioral Model: Figure 38 shows the details of the collaboration between the components and sub-components of the *Collaboration and Awareness Manager*. The *Annotation Manager* populates the *RDF Data Store*, which provides a base for information discovery using the *Information Discovery Rules*. The *Information Discovery Manager* retrieves the information from RDF Data Store and passes the retrieved information to the *Notification Manager*, which uses the *Notification Rules* to generate notifications for Tools' users. The specifications for notifications and information discovery are based on predefined rules specified at the time of instantiation of a TSPACE instance.

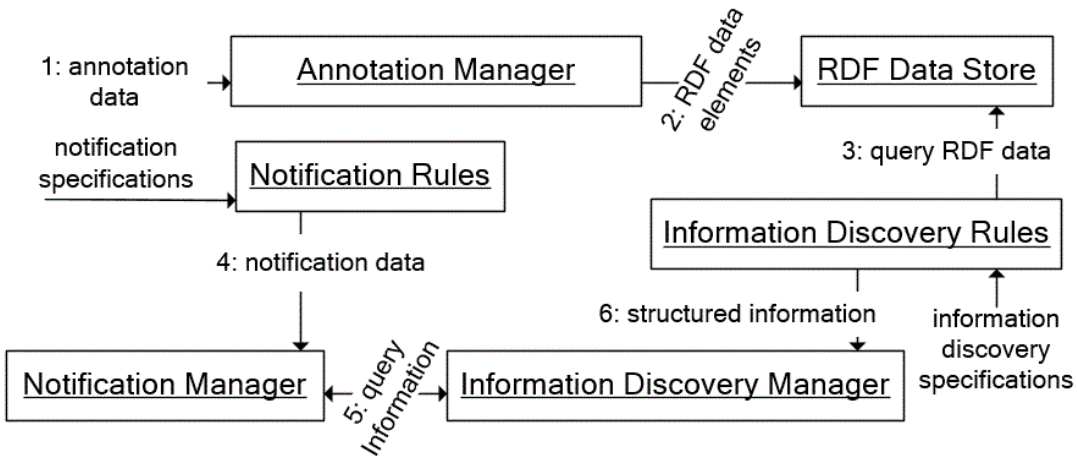


Figure 38: Collaboration and Awareness Manager - Process View

6.4.2.4. Tenant (and User) Manager and Event Logger

The analysis of the requirement QR2 identifies two dimensions of the multi-tenancy: (i) tenant specific instantiation of the ontologies and RDF structure corresponding to the tools provisioned for the tenant and (ii) isolation between the ontologies and RDF instances of one tenant from other tenants. Figure 39 shows details of components to handle multi-tenancy in TSPACE architecture.

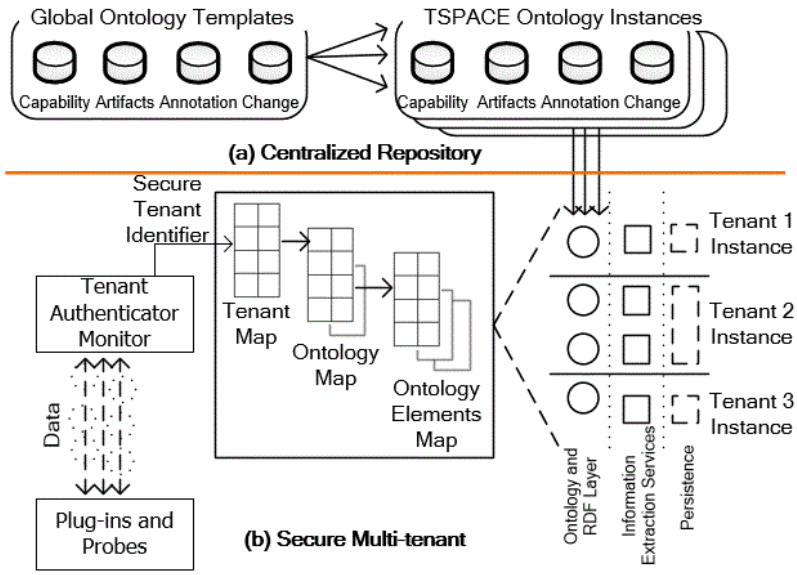


Figure 39: Multi-tenant Access to Artifacts and Data

Figure 39(a) shows the centralized ontology repositories that are used as a baseline in each instance of TSPACE (selected set of tools for a tenant), as described in Section 6.3.3. TSPACE is designed to serve multiple tenants each having its specific set of provisioned tools. We have used three levels of nested hash maps for efficient indexing and retrieval of RDFs corresponding to multiple instances of TSPACE, as shown in Figure 39(b). Interaction between provisioned tools and TSPACE takes place via TSPACE APIs (e.g., via plug-ins and probes that are installed on the hosted tools). When users belonging to a particular tenant corresponding to a TSPACE instance use the provisioned tools they sign in on the plug-ins using unique tenant ID and password. After successful authorization, an access code is returned to the plug-in that is sent by the plug-in with every API call to TSPACE. The unique code is used as an identifier of the tool that sends the data. Every data input request from the provisioned tools corresponding to artifact, annotation or change ontology that is received by TSPACE is verified by *Tenant Authentication Monitor* component. A successful authentication generates a unique key, which is a key index of the first HashMap of our three-layered hierarchy of hash maps. This key corresponds to a specific tenant. This key is used to fetch an object reference (in-memory or persisted object) of TSPACE instance corresponding to the tenant. The TSPACE instance contains the second HashMap, which has reference to RDF graphs corresponding to artifact, annotation and change ontology. The data that is sent by the plug-in contains information of which ontology it belongs to. That information is used to fetch reference to the RDF data structures. RDF data structures corresponding to the artifact, annotation and change ontology grows as the number of artifacts produced by the tools in a TSPACE instance increases along with the performed activities. Parsing the whole RDF graph every time to get to a specific node had been a very inefficient solution. To overcome this bottleneck, we introduced third level maps corresponding to RDFs of each of the artifact, annotation and change ontologies. The third level maps contain the object representing each node of the RDFs (when RDFs are in memory) or database row identifier (when RDFs are stored in the database). The three levels of nested HashMaps fetched a unique element of the RDF in constant time irrespective of the size of RDFs. Having separate instances of TSPACE internal components for each specific tenant also helps to provide a logical isolation between the data belonging to different tenants and *Tenant Authenticator and Monitor* component secures access to tenant specific data structures. Each instance of TSAPCE contains ontologies and corresponding RDFs, information extraction services to support collaboration and awareness, and persistence elements.

Figure 40 shows a detail of *Logger* component. *Activity Tracker* sub-component observes the changes that are being made in RDFs corresponding to artifacts, annotation and the change ontology and logs this information in

Log Store. The log information can be used for many purposes, including but not limited to pricing and billing.

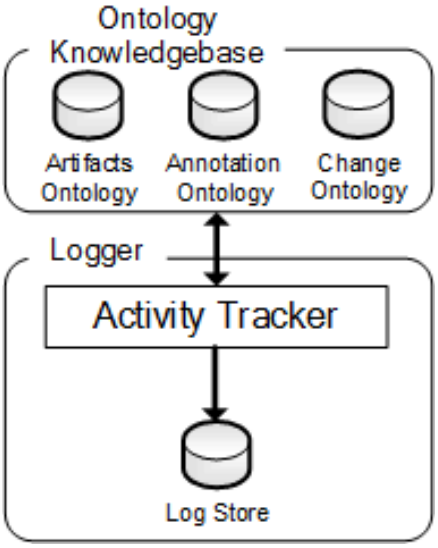


Figure 40: Log Management Component

6.4.3. Fourth Level Decomposition

This section describes the detailed decomposition of the important modules and components of TSPACE reference architecture. Multiple architecture and design patterns have been adopted for detailed design. Moreover, high-level abstractions of the important functions and Application Programmable Interfaces (APIs) have also been shown. However, only important functions are presented in the diagrams to void cluttering. The diagrams in this section are represented using Unified Modeling Language (UML).

6.4.3.1. Decomposition of Tools Selection and Provisioning Manager

Decomposition of Tools Selection and Provisioning Manager describes details of how the tools repositories are managed and how tools are selected and provisioned according to the defined parameters. Figure 41 shows details of the components and classes encompassing *Tools Selection and Provisioning* module. The façade of the modules have <<Service>> stereotype, indicating that the façade can be implemented as services to provide easy access to client applications.

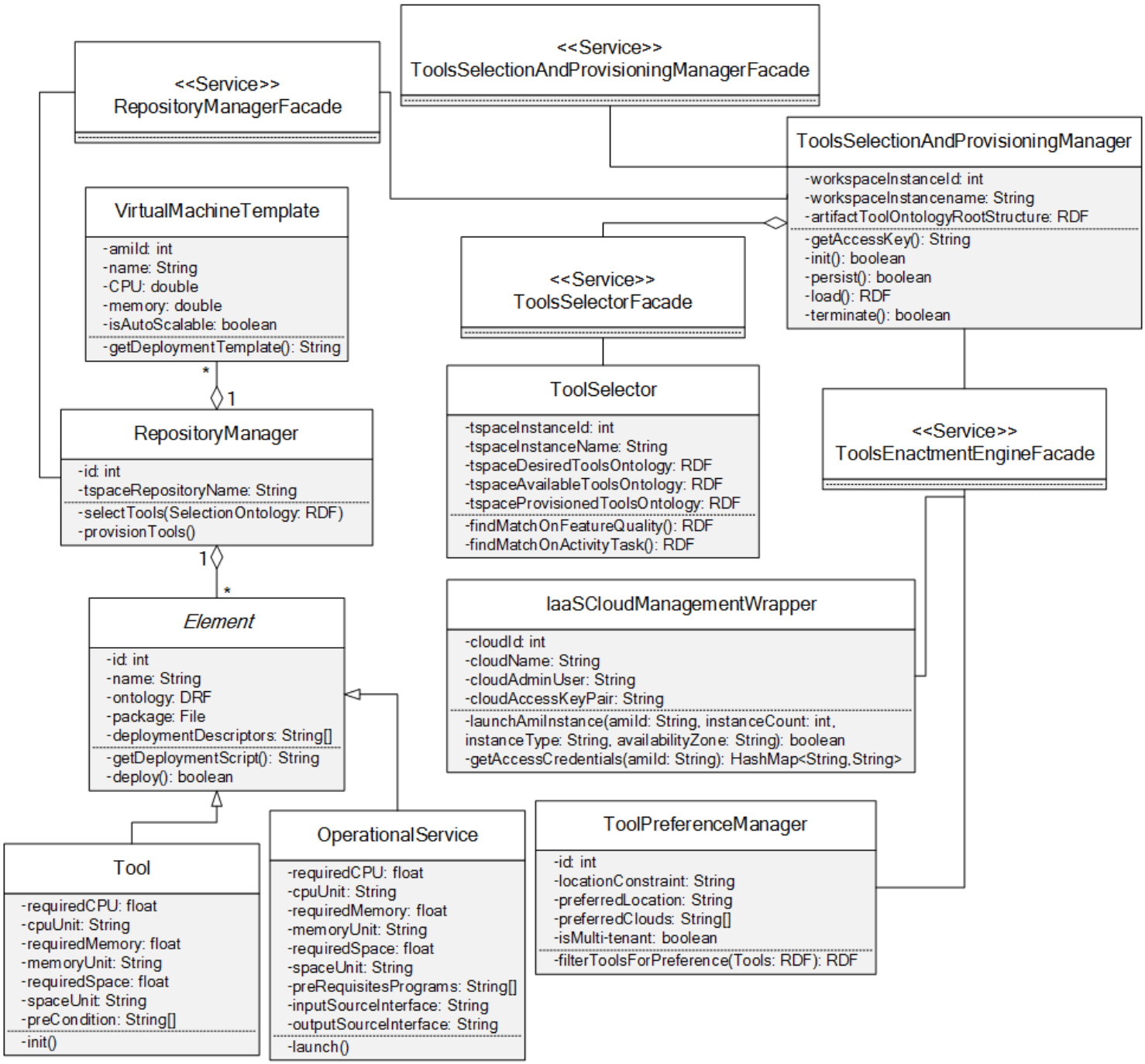


Figure 41: Tools Selection and Provisioning - Detailed Design

TSPACE repository consists of Virtual Machine Templates and Elements. *Elements* are an abstract representation of *Tools* and *OperationalServices*. As indicated earlier in this chapter, tools are attached to different nodes of the process whereas operational services are used to perform intermediate operations when artifacts are exchanged among the tools.

VirtualMachineTemplate is used to host the tools for provisioning. When a request is received for the enactment of a set of tools, the tools are selected by *ToolSelector* by establishing the closest match between the required tools and the tools that are available for provisioning, as described in Chapter 5. *ToolSelector* contains references to the ontologies that are used for tools selection and provisioning. Once a list of tools is selected for provisioning, the tools are provisioned in two different ways. (i) If the tools have deployment scripts associated with them and the tools can be deployed remotely (e.g. using Apache Ant [158] scripts), then the tools are deployed on virtual machines and provisioned. (ii) If the tools cannot be deployed remotely, then a preconfigured Virtual Machine (VM) template with a specific tool installed on it (e.g. Amazon Machine Images - AMIs) is used to provision the tool. A pre-configured VM template hosts only one tool. When more than one tool is required, multiple VM templates that are hosting the tools are instantiated. *ToolsSelectionAndProvisioningManager* fetches information from *ToolsSelector* and *RepositoryManager* and calls respective method of *IaaSCloudManagementWrapper* to instantiate and deploy the tool on underlying IaaS cloud. *ToolsPrefereneManager* takes care of enactment constraints (location constraints, constraints to choose a specific IaaS cloud to host the tools, and quality constraints on tool enactment e.g. to launch a separate instance of tool for every tenant etc.) of the tools.

Other than provisioning of the tools, Tools Selection and Provisioning Manager also needs to instantiate TSPACE artifacts, annotation and change ontologies according to a specific set of tools that are provisioned in a TSPACE instance. TSPACE initialization factory is represented in Figure 42, which shows a higher level of class hierarchy than is shown in Figure 41. Different elements of initialization factory are shown in Figure 42. For example *TspaceManager* is composed of *TspaceInitializer* and uses its methods to launch tools and ontology instances of a TSPACE instance. A detail of the methods and cardinality between different elements is shown in the figure.

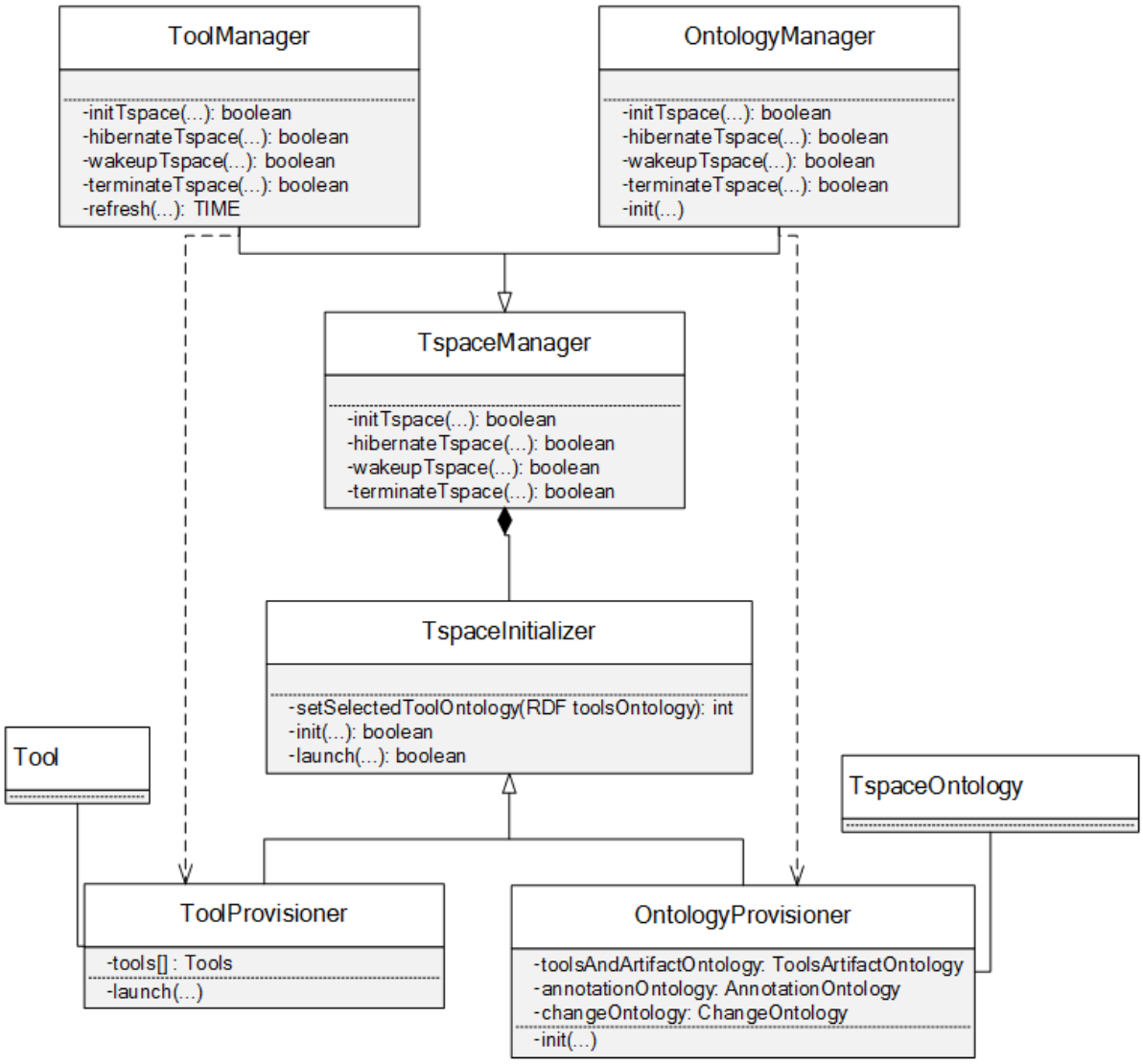


Figure 42: TSPACE Provisioning - Initialization Factory

6.4.3.2. Decomposition of Integration Manager

The decomposition of Integration Manager shows core elements of TSPACE integration mechanism and shows how specific tools can be integrated with TSPACE. The detailed elements are represented in Figure 43. In the figure, <<TSPACE>> stereotype shows elements of TSPACE whereas <<Tool>> stereotype shows elements of the tools that interact with TSPACE elements.

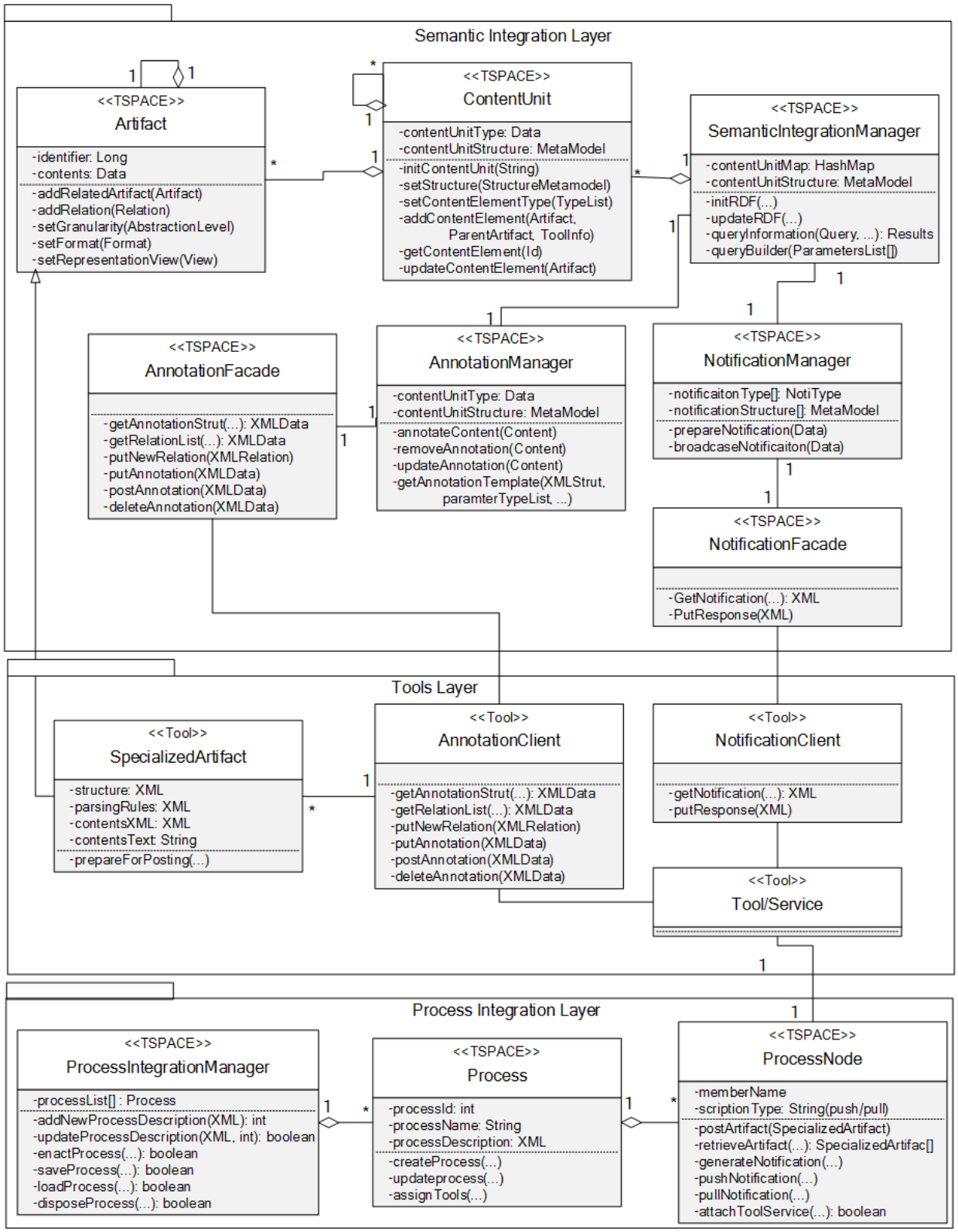


Figure 43: Integration Manager – Detailed Design

As TSPACE integration consists of semantic integration and process centric integration, Figure 43 shows two separate layers corresponding to each type of integration. *SemanticIntegrationManager* is at the core of semantic integration. It is composed of *ContentUnit*. *ContentUnit* represents a specific type of content in TSPACE. For example, in a TSPACE instance hosting three different tools: one for architecture knowledge management, one for architecture modeling and one to support decision making, there are three types of *ContentUnit* corresponding to each type of tool. A *ContentUnit* can be composed of multiple artifacts, which are represented as *Artifact* in Figure 43. *SemanticIntegrationManager* has one *AnnotationManager* and one *NotificationManager* associated with it, which exposes their interfaces to outside tools with their respective façade. These components use annotation ontology to annotate parts of the code and change ontology to track changes on the artifacts that are semantically related. The client services to utilize annotation and notification services can be written in the tools to access TSPACE respective features, which are represented as *AnnotationClient* and *NotificationClient* in Figure 43 in the tools layer.

The core component of the process integration layer is *ProcessIntegrationManager*, which can aggregate more than one process that is represented as *Process* in Figure 43. Each *Process* in turn can aggregate multiple nodes. The nodes are represented as *ProcessNode* in the figure. Once tools and operational services are attached to *ProcessNode*, these can post and retrieve artifacts as well as register for push notifications or use pull notification APIs. Once tools are assigned to a *ProcessNode*, TSPACE itself takes care of which tool is part of which process and handles artifacts and notifications accordingly. Detail of the methods and cardinality between different elements is shown in Figure 43.

The detail of Simple Storage Manager is shown in Figure 44. In the figure, a <<Service>> stereotype shows that methods of *SimpleStorageWrapper* can be exposed as service interfaces. <<Cloud>> stereotype shows that the elements are part of a specific cloud service provider. In the figure, the methods with keyword metamodel in these are used to post and update data associated with meta-model of the storage files such as file authors, file versions etc., whereas other methods are used to post, update and delete the files. watchFile() method allows to register for a notification when a particular file is updated or deleted.

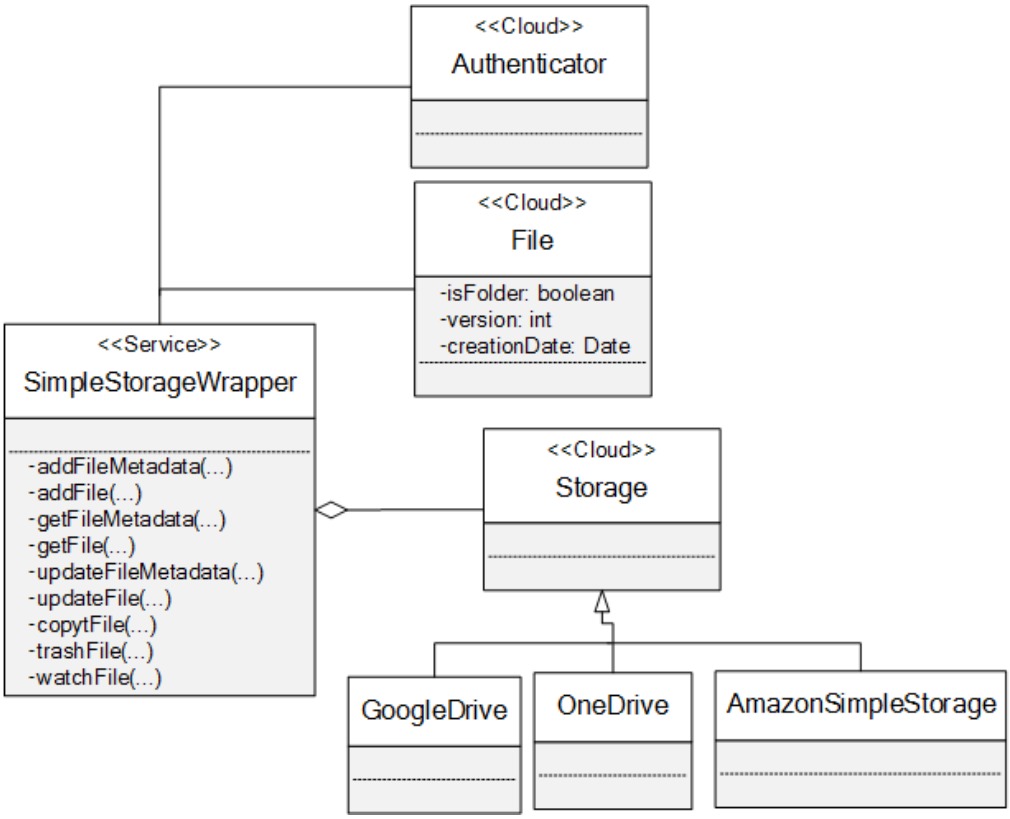


Figure 44: Simple Storage Manager - Details

6.4.3.3. Decomposition of Collaboration and Awareness Manager

The detail of Collaboration and Awareness Manager is shown in Figure 45. The elements in the figure that are marked by <<Service>> stereotype indicate that these elements can be implemented as services. *CollaborationAndAwarenessFactory* takes care of initialization of *Collaboration and Awareness Manager* by interacting with *NotificationManager*, *InformationDiscoveryManager* and *AnnotationManager*. As discussed in Section 6.4.2.3, notification and information extraction rules are used to fetch the desired information from ontology RDFs using SPARQL queries. *NotificationManager*, *InformationDiscoveryManager* and *AnnotationManager* interact with SparqlQueryExecuter to execute the queries on RDF data stores.

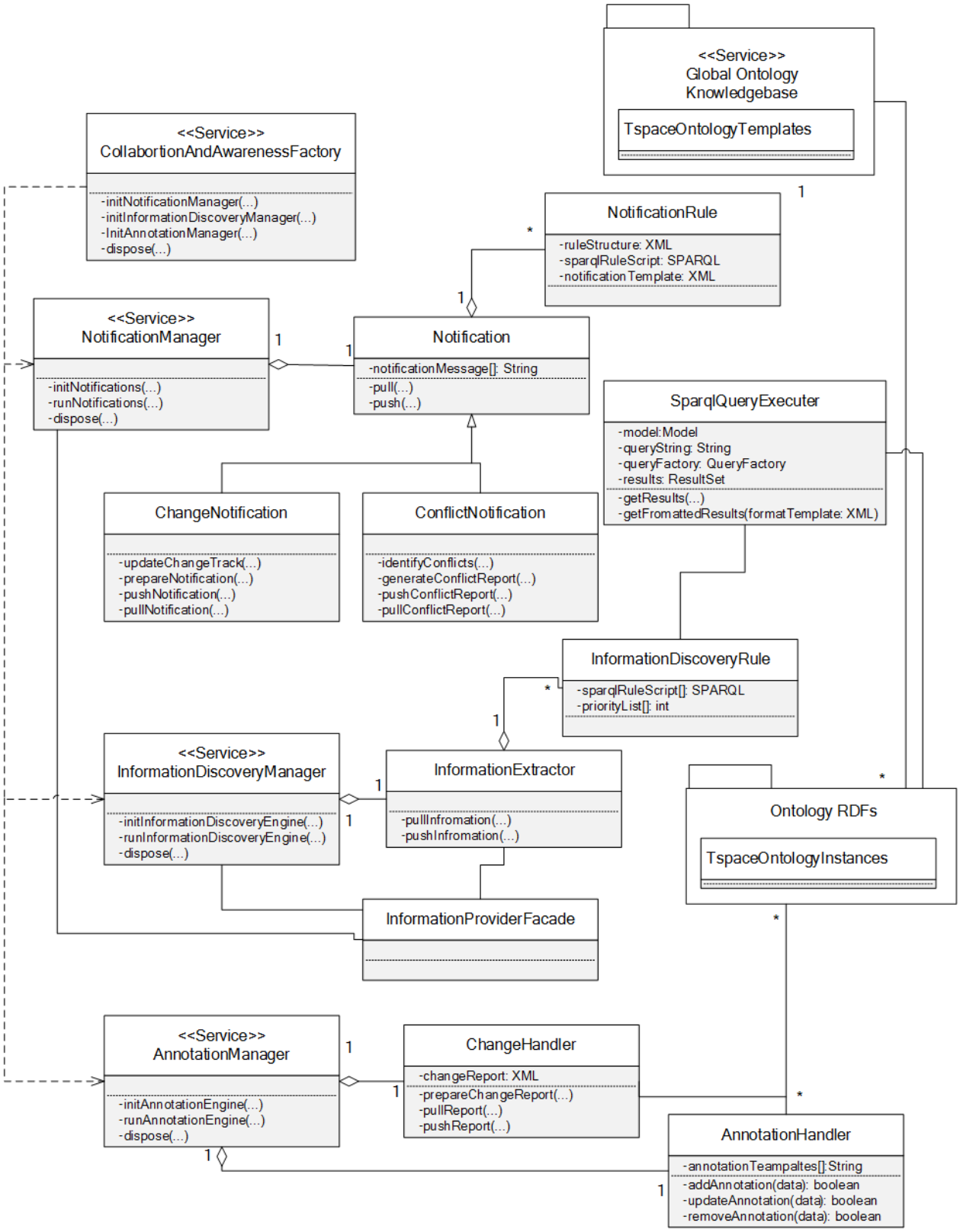


Figure 45: Collaboration and Awareness Manager - Detailed Design

NotificationManager can handle two types of notification: (i) *ChangeNotification*, which is triggered when a change is made in semantically integrated artifacts, and (ii) *ConflictNotification*, which is triggered when semantically integrated artifacts may present conflicting information. *AnnotationManager* facilitates semantic integration of the artifacts using annotations. *NotificationManager* interacts with *InformationDiscoveryManager* for SPARQL query execution and transformation of the extracted information to higher levels of abstractions. A detail of the methods and cardinality between different elements is shown in Figure 45.

6.4.3.4. Decomposition of Multi-tenancy and Authentication

The core of TSPACE multi-tenancy in combination with tenant and user authentication is presented in Figure 46. In the figure, the stereotype <<Service>> shows the elements that can be exposed as services, and the stereotype <<IaaSCloudService>> shows external IaaS cloud services that are used to complement TSPACE components. *CommonQueue*, *TenantQueue* and their corresponding façade present details of the data input streams queues as described in Section 6.3.4 and Section 6.5.3. Every *Tenant* can consist of more than one user. All the users belonging to a specific tenant can access TSPACE instance of that tenant. *Authentication* generates a unique authentication code for the tools that are provisioned via TSPACE. The authentication code is generated and sent to the tools when a user signs in a tool or virtual machine that is hosting the tool. The authentication code needs to be sent with every call to TSPACE APIs. The authentication code is based on OAuth protocol [159] and is only valid for a specific IP address for which it is generated. The authentication code is also used by the *FilterationRules* to identify the tenant when data is send by the tools to TSPACE. *ScalabilityController* (e.g. Amazon cloud watch and elastic load balancer [67-69]) is an external IaaS monitor that is used to replicate the queues according to defined parameters. Queues' façade provides a unified access point when queues are replicated. The important methods and cardinality between different elements of Multi-tenancy and Authentication are shown in Figure 46.

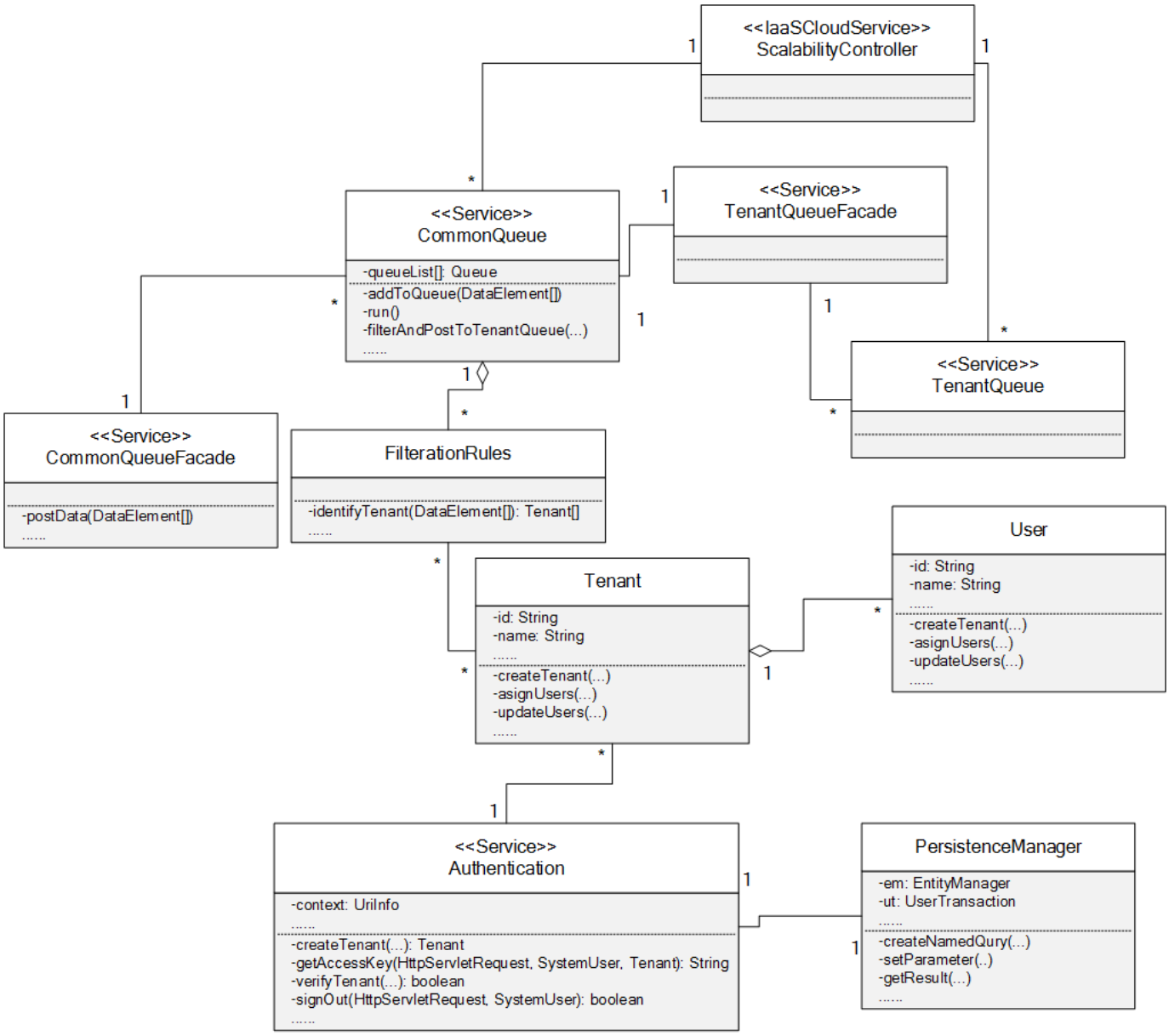


Figure 46: Multi-tenancy and Authentication - Detailed Design

6.5. Overview of Prototype Implementations

We have implemented a prototype of TSPACE reference architecture using JavaEE technologies. Interfaces of TSPACE prototype have been exposed as web services (REST and SOAP) using JAX-RS [160] and JAX-WS [161] service technologies. We have used Apache Jena Framework [162] to implement the semantic integration in the prototype. Persistence of TSPACE is handled following principles of Object Oriented Paradigm [163] and Java Persistence APIs (JPA) have been used to store data objects in an underlying database that is used for persistence. We have used jBPM core library [164] to

handle process workflow related features in TSPACE prototype. All the core TSPACE components and services have been deployed in GlassFish version 3.1.2.2 application server [165].

Amazon IaaS cloud [21] has been used to deploy TSPACE prototype and the tools that are provisioned by TSPACE. TSPACE deployment diagram on Amazon IaaS cloud is shown in Figure 47. The core components of TSPACE have been deployed on Amazon EC2 Windows Server 2012 instance [31] with 8GB of RAM and 2.4 GHz Intel Xeon processor. Amazon Cloud Watch [69] and Elastic Load Balancer [68] have been attached with the core services to enable auto scaling. Amazon EC2 instances and Amazon Machine Image (AMI) templates [31] have been used to host the tools that are provisioned by TSPACE. Amazon RDS for MySQL [166] have been used for persistence of the data objects. There is a Java Persistence API (JPA) [167] based wrapper that acts as bridge between TSPACE components and underlying database. Having a JPA wrapper also enables to easily replace the underlying database if TSPACE requires porting on a private or hybrid cloud infrastructures. We have used object representation of different elements (e.g. TSPACE meta-model shown in Figure 23 and ontology meta-model shown in Figure 25) of TSPACE reference architecture as persistence objects and JPA's object to relational mapping features are used. We have also used Amazon's Simple Storage Service (S3) for storing plain artifacts and data [168].

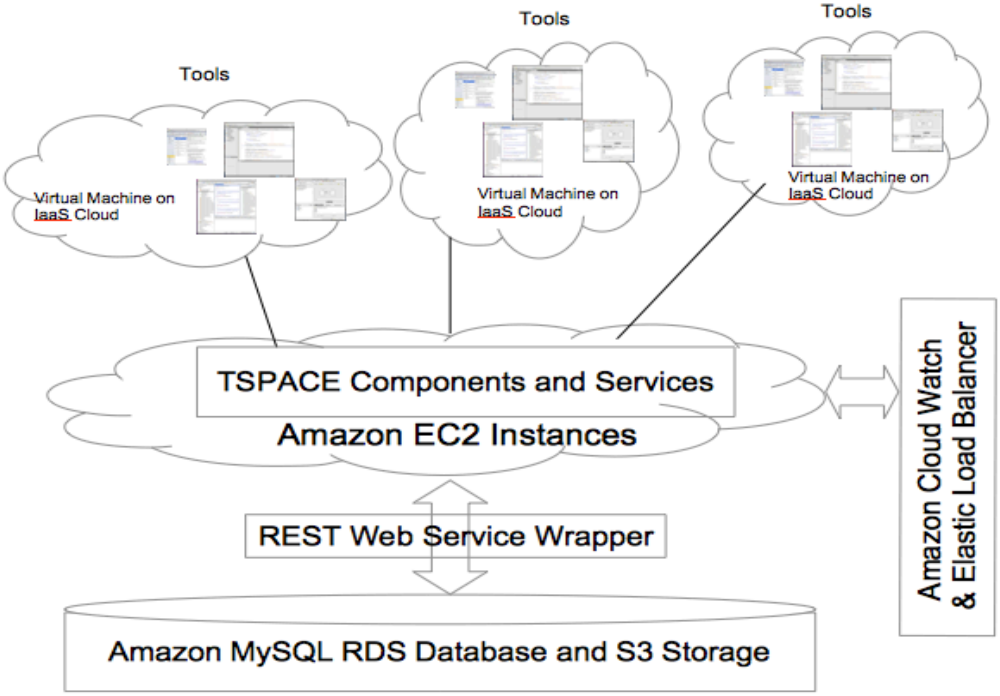


Figure 47: TSPACE Deployment on Amazon IaaS Cloud

In the following parts of this section, an overview of the GUIs and tools that are used for the proof of concept in the prototype implementation is provided. We have also shown code snippets of important methods in Listing 4 at the end of this chapter.

6.5.1. Administration User Interface

Figure 48 shows the administration Graphical User Interface (GUI) of TSPACE. The GUI is used to specify tools requirements in a TSPACE instance and specify notification that the users of the tools need. As indicated earlier, in TSPACE reference architecture we have focused on software architecting domain, hence the tools that are used in the prototype for proof of concept are related to software architecture requirements specification, architecture knowledge management, architecture analysis, architecture design and architecture modeling.

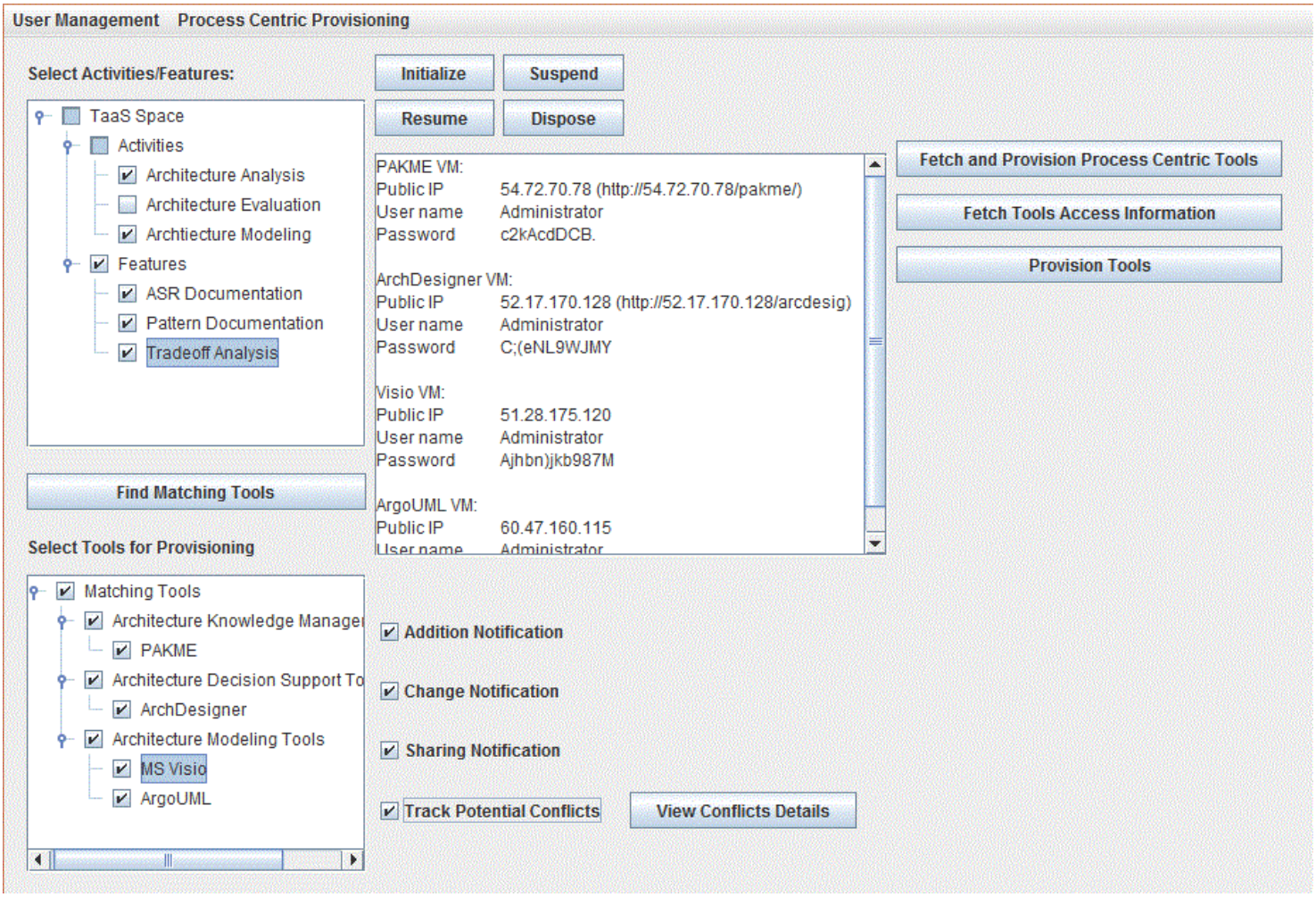


Figure 48: TSPACE Administration GUI

The GUI provides support for tools provisioning that is based on semantic integration as well as process-centric integration (e.g. based on a specific software architecting process).

Semantically integrated tools can be provisioned in two different ways via GUI. (a) The desired activities and features can be selected after which TSPACE provides the closest match of the tools that are available in a TSPACE and can be provisioned. Once desired activities and features are selected, a request is sent to TSPACE platform deployed on Amazon cloud by pressing *Find Matching Tools* button, as shown in top left of Figure 48. TSPACE selects the tools that adhere to the tools requirements criteria, as explained in Chapter 5 and returns the list of tools that are available for provisioning, as shown in button left of Figure 48. As the figure shows, PAKME, ArchDesigner, Microsoft Visio and ArgoUML qualify search criteria corresponding to specified activities and features as shown in the figure. After selecting the tools and pressing the initialize button, the tools are provisioned using Amazon EC2 cloud instances, and access information of the tools is presented. (b) The tools can also be directly selected for provisioning from the tree shown in bottom left of Figure 48. When the tools are provisioned, the ontologies are also initialized that are subsequently populated as the users perform different activities and operations using the tools. The detail of initialization and provisioning has been described in Chapter 5, Section 6.4.2.1, Section 6.4.2.2.1 and Section 6.4.3.1.

Process centric tools provisioning can be achieved by opening process manager from *Process Centric Provisioning* menu. The details of the tools provisioning and usage scenarios for both process-centric integration and semantic integration are described in following subsections.

6.5.2. Process Centric Provisioning and Integration

To demonstrate the process-centric tools bundling capabilities of TSPACE, an example scenario has been elaborated in which an architecture modeling tool is used to generate code templates from class diagrams, the code templates are consumed by an IDE to implement business logic in code, and an test instance is used to deploy the code for testing. Process-centric provisioning GUIs are opened from *Process Centric Provisioning* menu of TSPACE administration GUI that is shown in Figure 48. Figure 49 shows first GUI to create process. The GUI gives two options: either to create a new process or to select an existing process to add nodes in the process.

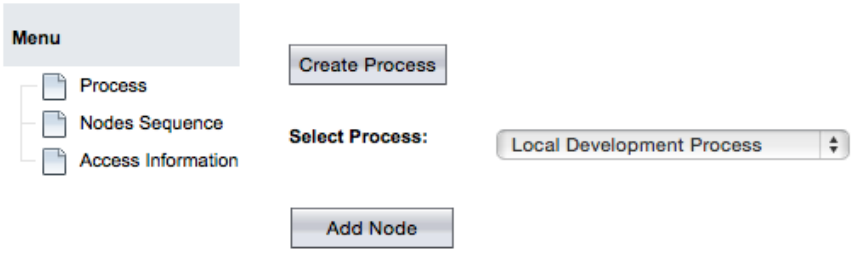


Figure 49: Process Centric Integration - Process Definition GUI

After clicking *Create Process* button, the create process GUI opens, as shown in Figure 50. As TSPACE provides an option to define nested processes, the GUI has fields to select parent process and a particular node of the parent process with which a nested process it to be created. After entering the name of the new process and pressing *Save* button, a new process is created.

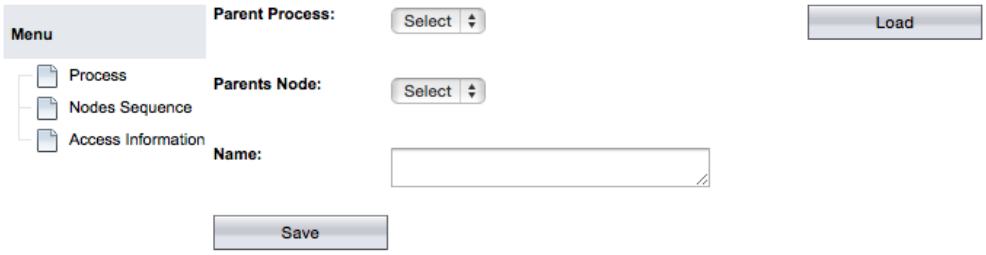


Figure 50: Create Process

Once a process is created, the next step is to define nodes in the process. After selecting the process and pressing the *Add Node* button at the GUI of Figure 49, the GUI to define new node is opened, which is depicted in Figure 51 and Figure 52. At each node, a specific tool can be selected and can be attached with the node. In Figure 51, ArgoUML tool is attached with Development Node 1. In Figure 52, NetBeans IDE is attached with Development Node 2. Similarly the tenants can also be attached with the nodes as ITUDev1 tenant is attached with Development Node 1 and ITUDev2 tenant is attached with Development Node 2. Similarly, a Testing Node 1 is created.

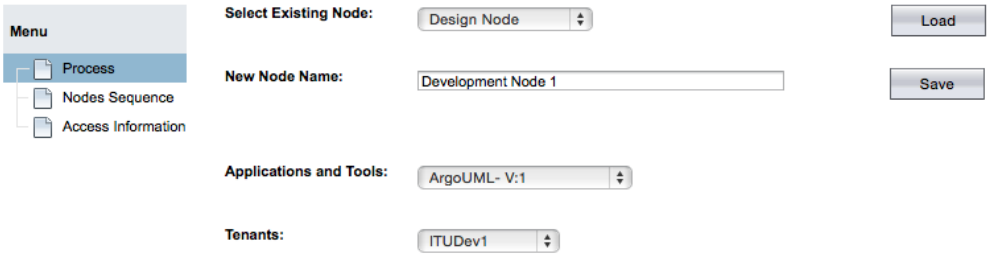


Figure 51: Process Centric Integration - Design Node

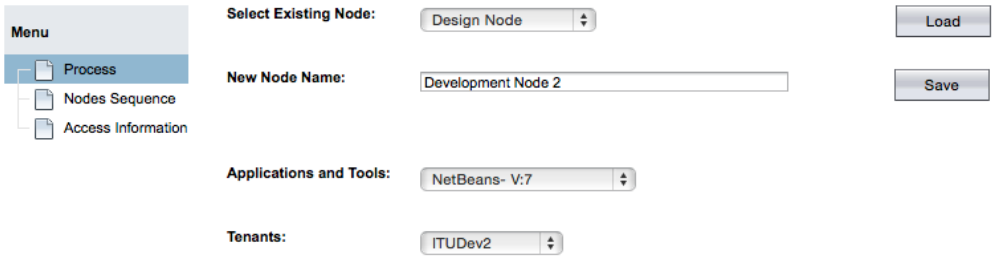


Figure 52: Process Centric Integration - Development Node

Once nodes are created, the next step is to define flow between the nodes. The GUI shown in Figure 53 provides a mechanism to define the sequence between the nodes of the process. By selecting a process name from the drop down menu againt *Select Process* label and pressing load button, the drop down menu grid is displayed. By selecting the sources and destination nodes, the sequence of process can be defined. For example, according to the sequence that is described in Figure 53, the output of Development Node 1 is used as input of Development Node 2 and output of Development Node 2 is used as input of Testing Node 1. As the artifacts flow from sources to destination, intermediate services can be attached to perform certain operations on the artifacts. In Figure 53, output of Development Node 2 is compiled using Java Code Compiler service and output of Java Code Compiler Service is passed on to Testing Node 1.

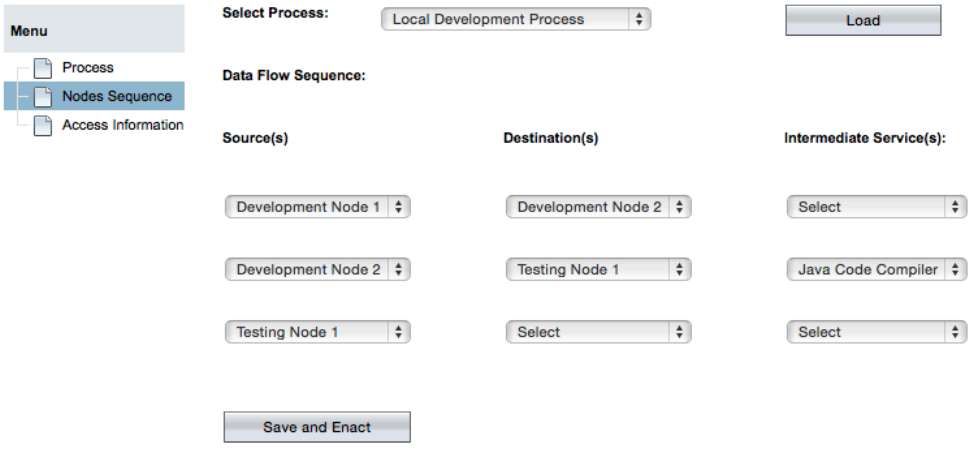


Figure 53: Process-Centric Integration - Artifacts' Flow Sequence

Once the sequence of the artifacts' flow between nodes is defined, by pressing *Save and Enact* button, the process is enacted and tools are provisioned. BPMN XMI [164] file that is generated for the process defined by the GUIs is used by *Process Workflow Engine* to enact the process (as described in Section 6.4.2.2.2). The tools access information if displayed, as shown in

Figure 54. *Fetch and Provision Process Centric Tools* button on the main GUI of Figure 48 also displayed tools access information.

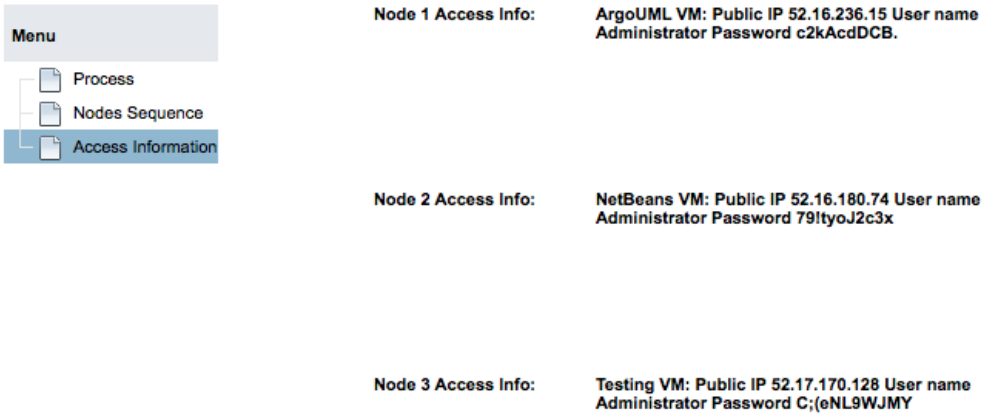


Figure 54: Process Centric Integration - Access Information

Figure 55 shows IaaSCloudManagementWrapper method implementation (described in Section 6.4.3.1) to use provisioning methods of Amazon IaaS cloud.

```
public void createAMInstancesExtended(String AMId, int min, int max,
        String keyPairName, String insType, String availabilityZone)
        throws Exception {
    RunInstancesRequest request = new RunInstancesRequest();
    request.setInstanceType(insType);
    request.setMinCount(min);
    request.setMaxCount(max);
    // set to zone
    Placement p = new Placement();
    p.setAvailabilityZone(availabilityZone);
    request.setPlacement(p);
    request.setImageId(AMId);
    // Set key pair for user..
    request.setKeyName(keyPairName);// assign Keypair name for this request

    RunInstancesResult runInstancesRes = this.ec2.runInstances(request);
    this.ReservationId = runInstancesRes.getReservation()
            .getReservationId();
    DescribeInstancesResult describeInstancesRequest = ec2
            .describeInstances();
    List<Reservation> reservations = describeInstancesRequest
            .getReservations();
    Set instances = new HashSet();

    for (Reservation reservation : reservations) {
        instances.addAll(reservation.getInstances());
        if (reservation.getReservationId().equals(this.ReservationId)) {
            this.instanceId = ((Instance) reservation.getInstances().get(0))
                    .getInstanceId();
            System.out.println("this.instanceId = " + this.instanceId);
        }
    }
    System.out.println(" " + instances.size()
            + " Amazon instance(s) created.");
}
```

Figure 55: Tool Invocation Method Wrapper for Amazon IaaS Cloud

Once tools are enacted and provisioned, these can be accessed either via virtual machines (for desktop-based tools) or via URIs (for web-based tools). Figure 56 shows ArgoUML and NetBeans IDE that are provisioned as part of the process centric tools provisioning and integration for the scenario that has been described in this section. We have implemented Java-based plug-ins for ArgoUML and NetBeans IDE to enable them for interaction with TSPACE core services. Once a user (belonging to a specific tenant that has been assigned in previous steps) signs in, the platform itself takes care of the sources and destinations of the artifacts. According to the defined process, ArgoUML generates code template based on the class diagrams and NetBeans consumes the code templates for further development. As depicted in Figure 56, the plug-in in ArgoUML shows only the destination because, in the defined process, there was no input node (tool) for the ArgoUML. The plug-in in NetBeansIDE, however, shows both sources and destination, as the output of ArgoUML is to be used as input of NetBeans IDE and the output of NetBeans IDE, after being compiled by Java Compiler Service, is to be used as input by Virtual Machine of Test Node 1. Test Node 1 has only the runtime environment for the code that is generated by NetBeans IDE node and has not been configured with a specific tool. The Check Status button of the plug-ins fetches and displays notifications from TSPACE (as discussed in Section 6.4.2.3 and Section 6.4.3.3).

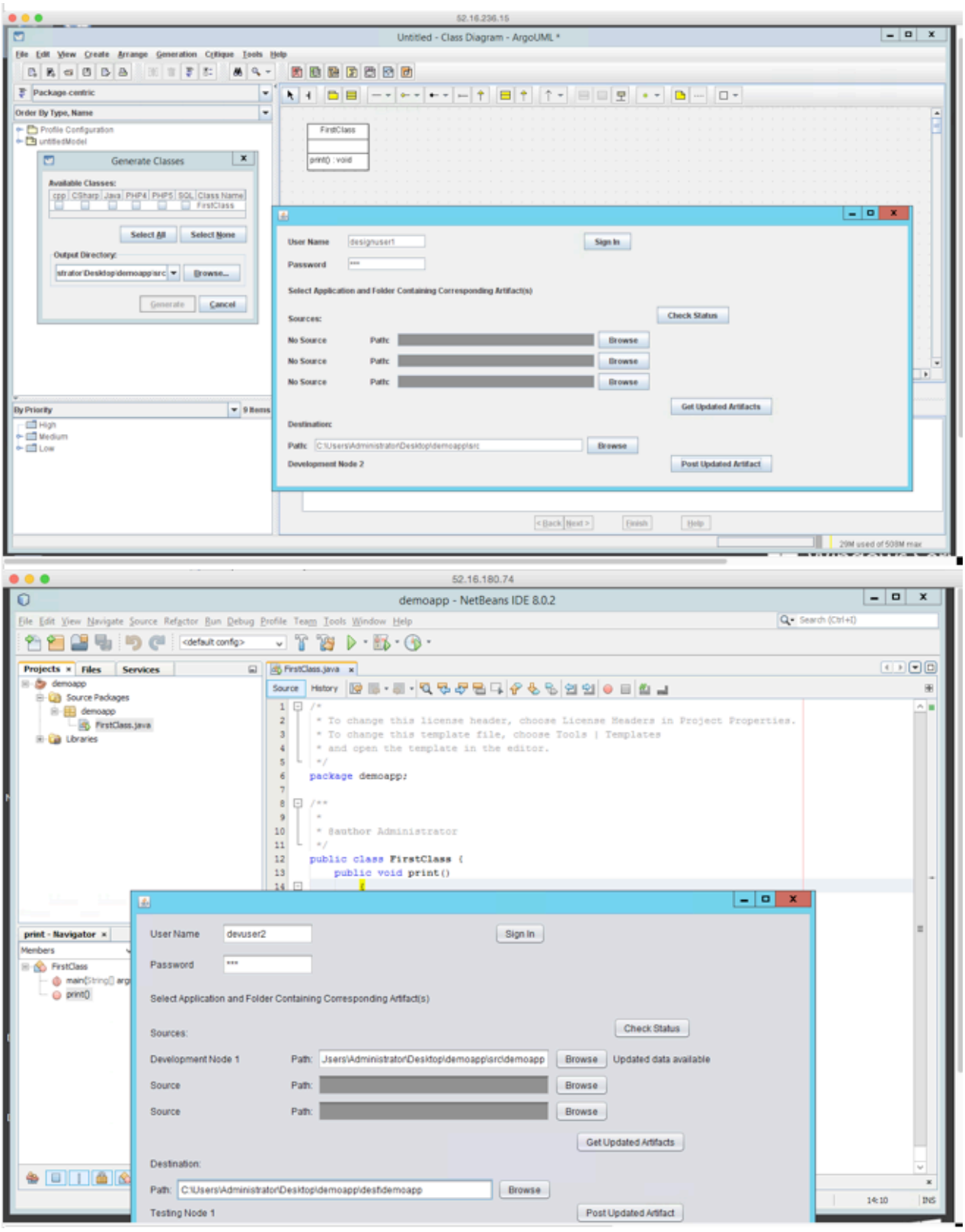


Figure 56: Process Centric Integration - Tools Provisioned and Hosted in VMs

6.5.3. Semantic Integration

The tools that are used in the prototype to demonstrate semantic integration capabilities of TSPACE deal with software architecting artifacts at different levels of abstraction. We have integrated one architecture knowledge management tool PAKME [107], a custom implementation of a decision support tool ArchDesigner, an open source UML modeling tool ArgoUML [104] and an architecture modeling tool Microsoft Visio [103].

In this section, we demonstrate how artifacts at different levels of abstraction that are produced and maintained by the tools are semantically integrated using TSPACE. We demonstrate an example use case in which some Architecture Significant Requirement (ASR) scenarios are captured (using ASR tool/module), decisions are made to select specific ASR (using decision support tool), architecture pattern have been selected to achieve ASR (using architecture knowledge management tool) in design and architecture patterns is modeled as detailed architecture design (using architecture design and modeling tools). Figure 66 in Listing D shows a code snippet of the authentication key generation mechanism. The authentication key is used for OAuth [159] authentication and tenant identification in TSPACE.

When a bundled suite of semantically integrated tools is provisioned by using administration GUI, as shown in Figure 48, TSPACE not only provisions the tools but also initializes TSPACE ontology structure that provides the backbone for semantic integration (Section 6.4.2.2 and Chapter 5). As described in Section 5.3.2 in Chapter 5, a Content Unit (CU) is created for each type of tool that is provisioned in TSPACE. All the tools-specific content units are aggregated by a root content unit, which corresponds to one instance of TSPACE (a specific set of tools bundled together and launched for a particular tenant). For a set of tools that are used in the prototype, four child content unites are created: two corresponding to PAKME (one for requirements and one for architecture knowledge), one corresponding to ArchDesigner and one corresponding to ArgoUML and Microsoft Visio. The artifacts that are produced and maintained by the tools are linked with their respective content unit. For example, the artifacts that are produced by PAKME are linked under PAKME content unit. A pictorial representation of this scenario has been depicted in Figure 20 in Chapter 5 and has been discussed in Section 5.3.2 in Chapter 5. Figure 65 in Listing D shows important parts of TSPACE initialization service method.

Figure 57 shows Architecture Significant Requirements (ASR) capturing the GUIs of PAKME tool. The GUIs are modified to incorporate annotation features of TSPACE, as described in Section 5.3.4 in Chapter 5 and Section 6.4.2.2.1. In Figure 57 two availability requirements and two scalability

requirements are shown. As shown in the figure, respective parts of the requirements are annotated with quality, value and metric annotation. Availability scenarios shown in Figure 57(a) and Figure 57(b) have different metric units but the same value. Scalability scenarios that are depicted in Figure 57(c) and Figure 57(d) have the same metric unit but different values of the metrics. When this information is saved in PAKME, the probes implemented in PAKME send ASR data and annotations to TSPACE. This information is added under requirements and scenario content unit (Figure 20 in Chapter 5). When TSPACE is initialized and tools are launched, different types of notifications can be configured as shown with check boxes in Figure 48. Conflict notifications are one of the notifications that can be configured. By pressing *View Conflict Details* button, the details of the conflicts can be viewed, as shown in Figure 58. Some sample notification rules are described in Table 33 in Chapter 5.

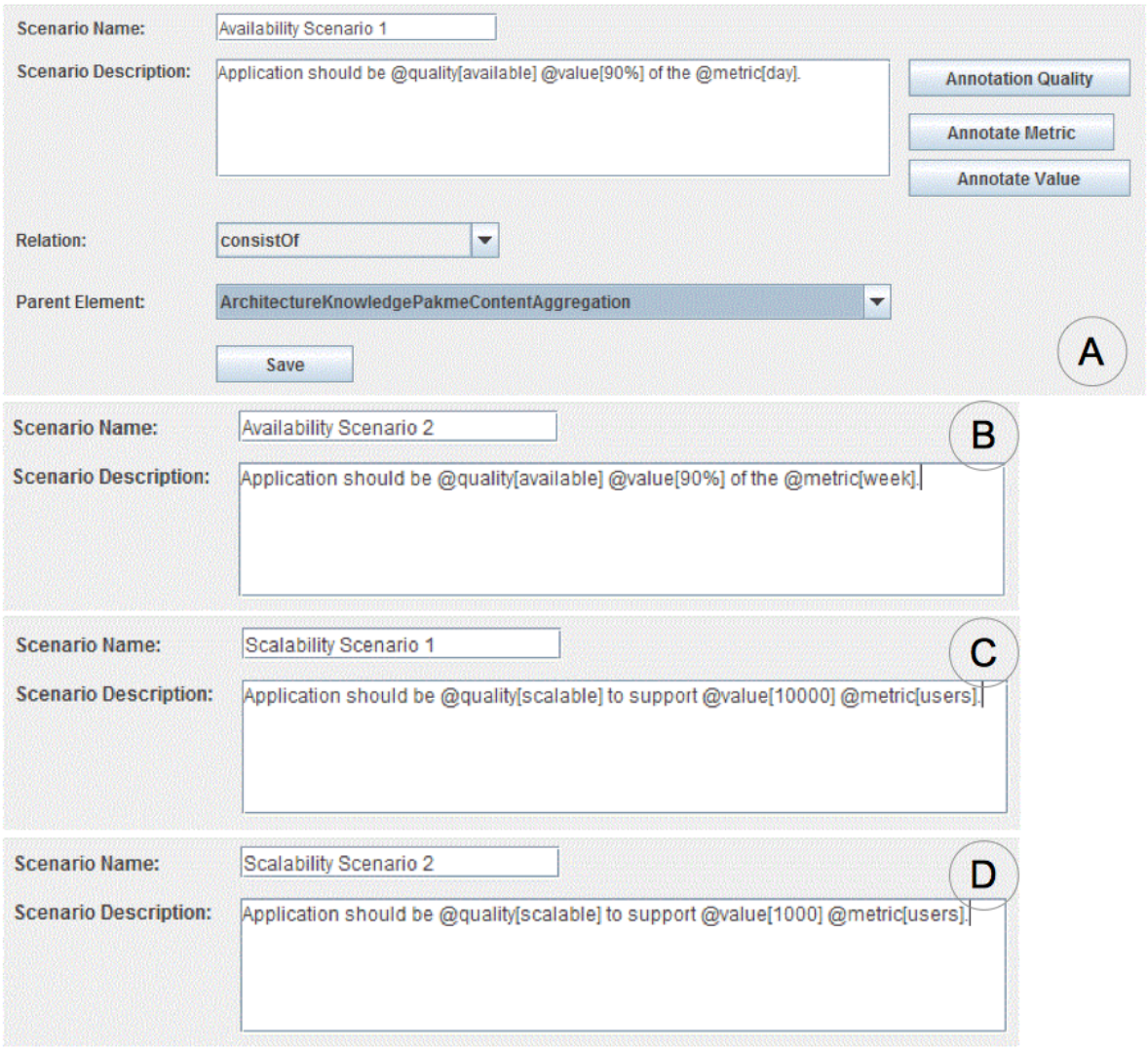


Figure 57: Semantic Integration - PAKME Architecture Significant Requirements GUIs

The conflict notifications that are presented in Figure 58 are generated by running SPARQL queries and complimentary algorithms, which look for same quality attributes in ASRs but with either different metric units or different metric values.

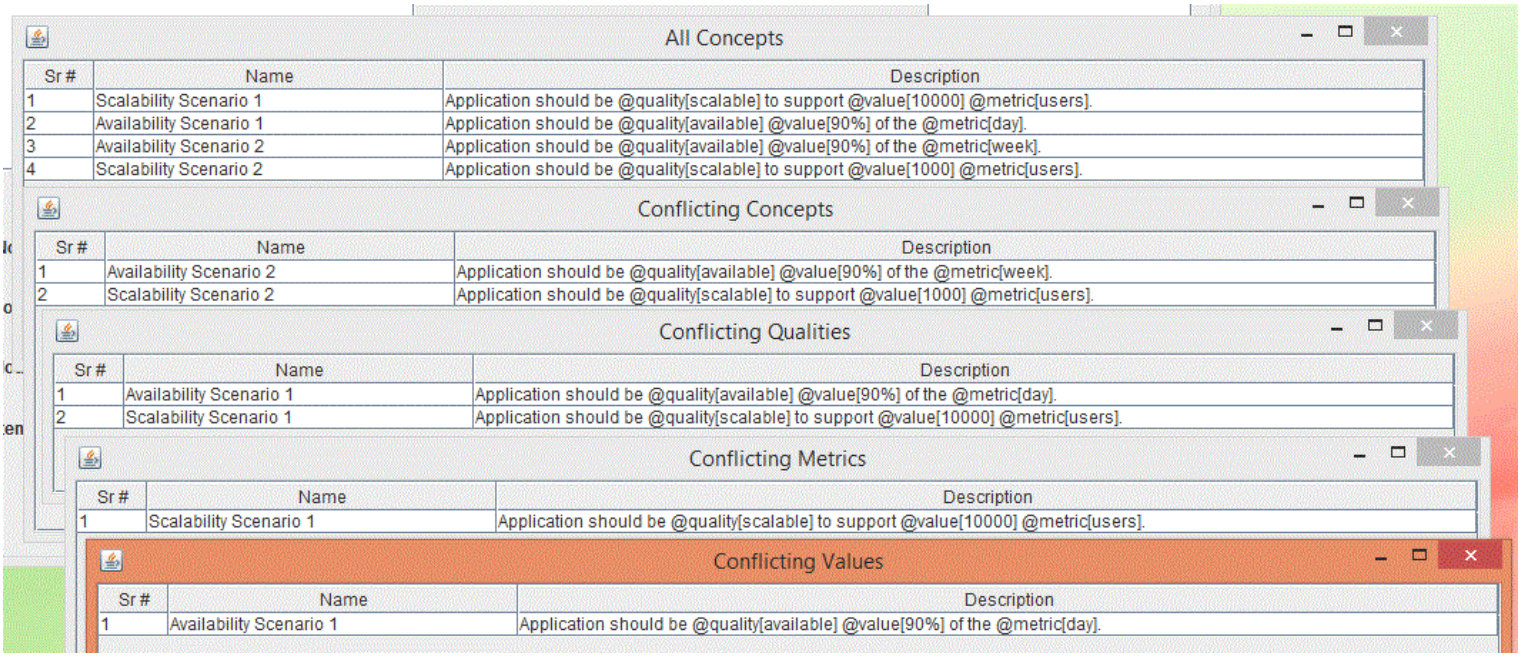


Figure 58: Conflict Notifications

The GUI in Figure 59(a) belongs to PAKME architecture knowledge management pattern documentation module and Figure 59(b) belongs to a Visio add-in that is used to relate diagrams in Visio with architecture patterns documentation in PAKME. When "Load Balancing" knowledge is saved in PAKME, it is added in TSPACE tools and artifacts ontology under *Knowledge* content unit (Figure 20 in Chapter 5). The architecture diagram generated in Visio is added to TSPACE tools and artifacts ontology under *Modeling* content unit (Figure 20 in Chapter 5). As shown in Figure 59(b), the architecture diagram can be related to "Load Balancing" knowledge via *isAssociated* relationship. The relationships are defined in TSPACE as elaborated in Section 5.3.2 and Table 32 (Chapter 5). Visio Add-in also receives notifications if information is updated in parent content, which in this example is "Load Balancing" decision knowledge. Figure 67 in Listing D shows a code snippet of parts of implementation to add artifacts (and data) in TSPACE.

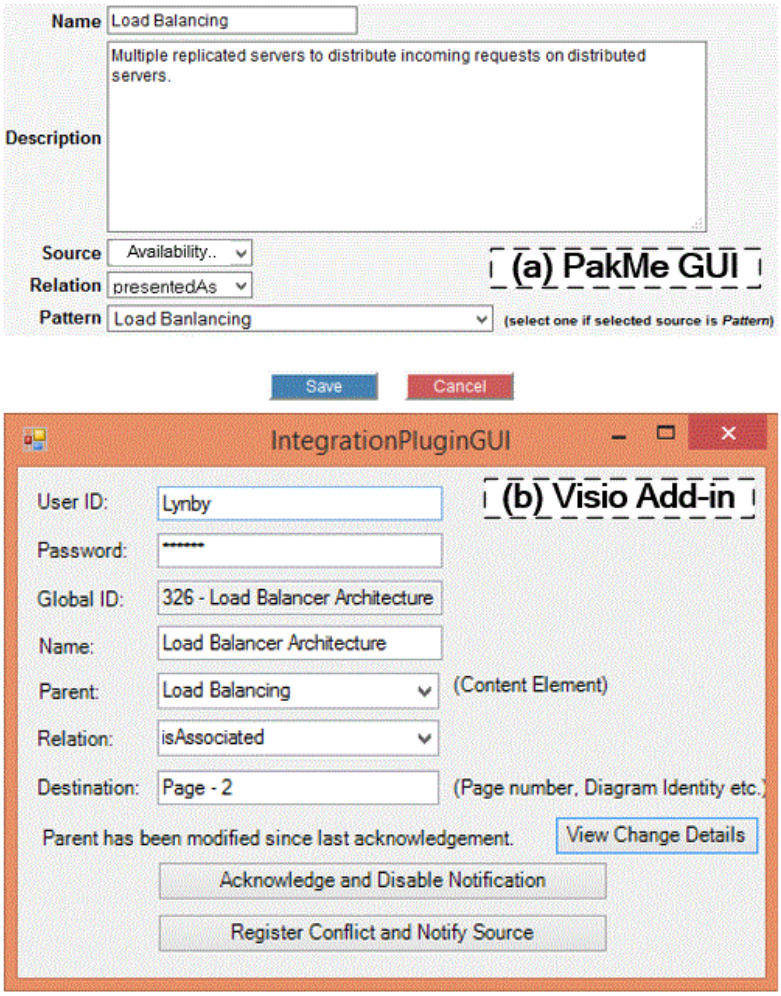


Figure 59: Semantic Integration - PAKME GUI and Visio Add-in

ArchDesigner decision support tool that is provisioned by TSPACE can be used to make decisions. Availability and scalability scenarios shown in Figure 57 have conflicts (as shown in Figure 58), and only one of the availability scenarios and one of the scalability scenarios can be correct. ArchDesigner GUI that is shown in Figure 60 can be used to make these decisions. The drop down menu corresponding to *concept* label fetches all the data from TSPACE. In Figure 60, Availability Scenario 1 is selected. Corresponding to *Relation* label, *value* relation is chosen. As indicated earlier, the relationships list is fetched from TSPACE (has been elaborated in Section 5.3.2 and Table 32). Once the decision is made, information is saved in TSPACE with *value* relation between *Availability Scenario 1* and the corresponding decision. The details of the TSPACE concept for which decision is to be made (which in our example is Availability Scenario 1) can be fetched from TSPACE by pressing *Fetch Details* button.

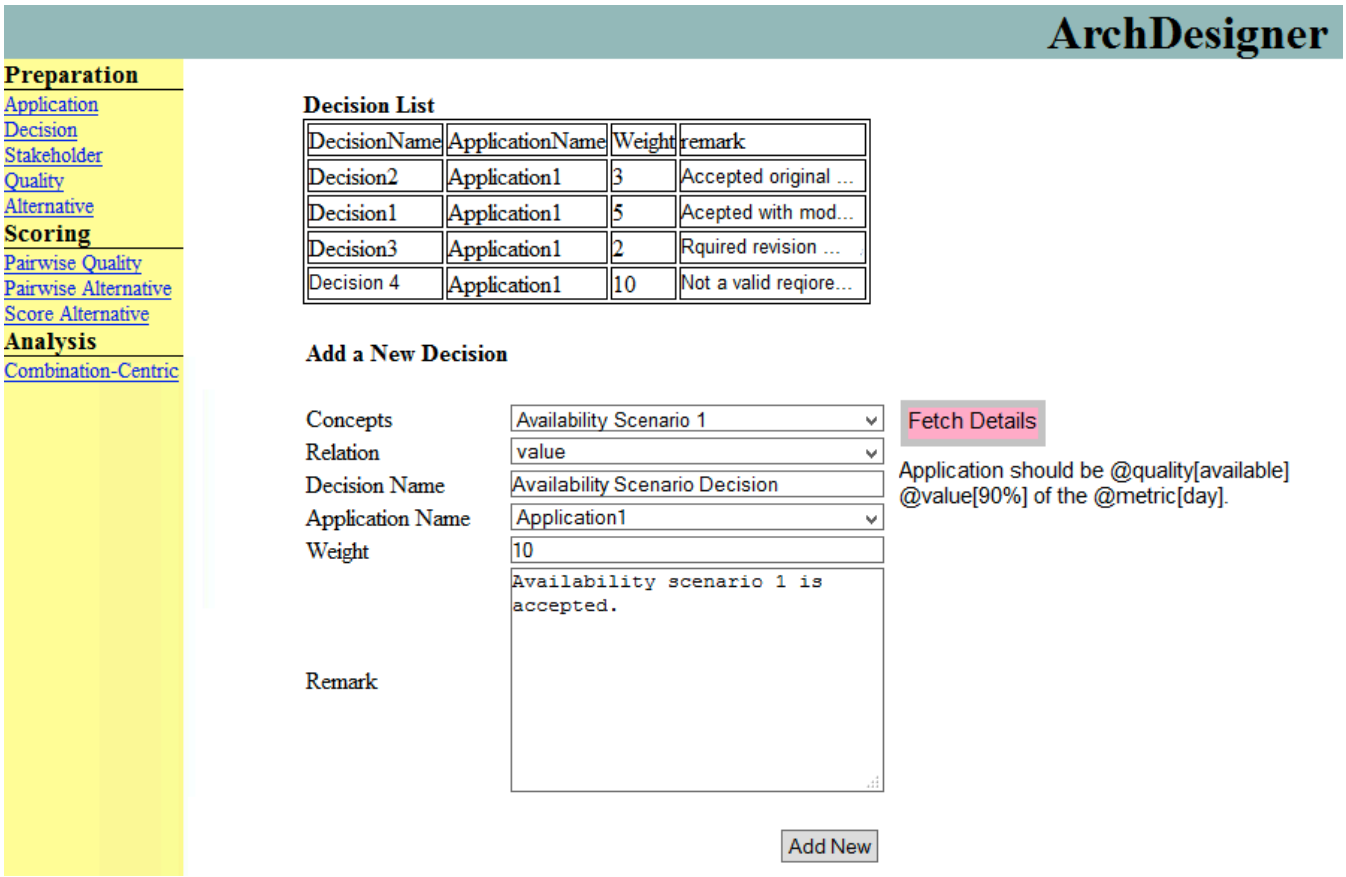


Figure 60: Semantic Integration - ArchDesigner Design Decisions GUI

An add-in has been added into ArgoUML so that when it is provisioned via TSPACE, the artifacts that are produced by it can be integrated with TSPACE semantic integration mechanism. Figure 61 shows integration of a detailed design of *Load Balancer class* with *Load Balancer pattern* via belongs to relationship. Figure 62 presents a summary of the sample semantic integration use case that has been discussed in this section. *TSPACE CU* is a root content unit (CU) and contains tools specific CUs. The artifacts that are generated and maintained by corresponding tools are mapped to CUs and relationships among the artifacts are managed via annotations. The details of the theoretical foundation have been described in Chapter 5.

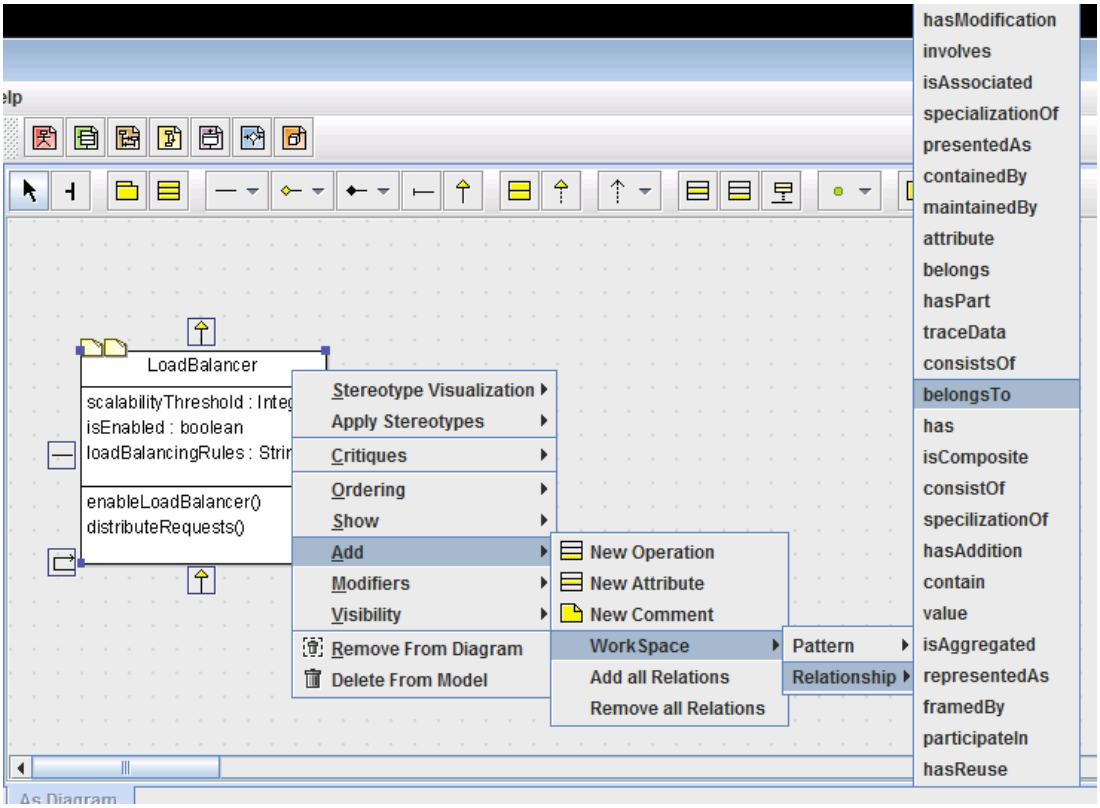


Figure 61: Semantic Integration - ArgoUML add-in

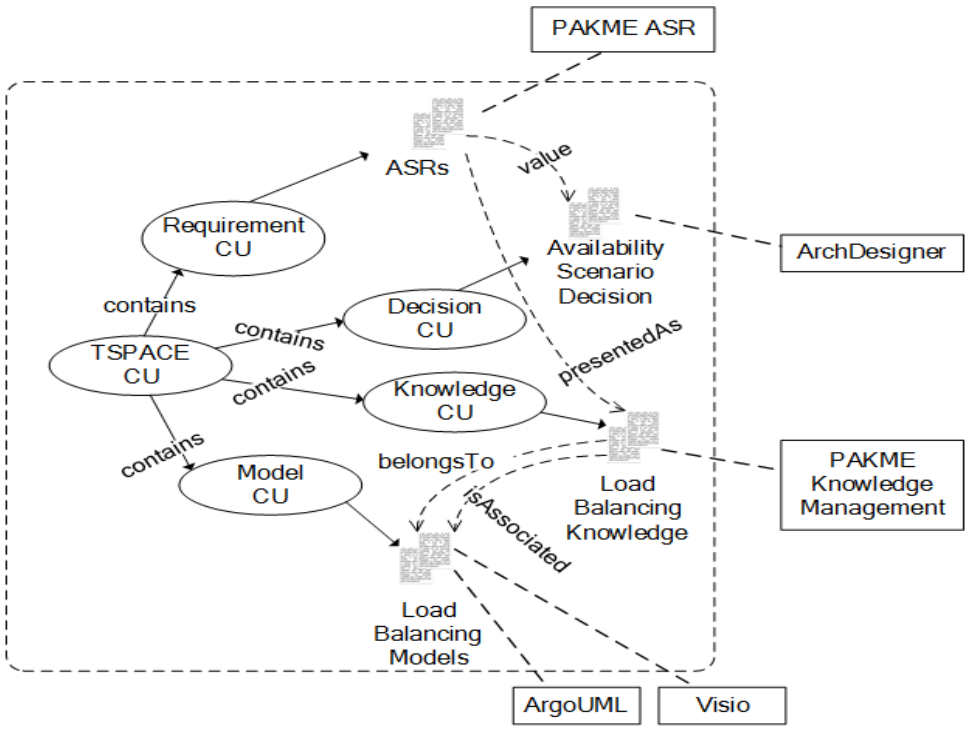


Figure 62: Semantic Integration Example Summary

Figure 62 shows a synthesized view of the semantic integration scenario that has been discussed in this section in terms of the relationship among the artifacts via TSPACE ontologies. The artifacts in textual formats are directly transformed into ontologies using TSPACE APIs by the Plug-ins, probes and Add-ins. The artifacts that correspond to architecture models, i.e. UML models generated by Visio and ArgoUML are converted into ontologies using a java-based UML2OWL library [169]. Figure 69 in Listing D shows an ontology corresponding to LoadBalancer design made in ArgoUML (shown in Figure 61). Figure 68 in Listing D shows a code snippet of incremental RDF generation (as discussed in Section 6.3.6) corresponding to TSPACE ontologies when an artifact (or data) is added or modified in TSPACE. Figure 70 in Listing D shows an abstract XML representation of TSPACE RDF ontology graph. It is to be noted that, because a large number of artifacts can be generated in TSPACE, all the ontologies RDFs are stored in Amazon MySQL RDS database using Apache Jena TDB persistence management component [170].

6.6. Evaluation of the Reference Architecture

An evaluation of a software architecture helps to identify its strong and weak aspects. As a reference architecture is aimed at serving as a guiding tool for diversified projects (based on same core idea) in context of multiple organizations, its evaluation prior to its adoption is of great importance. A positive evaluation of the reference architecture can facilitate its widespread adoption.

Multiple architecture evaluation methods have been proposed, including Software Architecture Analysis Method (SAAM) [12], Architecture Tradeoff Analysis Methods (ATAM) [13] and Quality-driven Architecture Design and Analysis Method (QADA) [14]. The choice of method to be used for the evaluation of a software architecture depends upon the goals of the evaluation activity and nature of the project. Literature provides some guidelines for the evaluation of the reference architecture [15-17]. However, to the best of our knowledge, there is no concrete method available for the evaluation of software reference architectures [17].

Hence, we have adopted a multi-facet strategy for the evaluation of TSPACE reference architecture. We have combined a classical reasoning approach with existing architecture evaluation methods. We have demonstrated feasibility and applicability of the reference architecture by implementing its prototype. We have also conducted an architecture evaluation session with experts from industry. In following subsection, we describe TSPACE reference architecture evaluation in detail.

6.6.1. Evaluation for Completeness of TSPACE Reference Architecture

In this section, we present the result of TSPACE reference architecture evaluation in terms of functional completeness of TSPACE. Use of scenario-based approaches such as SAAM [12] is a well established method for evaluation of completeness of an architecture. We have evaluated completeness of TSPACE reference architecture for functional requirements (FR1, FR2, FR3, FR4 and FR5) and have used scenario-based evaluation for non-functional (QR1, QR2, QR3, QR4, QR5, QR6 and QR7). We report the key reasoning points and outcomes of the evaluation decisions.

Table 36 shows the mapping of initialization and operational phases to high-level TSPACE components.

Table 36: Phases and Components Mapping

Phases	High-level Components
Tools Provisioning	Tools Selection and Provisioning Manager
Semantic and Process-centric Integration	Semantic Integration Manager, Process Integration Manager and Simple Storage Manager
Awareness and Collaboration	Collaboration and Awareness Manager

Table 37 shows the mapping between the lifecycle phases, functional requirements and corresponding components from decomposed architectural representations. It is clear from Table 37 that different parts of the reference architecture provide support for all the phases and corresponding requirements (Req).

Table 37: Activities, Requirements and Components Mapping

Activities	Req. ID	Detailed Components
Tools Registration	FR1, FR2	Tools/Capability Ontology, Tools Repository Manager.
Tools Selection	FR1, FR2	Tools/Capability Ontology, Tools Selector.
Tools Bundling, Enactment and Provisioning	FR3, QR1	Tools/Capability Ontology, Tools Enactment Preference Manager, Tools Enactment Engine, IaaS Cloud Management APIs.
Semantic Integration	FR4, QR5	Artifact Ontology, Annotation Ontology, Tenant Independent and Tenant Dependent Data

		Collection Queues, Data Monitor, RDF Data Store, Ontology Builder and RDF Generator, Annotation Mapper, Global Ontology Knowledgebase, Local Ontology Knowledgebase.
Process-Centric Integration	FR4	Process Engine, Process Workflow, Access Manager.
Awareness of the Operations and Collaboration	FR5	Annotation Ontology, Change Ontology, Information Extractor, SPQRQL Query Executer, Change Handler, Notification Manager.

We have presented TSPACE reference architecture in terms of its goals [171], which are transformed into functional and non-functional requirements, the TSPACE meta-model and its formalization using ontologies, different modules and components of the reference architecture at four levels of abstraction, and collaboration diagrams to show interactions between the components of the reference architecture. It covers all the important dimensions for reporting a reference architecture as per [15] and the views of Rational Unified Process [10]. It also positively addresses the completeness of the reference architecture (QR6). Our decision to use a layered approach supports separation of concerns among the components and high degree of modifiability (QR7). The Global Ontology Knowledgebase provides an abstract representation of the TSPACE ontologies and is a representation of the abstract data repository style. It not only achieves indirection in ontologies but also positively addresses flexibility (QR4) and Integration (QR5). Façade pattern is used at the interface layer to provide interoperability (QR5) between the tools and TSPACE reference architecture. Pipes and filter pattern is used to support scalability for handling ontology construction for multiple instances of a TSPACE and to support multi-tenancy (QR2) in the ontology-based semantic integration. The adoption of an ontology-based approach for tools selection, provisioning, integration, collaboration and awareness enables the reference architecture to be flexible (QR4), allowing support for heterogeneous tools and activities in a TSPACE instance. Although security is not explicitly considered, it is partially addressed with the help of Tenant Authentication and Monitoring components. Three level hash-map indexing guarantees faster access to data elements in TSPACE and positively address scalability (QR3). Having an ontology-based semantic integration approach facilitates adaptability of the reference architecture for different types of tool, as after transforming the artifacts into ontologies, the tools handling different types and abstraction levels of the artifacts can easily be integrated.

6.6.2. Evaluation of Feasibility and Applicability

Feasibility and applicability (QR6) of TSPACE reference architecture has been evaluated by implementing its prototype to provision different types of software architecting tools as a service, as described in Section 6.5. The tools that are used in the prototype have been selected based on following criteria.

- The tools that can be used to perform different types of activities related to software architecting.
- Each of the selected tools deals with artifacts at a different level of abstraction from other tools.
- When the tools are bundled together, the tools can be used to perform a set of software architecting activities that cannot be performed by using one single tool.

The prototype of TSPACE reference architecture has been deployed on Amazon IaaS cloud [21] and has been configured to use Amazon cloud resources for the deployment of the tools that are provisioned via TSPACE. As described in Section 6.5, to support architecture significant requirements capturing, architecture knowledge management, architecture analysis and architecture modeling, we have provisioned PAKME, ArchDesigner, ArgoUML and Microsoft Visio using the prototype. We have also provisioned NetBeans IDE on a cloud instance to demonstrate a process-centric integration workflow. TSPACE capabilities for tools provisioning, semantic integration, process-centric integration and awareness of the operation that are performed on the artifacts have been demonstrated with the help of a prototype. Complete detail of the prototype has been described in Section 6.5.

6.6.3. Evaluation via Potential Stakeholders' Participation

Architecture Tradeoff Analysis Method (ATAM) [13] supports evaluation for a wide range of architecture quality attributes. This method can be used to analyze software design strategies that have been used to incorporate architecture qualities in a software system's architecture. It also helps to identify potential conflicts, sensitivity points and tradeoff points in a software architecture [5]. While analyzing architecture for identifying sensitivity and tradeoff points, architecture scenarios (such as in SAAM [12]) are used. We have used ATAM to evaluate TSPACE reference architecting by conducting an evaluation session with potential stakeholders.

6.6.3.1. Evaluation Settings

We organized an architecture evaluation session with six software architects/designers. Although all participants were familiar with software

architecture evaluation methods and techniques, we provided them with an overview of ATAM and other architecture evaluation methods in a preliminary session. All the participants of the evaluation session have at least a university degree (one participant had a bachelor degree, four participants had masters degree and one participant had a PhD degree) in Software Engineering/Computer Science. The participants of the evaluation session had detailed knowledge of design and development of cloud-enabled software systems. As TSPACE reference architecture deals with different dimensions of TaaS including both semantic and process-centric integration, the participants were invited in the evaluation session who could be a good combination for analysis of all aspects of TSPACE reference architecture. Expertise of the participants were as follows: two participants had expertise in designing and developing workflow-based tools and system, two participants had experience with designing and developing software engineering tools for distributed software development, and two participants had experience with design and development of cloud-based and web-based applications. The experiences and expertise of the participants are summarized in Table 38. All the participants were given a document describing TSPACE reference architecture requirements, architecture design decisions and solutions a week before the evaluation session.

Table 38: Participants' Software Architecting and Development Expertise

Participant Identity	Software Design and Development Experience (Years)	Expertise
P1	11	Workflow-based tools and Systems
P2	11	Workflow-based tools and Systems
P3	8	Tools to support distributed development teams and distributed systems
P4	5	Tools to support distributed development teams, SaaS applications and web applications
P5	5	SaaS applications and web applications
P6	2	SaaS applications and web applications

Table 39: Questionnaire Used in Evaluation

ID	Description
Q1	To what extent (detail) TSPACE functional requirements cover drivers for providing software architecting Tools as a Service?
Q1-D	What additional requirements do you think should be addressed by the reference architecture? Please describe in sufficient detail.
Q2	To what extent are quality attributes that are considered for the quality of TSPACE as well as quality of the reference architecture relevant to runtime and design time quality of TSPACE?
Q2-D	What additional quality attributes do you think should be included? Please mention the quality attribute name and your rationale i.e. why you think that it should be included?
Q3-D	Are there scenarios/requirements (that are discussed during the presentation) that you think are not relevant for TSPACE?
Q4	To what extent are the design decisions that are taken to address the design time and runtime quality of TSPACE architecture relevant to TSPACE?
Q5-D	Please indicate if there are risks, sensitivity points and tradeoff points that are important for TSPACE but are not considered while designing the reference architecture. Please identify each risk in a separate bullet point and indicate your rationale why it should be considered.
Q6	To what extent are the functional requirements and quality characteristics addressed in TSPACE reference architecture?
Q6-D	Please provide details about which requirements are not addressed in the reference architecture.
Q7	To what extent the presented reference architecture addresses challenges that are associated with providing TSPACE in a general context of TaaS? I.e. For other domain other than software architecting.
Q7-D	Please provide your comments/details if additional dimensions should be considered in the reference architecture.

In the beginning of the evaluation session, an introduction and context of TSPACE reference architecture was presented to the participants. After a particular activity of the evaluation session was conducted, the participants were given an evaluation form to give feedback on TSPACE reference architecture corresponding to the evaluation activity. First, the participants were presented with TSPACE functional and quality requirements and were asked to provide their feedback on them. Then different design decisions corresponding to the requirements were described, followed by an exercise of identifying sensitivity and tradeoff points, and building a utility tree. Finally, the applicability of TSPACE reference architecture in broader context of TaaS

(not only software architecting tools, rather on applicability of tools in general that can be used for software design and development activities) was discussed.

Table 40: Evaluation Scale corresponding to Questions

ID	Scale
a	Very Low
b	Low
c	Medium
d	High
e	Very High

The questions that have been used in the evaluation questionnaire are listed in Table 39. In the questionnaire, there were two types of questions: (a) the questions (Q1, Q2, Q4, Q6, Q7) those had a qualitative scale to be chosen for their answers, (b) and the questions (Q1-D, Q2-D, Q3-D, Q5-D, Q6-D, Q7-D) those had descriptive answers. Table 40 lists the options of the qualitative scale for Q1, Q2, Q4, Q6 and Q7.

6.6.3.2. Evaluation Results

Q1 and Q2 aimed to seek input on functional and quality completeness of TSPACE reference architecture. Q6 aimed to identify to what extent the solutions that have been proposed in TSPACE reference architecture address the stated requirements and quality characteristics. Q7 aimed to identify the relevance of TSPACE reference architecture for the general tools (not only software architecting tools) that can be used to perform software engineering activities. The results of the questions (Q1, Q2, Q6 and Q7) are shown in Table 41. The results show that on average a high value score (corresponding to the questions) was chosen by the participants.

Table 41: TSPACE Evaluation corresponding to Quality Scale

Participant	Questions			
	Q1	Q2	Q6	Q7
P1	d	d	e	d
P2	d	d	e	c
P3	d	d	d	d
P4	d	d	d	c
P5	d	d	d	d
P6	d	e	d	d

Table 42: Design Decision Ranking

Design Decision	Participant					
	P1	P2	P3	P4	P5	P6
	Ranking					
TSPACE meta-model	d	e	e	e	e	c
Use of ontologies	e	d	d	d	d	d
Use of SOA (SOAP and REST)	d	d	e	d	e	e
Shared repository templates (shared repository pattern)	e	e	e	d	d	e
Tenant neutral queues and tenant specific queues and filters	d	e	d	d	e	e
Layered architecture style	d	e	d	e	e	e

Q4 aimed to identify effectiveness of the important design decisions that had been made during the design of TSPACE reference architecture. During the evaluation session, some of the key design strategies (that have been discussed in Section 6.3) were presented to the participants of the evaluation session, and the participants were asked to rank the design decisions and strategies according to the scale described in Table 40. The results in Table 42 show that on average all the design decisions were ranked high.

Questions Q1-D and Q2-D were aimed at identifying whether there were additional functional and quality aspects to be incorporated in TSPACE reference architecture. The feedback of the session participants has been incorporated in TSPACE reference architecture. For example, one of the concerns was to have more details on multi-tenancy, security and integration features of TSPACE. Fourth level decomposition (Section 6.4.3) has been added in the reference architecture to provide more details on the important components of TSPACE reference architecture. There was no concern raised against Q6-D.

Q5-D aimed at identifying the risks, sensitivity points and tradeoff points [5] in TSPACE reference architecture. The identified risks and tradeoff points were mainly related with the performance of TSPACE. A risk that was identified during the evaluation session was comprehensiveness of the ontologies to capture artifacts and different types of tools (other than software architecting tools) in TSPACE. This risk is mitigated in the reference architecture by providing flexibility to extend the ontologies and having the possibility of adding new ontology templates if needed (as discussed in Chapter 5 and Section 6.4). While discussing sensitivity and tradeoff points, it

was determined that Queues (Section 6.3.4 and Section 6.4.2.2.1) and shared repositories (Section 6.3.3) could become a bottleneck to the performance when a large number of tenants are to be served by TSPACE. These risks can be mitigated with the help of scalability features of hosting IaaS cloud (by replicating queues and repositories and by having automated scalability and load balancing components as discussed in Section 6.3, Section 6.4 and Section 6.5). Queues and shared repositories were also identified as tradeoff points (tradeoff between unified access point and scalability). Tenant-independent and tenant-specific queues along with the respective façade (Section 6.4.2.2.1 and Section 6.4.3.4) are needed to provide a single point of access, and shared ontology repositories are needed to provide a common ontology knowledgebase.

6.6.3.2.1. Utility Tree for TSPACE Architecture and System Qualities

During the evaluation session, two utility trees were constructed. Figure 63 shows the utility tree to discuss architecture design qualities of TSPACE reference architecture (completeness, feasibility, applicability and modifiability), whereas Figure 64 shows system qualities of TSPACE reference architecture. Completeness of TSPACE reference architecture is measured in terms of completeness of functional and quality requirements as well as completeness of the reference architecture elements (high-level and detailed components) that achieve functional and quality requirements.

Feasibility of the reference architecture is evaluated in terms of structure and conceptual integrity as well as by implementation of its prototype. Applicability of the reference architecture is analyzed by demonstrating provisioning of different types of tool in TSPACE and by supporting integration among the artifacts of different levels of abstractions. Moreover, TSPACE applicability for different types of tools was also analyzed during the evaluation session. Layered and components-based architecture enable addition of new components, enhancements in integration approach and incorporation of different IaaS clouds in TSPACE.

TSPACE system qualities (shown in utility tree of Figure 64) deal with runtime qualities of TSPACE reference architecture. These include automated provisioning, multi-tenancy, scalability, security and integration. Although security is not explicitly discussed while describing TSPACE requirements, it is considered while designing TSPACE reference architecture to achieve multi-tenancy. Design decisions to achieve TSPACE system qualities have been shown in Figure 64. The detail of design decisions has been discussed in Section 6.3, Section 6.4 and Section 6.6.1.

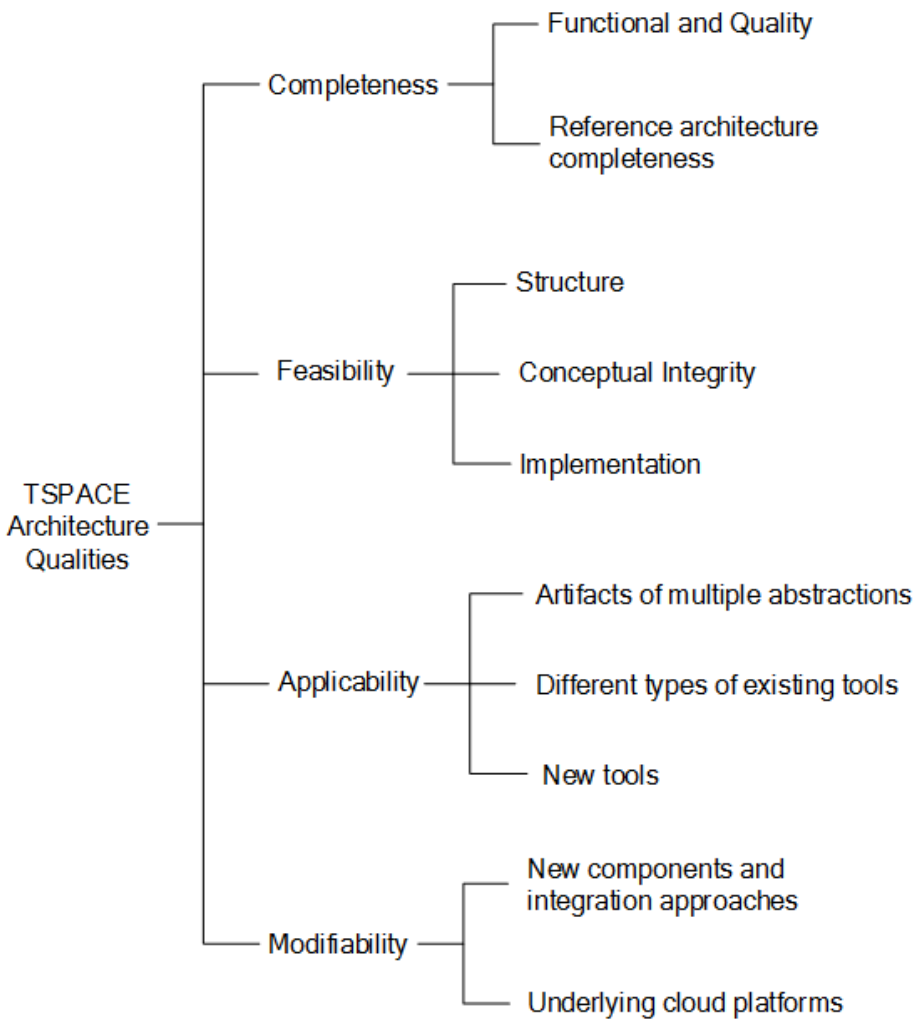


Figure 63: TSPACE Reference Architecture Quality Utility Tree

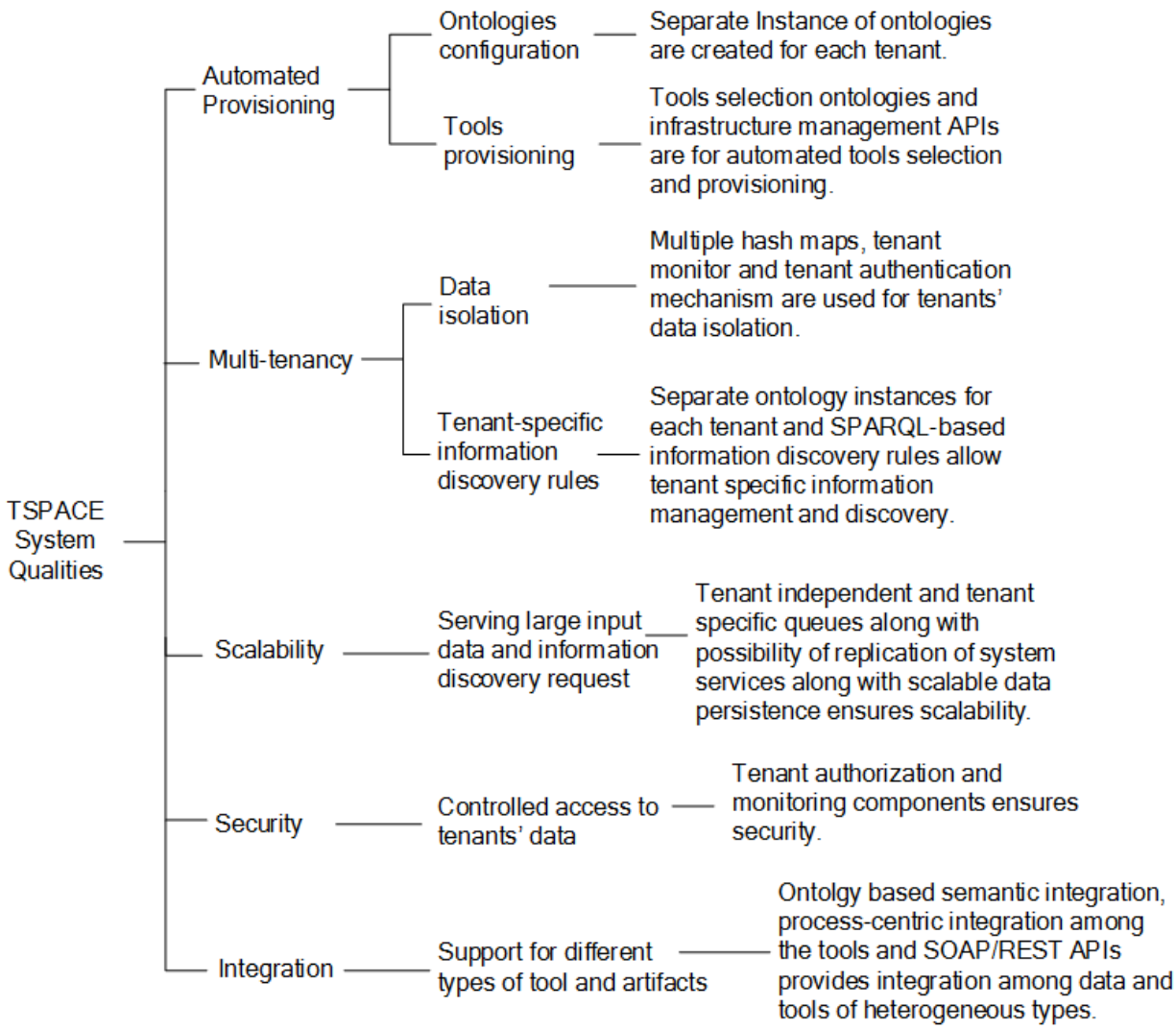


Figure 64: TSPACE System Quality Utility Tree

6.7. Related Work

Some efforts have been made to report the architecture of cloud-based tools but none provides a coherent solution covering all the required dimensions. Calvo et. al propose an architecture for textual information retrieval from cloud-based collaborative writing tools [172] but their effort is limited to support automated feedback and process analysis of students' academic assignment write-ups. Oliveira and Nakagawa propose a Service-Oriented Architecture for software testing tools [173]. Their work provides the detail on architectural requirements and a layered model to map tools onto the business process but does not cover a complete lifecycle of tools provisioning and operations. Integration approaches using service and graphical user end points have been reported in [174, 175]. An extensible architecture description

language (xADL) to support integration among architecture-centric tools is presented in [176]. Zhao et al. provide a survey of ontologies that have been proposed for software engineering [110]. We have proposed TSPACE ontology meta-models for the reference architecture because the reported software engineering ontologies do not satisfy the specific needs of TSPACE. There are also commercial offerings of cloud-based tools such as Cloud9 IDE [145] and a diagramming tool Griffy [146].

In comparison to the discussed existing work in the area, the TSPACE reference architecture has been designed not only to support on demand tools provisioning but also to enable bundling of tools based on stakeholders' needs and to provide a mechanism to raise awareness of the operations that are performed on the artifacts as a result of stakeholders' activities in a bundled suite of tools. TSPACE reference architecture also supports process centric integration among the tools and semantic integration (using TSPACE ontologies) among the artifacts that are consumed or produced during different activities that are performed using the tools.

6.8. Conclusions

We have presented and discussed Tools as a service workSPACE (TSPACE) Reference Architecture (RA), TSPACE meta-model that identifies the RA elements and relations among the elements, an ontology-based semantic integration meta-model that provides the backbone of the proposed RA and different views of the RA at multiple levels of abstractions. The presented reference architecture introduces a standardized view of a TSPACE and has the potential to provide a number of benefits to practitioners and researchers. The reference architecture can provide an increased understanding of the TSPACE for software architecting domain in particular and other engineering domains in general. The main aim of the reference architecture is to facilitate the design of concrete TSPACE systems in various domains. The practitioners can use the reference architecture to communicate a TSPACE's requirements and the main architectural principles in software engineering teams. The researchers can use the reference architecture for the identification of potential research areas. Investigation of the application of the existing automated information retrieval mechanisms in the context of the TSPACE to provide fully automated semantic integration among different types of artifacts is one possible direction for future research. There can be a need for extending the reference architecture meta-model for other domains and analyzing the reference architecture components for the extended model. In the proposed reference architecture, we have discussed security implicitly as part of the multi-tenancy. As a future enhancement, the reference architecture also needs to be enhanced by considering security as an explicit non-functional

requirement to provide more comprehensive security support in TSPACE reference architecture.

Listing D

```
@WebMethod(operationName = "initWorkspace")
public boolean initWorkspace(@WebParam(name = "workspaceIdentifier") String workspaceIdentifier) {
    boolean retValue  = false;
    try
    {
        if( workspaceIdentifier != null && workspaceIdentifier.length() > 0 )
        {
            MessageContext messageContext = webServiceContext.getMessageContext();
            HttpServletRequest request = (HttpServletRequest) messageContext.get
                    (MessageContext.SERVLET_REQUEST);
            UtilityMethods utilityMethods = new UtilityMethods();

            if( authenticqtionEnabled ){
                if( !utilityMethods.verifySystemUser(request, em) )
                {
                    return retValue;
                }
            }

            synchronized(this)
            {
                Workspace workspace = (Workspace) workspaceMap.get(workspaceIdentifier);
                if( workspace == null )
                {
                    OntologyManager ontologyManager = new OntologyManager();
                    ontologyManager.populateConceptOntolotyRelationship();
                    ontologyManager.populateAnnotationOntolotyRelationship();
                    ontologyManager.populateChangeOntolotyRelationship();
                    ontologyManager.populateFeatureOntolotyRelationship();
                    ontologyManager.buildAggregatedDesignWorkspaceOntology();
                    workspace = new Workspace(workspaceIdentifier, ontologyManager);
                    workspaceMap.put(workspaceIdentifier, workspace);
                    retValue  = true;
                }
            }
        }
    }
    catch(Exception ex)
    {
        System.out.println("Exception in DesignWorkspaceOntology:initWorkspace");
        System.out.println(ex.toString());
    }
    return retValue;
}
```

Figure 65: Initialize TSPACE - Code

```
@POST
@Consumes({"application/xml", "application/json"})
@Produces({"application/xml", "application/json"})
public SystemUser getAccessKey(@Context HttpServletRequest request, SystemUser systemUser) {
    SystemUser returnSystemUser = new SystemUser();
    SystemUser systemUserTemp = new SystemUser();
    try
    {
        String ip = request.getRemoteAddr();

        Query querySystemUserByUserId = em.createNamedQuery("SystemUser.findByUseridAndPassword");
        querySystemUserByUserId.setParameter("userid", systemUser.getUserid());
        querySystemUserByUserId.setParameter("password", new UtilityMethods().
                generateHashKey(systemUser.getPassword()));
        List<SystemUser> stakeholderList = querySystemUserByUserId.getResultList();
        if( stakeholderList != null && stakeholderList.size()>0 )
        {
            int i=0;
            systemUserTemp = (SystemUser) stakeholderList.get(i);
            systemUserTemp.setCurrentIp(ip);
            UtilityMethods utilityMethod = new UtilityMethods();
            String authKey = utilityMethod.generateAuthKey();
            systemUserTemp.setCurrentAuthKey(authKey);

            DatabaseHandler databaseHandler = new DatabaseHandler(SystemUser.class, em, userTransaction);
            //userTransaction.begin();
            databaseHandler.edit(systemUserTemp);
            //userTransaction.commit();
        }
    }
    catch(Exception ex)
    {
        System.out.println("Exception in method: AuthenticationResoruce-getAccessKey");
        System.out.println(ex.toString());
    }

    returnSystemUser.setUserid(systemUserTemp.getUserid());
    returnSystemUser.setName(systemUserTemp.getName());
    returnSystemUser.setCurrentAuthKey(systemUserTemp.getCurrentAuthKey());
    returnSystemUser.setPicture(systemUserTemp.getPicture());
    returnSystemUser.setPictureFileName(systemUserTemp.getPictureFileName());

    return returnSystemUser;
}
```

Figure 66: Get Authentication Key (OAuth) - Code

```
@WebMethod(operationName = "addConcept")
public boolean addConcept(@WebParam(name = "workspaceIdentifier")
        String workspaceIdentifier,
    @WebParam(name = "ConceptBranchIdentifier")
            String conceptBranchIdentifier,
    @WebParam(name = "ChildConcept") Concept childConcept,
    @WebParam(name = "Relationship") String relationship,
    @WebParam(name = "ParentConcept") Concept parentConcept
    ) {
    boolean returnValue = false;

    try
    {
        MessageContext messageContext = webServiceContext.getMessageContext();
        HttpServletRequest request = (HttpServletRequest) messageContext.get(MessageContext.SERVLET_REQUEST);

        if( authenticqtionEnabled ){
            UtilityMethods utilityMethods = new UtilityMethods();
            if( !utilityMethods.verifySystemUser(request, em) )
            {
                return returnValue;
            }
        }

        Workspace tempWorkspace = (Workspace) workspaceMap.get(workspaceIdentifier);
        if( tempWorkspace != null )
        {
            OntologyManager tempOntologyManager = tempWorkspace.getOntologyManager();
            if( tempOntologyManager != null )
            {
                if( tempOntologyManager.isValidConceptBranchIdentifier(conceptBranchIdentifier) )
                {
                    Concept contentAggregationConcept = tempOntologyManager.
                            getItemFromContentAggregationMap(conceptBranchIdentifier);
                    returnValue = tempOntologyManager.addConcept(childConcept, relationship,
                            parentConcept, contentAggregationConcept);
                }
            }
        }
    }
    catch(Exception ex){}
    return returnValue;
}
```

Figure 67: Add Artifact Data and Relationship - Code

```
public boolean createIncreamentalRDFWithStringLiteralsVersion1(Concept concept, Concept parentConcept, Relation relation)
{
    boolean returnValue = false;
    try
    {
        if( rootOntModel == null )
        {
            rootOntModel = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM_RULE_INF);
        }

        Resource rdfResource = rootOntModel.createResource(Constants.ontologyUri+"#"+concept.getName());
        Property propertyConceptIdentifier = rootOntModel.createProperty(Constants.relationKeywork+":"+Constants.conceptIdentifier);
        rdfResource.addProperty(propertyConceptIdentifier, concept.getName());
        //Adding properties for stakholder and data keywords
        if( concept.getDataKeywords() != null )
        {
            Property propertyDataKeywordIdentifier = rootOntModel.createProperty(Constants.relationKeywork+":"+Constants.dataKeyword);
            rdfResource.addProperty(propertyDataKeywordIdentifier, concept.getDataKeywords());
        }
        if( concept.getStakeholderList() != null )
        {
            Property propertyStakeholderIdentifier = rootOntModel.createProperty(Constants.relationKeywork+":"+Constants.stakeholderIdentifier);
            rdfResource.addProperty(propertyStakeholderIdentifier, concept.getStakeholderIdListInString());
        }

        if(parentConcept != null && relation != null)
        {
            Resource rdfResourceParent = (Resource) rdfResourceIndex.get(parentConcept.getName());
            Resource rdfResourceParentRelation = (Resource) rdfResourceRelationIndex.get(parentConcept.getName());
            if( rdfResourceParentRelation == null )
            {
                rdfResourceParentRelation = rootOntModel.createResource(Constants.ontologyUri+"/"+parentConcept.getName()+"#"+Constants.relationIdentifier);
                rdfResourceRelationIndex.put(parentConcept.getName(), rdfResourceParentRelation);
                Property propertyRelationIdentifier = rootOntModel.createProperty(Constants.relationKeywork+":"+Constants.relationIdentifier);
                rdfResourceParent.addProperty(propertyRelationIdentifier, rdfResourceParentRelation);
            }
            Property relationName = rootOntModel.createProperty(Constants.relationKeywork+":"+relation.getRelationType().getName());
            rdfResourceParentRelation.addProperty(relationName, rdfResource);
        }
        rdfResourceIndex.put(concept.getName(), rdfResource);
        returnValue = true;
    }
```

Figure 68: Create RDF Incrementally – Code

```
<?xml version="1.0"?>
<rdf:RDF xmlns="http://www.itu.dk/LoadBalancerOut.owl#"
     xml:base="http://www.itu.dk/LoadBalancerOut.owl"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:out="http://www.itu.dk/out.owl#">
    <owl:Ontology rdf:about="http://www.itu.dk/LoadBalancerOut.owl"/>

    <rdfs:Datatype rdf:about="http://www.itu.dk/LoadBalancerOut.owl#anyType"/>


    <owl:DatatypeProperty rdf:about="hasEnable">
        <rdfs:domain rdf:resource="http://www.itu.dk/LoadBalancerOut.owl#LoadBalancer"/>
        <rdfs:range rdf:resource="http://www.itu.dk/LoadBalancerOut.owl#anyType"/>
    </owl:DatatypeProperty>

    <owl:DatatypeProperty rdf:about="hasLoadBalancerRules">
        <rdfs:domain rdf:resource="http://www.itu.dk/LoadBalancerOut.owl#LoadBalancer"/>
        <rdfs:range rdf:resource="http://www.itu.dk/LoadBalancerOut.owl#anyType"/>
    </owl:DatatypeProperty>

    <owl:DatatypeProperty rdf:about="hasScalabilityThreshold">
        <rdfs:domain rdf:resource="http://www.itu.dk/LoadBalancerOut.owl#LoadBalancer"/>
        <rdfs:range rdf:resource="http://www.itu.dk/LoadBalancerOut.owl#anyType"/>
    </owl:DatatypeProperty>

    <owl:Class rdf:about="http://www.itu.dk/LoadBalancerOut.owl#LoadBalancer">
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="hasScalabilityThreshold"/>
                <owl:qualifiedCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#
                        nonNegativeInteger">1</owl:qualifiedCardinality>
                <owl:onDataRange rdf:resource="http://www.itu.dk/LoadBalancerOut.owl#anyType"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="hasLoadBalancerRules"/>
                <owl:qualifiedCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#
                        nonNegativeInteger">1</owl:qualifiedCardinality>
                <owl:onDataRange rdf:resource="http://www.itu.dk/LoadBalancerOut.owl#anyType"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="hasEnable"/>
                <owl:qualifiedCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#
                        nonNegativeInteger">1</owl:qualifiedCardinality>
                <owl:onDataRange rdf:resource="http://www.itu.dk/LoadBalancerOut.owl#anyType"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>


    <owl:Class rdf:about="http://www.itu.dk/out.owl#boolean"/>
</rdf:RDF>
```

Figure 69: Load Balancer UML Model Ontology

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:j.0="http://itu.dk/TspaceOntology/1.0/"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <rdf:Description rdf:about="http://itu.dk/TspaceOntology/1.0/ContentAggregation">
    <j.0:Relationship rdf:parseType="Resource">
      <j.0:hasPart>
        <rdf:Description rdf:about="http://itu.dk/TspaceOntology/1.0/AggregationItem">
          <j.0:Relationship rdf:parseType="Resource">
            <j.0:consistOf>
              <rdf:Description rdf:about="http://itu.dk/TspaceOntology/1.0/ContentUnit">
                <j.0:Relationship rdf:parseType="Resource">
                  <j.0:contains>
                    <rdf:Description rdf:about="http://itu.dk/TspaceOntology/1.0/Model">
                      <j.0:Concept>ModelContentUnit</j.0:Concept>
                    </rdf:Description>
                    <j.0:contains>
                        <rdf:Description rdf:about="http://itu.dk/TspaceOntology/1.0/ModelElement">
                                <j.0:Concept>
                                        <j.0:ConceptID>ArgoUML_543_LoadBalancer</j.0:ConceptID>
                                        .
                                        .
                                </j.0:Concept>
                        </rdf:Description>
                    </j.0:contains>
                  </j.0:contains>
                  <j.0:contains>
                    <rdf:Description rdf:about="http://itu.dk/TspaceOntology/1.0/Knowledge">
                      <j.0:Concept>KnowledgeContentUnit</j.0:Concept>
                    </rdf:Description>
                    <j.0:contains>
                        <rdf:Description rdf:about="http://itu.dk/TspaceOntology/1.0/KnowledgeElement">
                                <j.0:Concept>
                                        <j.0:ConceptID>PAKME_243_Load Balancer Architecture</j.0:ConceptID>
                                        .
                                        .
                                </j.0:Concept>
                        </rdf:Description>
                    </j.0:contains>
                  </j.0:contains>
                  <j.0:contains>
                    <rdf:Description rdf:about="http://itu.dk/TspaceOntology/1.0/RequirementsContentUnit">
                      <j.0:Concept>RequirementContentUnit</j.0:Concept>
                    </rdf:Description>
                    .
                    .
                  </j.0:contains>
                  <j.0:contains>
                    <rdf:Description rdf:about="http://itu.dk/TspaceOntology/1.0/DecisionContentUnit">
                      <j.0:Concept>DecisionContentUnit</j.0:Concept>
                    </rdf:Description>
                    .
                    .
                  </j.0:contains>
                </j.0:Relationship>
                <j.0:Concept>ContentUnit</j.0:Concept>
              </rdf:Description>
            </j.0:consistOf>
          </j.0:Relationship>
          <j.0:Concept>AggregationItem</j.0:Concept>
        </rdf:Description>
      </j.0:hasPart>
    </j.0:Relationship>
    <j.0:Concept>ContentAggregation</j.0:Concept>
  </rdf:Description>
</rdf:RDF>
```

Figure 70: XML Representation of Abstract TSPACE Ontology RDF

Chapter 7. Lessons Learned, Conclusions and Directions for Future Work

The main objective of the research presented in this dissertation was to design a reference architecture for providing Tools as a service workSPACE (TSPACE), that can be used to provision software architecting Tools as a Service (TaaS). To achieve the research objective, we have performed a number of research activities. We have investigated the research challenges associated with architecture design and development of cloud-enabled software systems. We have analyzed different approaches that have been proposed for designing and developing software reference architecture and have adopted a tailored reference architecture design methodology that was suitable for TSPACE. We have analyzed functional and quality requirements of TSPACE reference architecture and have provided detailed of the reference architecture at multiple levels of abstractions. In this chapter, we discuss our experiences with designing TSPACE reference architecture, our conclusions and directions for future work.

7.1. Lessons Learned

The research activity of designing TSPACE reference architecture has enabled us to make many important observations, not only in terms of design of the reference architecture but also the process that has lead to the design. In this section, we elaborate our experiences and lessons learned related to design of TSPACE reference architecture.

7.1.1. Adopting Appropriate Reference Architecture Design Approach

The selection of appropriate reference architecture design approaches play a vital role in the identification of software reference architecture (RA) goals, RA design strategy, RA representations in terms of reference architecture views and RA application. Existing literature provides a number of meta-level reference architecture design approaches [15-17]. Although the reported approaches identify general elements of the reference architecture design process, the detailed activities that are needed to explore different dimensions of reference architecture design are not described. Hence, the design strategy to be adopted for designing a software reference architecture should guide identification of reference architecture goals (requirements), identification of different elements of the reference architecture, the process that guides the relationships between reference architecture elements and the reference architecture enactment strategies.

To fill this gap, we have elaborated TSPACE reference architecture design process in Chapter 3. Other than focusing on reference architecture description, goal, design and instantiation of the reference architecture, TSPACE reference architecture design process also focuses on leveraging domain models (e.g. IEEE 1471-2000 [47] and ISO/IEC/IEEE 42010:2011 [7]) for identification of reference architecture elements and relationships between the elements, analysis of run time characteristics of the reference architecture (e.g. provisioning, enactment and *aaS model) and analysis of the hosting environments (e.g. IaaS Cloud platforms).

7.1.2. Functional Demarcation between the Reference Architecture and the Tools to be provisioned

TSPACE reference architecture is specific in the way that TSPACE features are used to complement the functionality that is provided by the provisioned tools. TSPACE facilitates tools bundling, provisioning and tools operations, whereas the provisioned tools provide features that are required to perform different activities. For example, when architecture significant requirements documentation, architecture knowledge management, architecture analysis and architecture design tools (as described in Chapter 6) are provisioned by TSPACE, integration among the tools and awareness of the operations on the artifacts is supported by TSPACE, whereas activities associated with software architecting are supported by the features that are implemented in the provisioned tools. Hence, it is very important to have a clear demarcation between the functionality of TSPACE and the functionality that is to be supported by the provisioned tools. The research that is presented in this dissertation has been focusing on software architecting tools. For adoption of TSPACE in other engineering domains, some of its functionality may need to be enhanced to meet the requirements of the respective domain. Functional demarcation analysis of the respective engineering domain will be required to analyze and identify the features that TSPACE would require to complement the tools of that domain.

7.1.3. Impact of Standardized Domain Models on the Reference Architecture Design Process

Availability of standardization models for respective domains impacts the reference architecture design process. While designing TSPACE reference architecture for software architecting domain, we have leveraged IEEE 1471-2000 [47] and ISO/IEC/IEEE 42010:2011 [7] architecture documentation models that have been used as a baseline for identification of TSPACE architecture elements and TSPACE meta-model (Section 6.3) design. The meta-model has been further enhanced by analyzing TSPACE requirements. Incorporation of a standardized domain model in the reference architecture

design ensures the applicability of reference architecture for a broader range of tools (as has been demonstrated in Section 6.5). Unavailability of the standardization models for the respective domain or their disuse during the reference architecture design can negatively impact the applicability of the reference architecture.

7.1.4. Selecting Appropriate Approach to Establish Relationship between Artifacts Produced by the Tools

Establishing a relationship among the artifacts of different abstraction levels that are maintained by the provisioned tools is a critical characteristic of TSPACE reference architecture and can play a significant role in TSPACE reference architecture adoption. Hence, it is important to identify integration needs for the artifacts of different tools to be provisioned in a cloud-enabled workspace. As different tools have different formats of the artifacts (e.g. text documents, standardization formats such as UML, proprietary formats and database tables used for persistence), there is a need to have appropriate semantic integration models that can provide semantic integration among different formats. Our experiences with designing TSPACE architecture and its implementation for tools used for software architecting have shown that an ontology-driven semantic integration model (Chapter 5) can provide support for relating different artifacts with each other even though the artifacts are maintained by different tools using their proprietary data structures.

7.1.5. Analyzing Integration needs of TSPACE Reference Architecture

As TSPACE can provision different type of tools, it is important to analyze integration needs for different types of tools used in a specific domain. As our research effort has been focusing on providing TSPACE for software architecting domain, we have focused on three different types of integration. (a) Semantic integration that facilitates the relation of artifacts of different abstractions and different formats. (b) Process-centric integration to incorporate use cases in which the tools provisioned as part of the tools chain need to exchange information according to project-specific development processes (e.g. artifacts flow from a design tool in model driven development to code generation tool, or to manage collaboration in distributed architecture evaluation processes [177]). In such cases, the semantic integration support needs to be complemented by a workflow-based process, so that artifacts among the tools can be exchanged according to defined software development processes. (c) Support for simple exchange of artifacts among the tools so that the artifacts can be exported from a tool and can be imported into another tool.

Integration needs for other engineering domains can vary, and TSPACE adoption in that domain may need to investigate specific integration

requirements of the respective domain. E.g. mechatronics domain [178], in which software design is tightly integrated with design of mechanical components, integration models and approaches to integrate software system design with mechanical system design can be required.

7.1.6. Selection of Appropriate IaaS Clouds and Cloud Deployment Models

As tools in TSPACE reference architecture are considered as black box, the tenant-specific constrains on artifacts' storage location are applied on the tools and the tools are provisioned on the location that is compliant with the constraints (in our prototype implementation, we have used Amazon EC2 location-specific provisioning features). However, in more complex use cases, where location constraints on the artifacts can change during their lifecycle, Virtual Machines (VMs) hosting the tools might need to be migrated from one location to another. In such cases, the capability of underlying IaaS to support VMs migration would play a critical rule. Hence, IaaS cloud selection and selection of cloud deployment model (e.g. public, private or hybrid) should be carefully made. A cloud environment that supports the desired features should be selected.

7.1.7. TSPACE adoption for Quality Critical Domains

In our proposed TSPACE reference architecture, we have considered each of the provisioned tools as a black box and have not considered the management of quality characteristics of each individual provisioned tool during its lifecycle. However, for certain tools that are based on executable artifacts, e.g. testing tools that are used for performance testing, can require extra computing, memory or other resources during their life cycle depending on the tasks to be executed. In such cases, TSPACE needs to incorporate the metrics and corresponding prediction models so that additional resources can be acquired according to the resource requirements of the tasks that are being performed using a specific tool or a set of tools.

7.1.8. A Hybrid Approach for TSPACE Reference Architecture Evaluation

Considering the generic nature of TSPACE reference architecture and a broad range of potential stakeholders, multiple architecture evaluation techniques needs to be adopted for evaluating the reference architecture from different perspectives. As elaborated in Chapter 6, we have evaluated TSPACE reference architecture using a scenario-based evaluation method (technique) [12], architecture tradeoff analysis method [13] and a prototype implementation of the reference architecture. Scenario-based evaluation

approach facilitates the evaluation of the completeness of the reference architecture with respect to reference architecture objectives and requirements. Architecture tradeoff analysis method facilitates identification of strong and weak points of the reference architecture. A prototype is a viable way to demonstrate feasibility of the reference architecture.

7.2. Conclusions

The research that has been presented in this dissertation aims at providing a cloud-enabled workspace that can facilitate on demand provisioning of tools as a service. In this dissertation, four main research contributions can be distinguished.

Firstly, this dissertation presents an overview of challenges associated with software architecting of cloud-enabled systems and the solutions that can be adopted to address the challenges (Chapter 2). The findings are based upon the extensive review of the 111 journal papers. We have also considered multiple commercial solutions to make a comparison between how the cloud software architecting is addressed by researchers and practitioners. The findings of the study have been classified into four high-level categories. (a) Architecture challenges and solutions that have been discussed under *Resource and Service Management* category address problems associated with achieving different functional and quality requirements in cloud-enabled software systems. (b) *Workflow Management* category describes synthesis of architecture challenges and solutions for designing workflow-based systems on cloud. (c) *Service Level Agreement (SLA) Compliance* category discusses solutions to satisfy SLAs on cloud. (d) *Energy Awareness* category describes architecture solutions to have energy efficient systems on the cloud. The findings of the review highlight the importance of identifying important cloud-specific quality and functional characteristic in cloud-enabled software systems. The review also reveals that particular IaaS or PaaS cloud may not be sufficient to meet all the requirements of a specific solution and often a hybrid of multiple cloud platforms is required.

Secondly, this dissertation presents TSPACE reference architecture design process (Chapter 3), which is based on existing reference architecture design guidelines [15-17] and our experience with developing process guidelines for architecting cloud-based systems [27, 37, 46]. The tailoring of software reference architecture design process suggests that there is a need to have a clear demarcation between quality characteristics and functionality to be supported by TSPACE and functionality to be supported by the provisioned tools. The process guidelines also suggest that domain models of the respective domain (e.g. software architecting domain that has been focused in this dissertation) play a key role in identifying TSPACE elements for the

domain. As in the design of every software system, general architecture styles and patterns as well as cloud-specific architecture styles and patterns facilitate TSPACE reference architecture design. Moreover, analysis of the potential cloud environments that are to be used for TSPACE deployment and hosting of the tools that are to be provisioned by TSPACE with respect to the tools' domain (e.g. software architecting tools) is also important.

Thirdly, based on the analysis of TSPACE business paradigm and requirements, we have proposed an ontology-driven semantic integration model (Chapter 5). The proposed ontology model facilitates tools selection and provisioning, provides support for semantic integration among the artifacts of different levels of abstraction and tracking operations that are performed on the artifacts by the users using different tools (that are provisioned by TSPACE). The proposed ontology model consists of artifacts, annotations and tools ontology to establish the semantic relationship between different types of artifacts and different parts of the artifacts using annotations. The proposed ontologies provide a basic foundation for semantic integration. The ontologies are evolved during the lifecycle of TSPACE as artifacts are produced and consumed by the tools.

Fourthly, we have provided a detailed TSPACE reference architecture in terms of multiple TSPACE elements (Chapter 6). The core of TSPACE reference architecture is TSPACE meta-model that captures abstractions of different elements of TSPACE reference architecture and describes the relationship between the elements. These elements are further elaborated using ontology meta-models (based on the ontology model described in Chapter 5) and four different levels of architecture abstraction using logical, process and deployment views of TSPACE reference architecture. TSPACE architecture is evaluated using scenario based evaluation, Architecture Tradeoff Analysis Method (ATAM) [13], a prototype of the reference architecting utilizing Amazon IaaS cloud. Evaluation results demonstrate that TSPACE functional and quality requirements are positively addressed in TSPACE reference architecture.

7.3. Threats to Validity of TSPACE Reference Architecture

We have adopted a number of strategies to address threats to TSPACE Reference Architecture validity. A comprehensive empirical study conducted on design and adoption of reference architecture [44] has reported a number of challenges and threats to reference architecture adoption. The main problem that has been reported in the study [44] is that the reference architectures are often too abstract to be transformed into concrete implementations. To counter this threat, we have presented TSPACE reference architecture at four different levels of abstraction and fourth level

decomposition of the reference architecture document concrete methods that can be used in implementation of the reference architecture. Moreover, a proof of concept implementation of TSPACE reference architecture prototype also provides insight for the reference architecture adoption. Unavailability of the important view in reference architecture documentation is also a commonly occurring problem [44]. We have addressed this issue by describing reference architecture using multiple logical views, corresponding process views and a deployment view of TSPACE on Amazon cloud. The reference architecture views have been complemented using textual description. We have also provided details on TSPACE ontologies that facilitate tools selection, provisioning and semantic integration, and have described details of the important methods in terms of algorithms and mathematical formulas. Unavailability of the reference architecture design and evaluation method with a reference architecture has been reported as a major concern [44]. To address this issue, we have provided extensive details on TSPACE reference architecture design process and evaluation approach. If the reference architecture is to be adopted in another domain, TSPACE reference architecture design process can be followed to fill the gaps in TSPACE reference architecture for that specific domain (e.g. by adding new components or more details of the existing components) and reported TSPACE reference architecture evaluation strategy can be used for the evaluation of the enhanced architecture.

Selection of architecture solutions that have been used to achieve the functional and quality characteristics of TSPACE reference architecture can be a threat to internal validity [80]. This threat has been mitigated by using well known architecture styles and patterns [50]. The process for designing TSPACE reference architecture and the reference architecture documentation approach can be a threat to construct validity [80]. It has been addressed by following a reference architecture design and documentation approach that is based on established principles [15-17, 44].

7.4. Directions for Future Work

A number of research problems for TSPACE and its reference architecture are open for further research, which we tend to explore during future work. To facilitate adoption of TSPACE reference architecture in different engineering domains, a catalogue of architecture patterns and styles for cloud-based system can significantly improve the process of tailoring and adoption of TSPACE reference architecture. The semantic integration approach that has been presented in this dissertation provides a semi-automated mechanism for integrating artifacts of different types and different levels of abstraction, where ontologies provide a foundation for semantic integration among the artifacts. The annotations based on user inputs are used as semantic

integration rules. The process of annotating the artifacts and parts of the artifacts can be automated by applying advance information retrieval and machine learning techniques [109].

The presented tailored reference architecture design process satisfies TSPACE reference architecture design requirements. However, the process needs to be further explored in a broader context of Tools as a Service (TaaS) in different domains. In this dissertation, we have focused on the tools (software architecting related tools) that are not used to perform safety critical or mission critical tasks. For TSPACE reference architecture adoption in safety critical and mission critical domains, the reference architecture needs to incorporate architecture abstraction for different types of system quality metrics as well as metrics' data collection methods, e.g. runtime quality measurement metrics such as performance and reliability [179]. TSPACE reference architecture also needs to incorporate methods for Service Level Agreement (SLA) compliance for safety-critical and mission-critical TSPACE.

In the reported TSPACE reference architecture, we have focused on bundling tools into a suite of tools and the tools provisioning, whereas individual tools have been treated as black box. There are two possible research directions for enhancements to this approach. (a) Only selected features of the tools are enabled when the tools are bundled in a suite to have more controlled service and pricing model. For this purpose, feature modeling techniques [180] (that are used for product line engineering) in combination with Aspect Oriented Paradigm (AOP) [181] can be adopted in the reference architecture. (b) For next generation tools, which are implemented using Service Oriented Architecture (SOA) principles, a tool can be dynamically constructed by following principle of service composition [182] based on the required features and activities. TSPACE reference architecture ontologies and provisioning models, which already provide extensive support for tools selection and provisioning, can be enhanced by incorporating service composition methods.

References

[1] M. A. Chauhan, M. Ali Babar, and Q. Z. Sheng, "A Reference Architecture for a Cloud-Based Tools as a Service Workspace," presented at the 2015 IEEE Conference on Service Computing (SCC), New York, USA, 2015.

[2] M. A. Chauhan, "A reference architecture for providing tools as a service to support global software development," in *Proceedings of the WICSA 2014 Companion Volume*, 2014, p. 16.

[3] P. C. Clements and L. M. Northrop, "Software Architecture: An Executive Overview," DTIC Document1996.

[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-oriented software architecture: a system of patterns*: John Wiley & Sons, Inc., 1996.

[5] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*: Addison-Wesley Professional, 2012.

[6] I. Gorton, "Essential Software Architecture," ed: Springer Berlin Heidelberg, 2011.

[7] "ISO/IEC/IEEE Systems and software engineering -- Architecture description," *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000),* pp. 1-46, 2011.

[8] C. Hofmeister, P. Kruchten, R. L. Nord, H. Obbink, A. Ran, and P. America, "A general model of software architecture design derived from five industrial approaches," *Journal of Systems and Software,* vol. 80, pp. 106-126, 1 2007.

[9] R. Wojcik, F. Bachmann, L. Bass, P. Clements, P. Merson, R. Nord*, et al.*, "Attribute-Driven Design (ADD), Version 2.0," DTIC Document2006.

[10] P. Kruchten, *The rational unified process: an introduction*: Addison-Wesley Professional, 2004.

[11] P. America, E. Rommes, and H. Obbink, "Multi-view variation modeling for scenario analysis," in *Software Product-Family Engineering*, ed: Springer, 2004, pp. 44-65.

[12] R. Kazman, L. Bass, G. Abowd, and M. Webb, "SAAM: a method for analyzing the properties of software architectures," in *Software Engineering, 1994. Proceedings. ICSE-16., 16th International Conference on*, 1994, pp. 81-90.

[13] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, "The architecture tradeoff analysis method," in *Engineering of Complex Computer Systems, 1998. ICECCS '98. Proceedings. Fourth IEEE International Conference on*, 1998, pp. 68-78.

[14] M. Matinlassi, E. Niemelä, and L. Dobrica, "Quality-driven architecture design and quality analysis method," *VTT publication,* vol. 456, p. 128, 2002.

[15] S. Angelov, P. Grefen, and D. Greefhorst, "A framework for analysis and design of software reference architectures," *Information and Software Technology,* vol. 54, pp. 417-431, 2012.
[16] P. Avgeriou, "Describing, instantiating and evaluating a reference architecture: A case study," *Enterprise Architect Journal,* p. 24, 2003.
[17] S. Angelov, J. J. Trienekens, and P. Grefen, "Towards a method for the evaluation of reference architectures: Experiences from a case," in *Software Architecture*, ed: Springer, 2008, pp. 225-240.
[18] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski*, et al.*, "A view of cloud computing," *Commun. ACM,* vol. 53, pp. 50-58, 2010.
[19] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, "What's inside the Cloud? An architectural map of the Cloud landscape," in *Software Engineering Challenges of Cloud Computing, 2009. CLOUD '09. ICSE Workshop on*, 2009, pp. 23-31.
[20] P. Louridas, "Up in the Air: Moving Your Applications to the Cloud," *Software, IEEE,* vol. 27, pp. 6-11, 2010.
[21] "Amazon, http://aws.amazon.com/ [July, 2015]."
[22] "Salesforce, https://developer.salesforce.com/page/Multi_Tenant_Architecture [July, 2015]."
[23] R. Buyya, S. Pandey, and C. Vecchiola, "Cloudbus toolkit for market-oriented cloud computing," in *Cloud Computing*, ed: Springer, 2009, pp. 24-44.
[24] P. Lago and T. Jansen, "Creating Environmental Awareness in Service Oriented Software Engineering Service-Oriented Computing." vol. 6568, E. Maximilien, G. Rossi, S.-T. Yuan, H. Ludwig, and M. Fantinato, Eds., ed: Springer Berlin Heidelberg, 2011, pp. 181-186.
[25] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.,* vol. 39, pp. 50-55, 2008.
[26] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," *NIST,* vol. Special Publication 800-145, 2011.
[27] M. A. Babar and M. A. Chauhan, "A tale of migration to cloud computing for sharing experiences and observations," presented at the Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing, Waikiki, Honolulu, HI, USA, 2011.
[28] F. Baiardi and D. Sgandurra, "Securing a Community Cloud," in *Distributed Computing Systems Workshops (ICDCSW), 2010 IEEE 30th International Conference on*, 2010, pp. 32-41.
[29] A. Khajeh-Hosseini, D. Greenwood, and I. Sommerville, "Cloud Migration: A Case Study of Migrating an Enterprise IT System to

IaaS," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, 2010, pp. 450-457.
[30] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications,* vol. 1, pp. 7-18, 2010.
[31] "AmazonEC2, http://aws.amazon.com/ec2/ [July, 2015]."
[32] "Amazon Simple Storage Services, http://aws.amazon.com/s3/ [July, 2015]."
[33] "Eucalyptus, http://www.eucalyptus.com/ [July, 2015]."
[34] "OpenNebula, http://opennebula.org/ [July, 2015]."
[35] "Google App Engine, http://code.google.com/appengine/ [July, 2015]."
[36] "Microsoft Azure, http://azure.microsoft.com [July, 2015]."
[37] M. A. Chauhan and M. A. Babar, "Migrating Service-Oriented System to Cloud Computing: An Experience Report," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, 2011, pp. 404-411.
[38] M. A. Chauhan and M. A. Babar, "Cloud infrastructure for providing tools as a service: quality attributes and potential solutions," presented at the Proceedings of the WICSA/ECSA 2012 Companion Volume, Helsinki, Finland, 2012.
[39] B. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering," in *Version* vol. 2, ed, 2007, pp. 2007-01.
[40] B. P. Gallagher, "Using the architecture tradeoff analysis methodsm to evaluate a reference architecture: a case study," DTIC Document2000.
[41] M. A. Chauhan, M. Ali Babar, and B. Benatallah, "Architecting Cloud-Enabled Systems: A Systematic Survey of Challenges and Solutions," *Under review in Software: Practice and Experience Journal*.
[42] M. A. Chauhan and M. A. Babar, "A Systematic Mapping Study of Software Architectures for Cloud Based Systems," *Technical Report TR-2014-175, IT University of Copenhagen,* 2014.
[43] S. Angelov and P. Grefen, "An e-contracting reference architecture," *Journal of Systems and Software,* vol. 81, pp. 1816-1844, 2008.
[44] S. Angelov, J. Trienekens, and R. Kusters, "Software reference architectures-exploring their usage and design in practice," in *Software Architecture*, ed: Springer, 2013, pp. 17-24.
[45] N. Harrison and P. Avgeriou, "Using Pattern-Based Architecture Reviews to Detect Quality Attribute Issues - An Exploratory Study," in *Transactions on Pattern Languages of Programming III*. vol. 7840, J. Noble, R. Johnson, U. Zdun, and E. Wallingford, Eds., ed: Springer Berlin Heidelberg, 2013, pp. 168-194.

[46] M. A. Chauhan and M. A. Babar, "Towards Process Support for Migrating Applications to Cloud Computing," in *2012 International Conference on Cloud and Service Computing*, 2012, pp. 80-87.
[47] "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems," *IEEE Std 1471-2000,* pp. i-23, 2000.
[48] F. Arvidsson and A. Flycht-Eriksson, "“Ontology I”. Retrieved 23 June 2014.."
[49] M. A. Chauhan, M. Ali Babar, Q.Z. Sheng, "A Reference Architecture for Tools as a Service Workspace: Meta-model, Ontologies and Design Elements," *Under review in Future Generation Computer Systems Journal*.
[50] P. Avgeriou and U. Zdun, "Architectural patterns revisited–a pattern," 2005.
[51] P. B. Kruchten, "The 4+1 View Model of architecture," *Software, IEEE,* vol. 12, pp. 42-50, 1995.
[52] M. A. Chauhan and M. A. Babar, "Towards a Reference Architecture to Provision Tools as a Service for Global Software Development," in *Software Architecture (WICSA), 2014 IEEE/IFIP Conference on*, 2014, pp. 167-170.
[53] M. A. Chauhan and M. A. Babar, "Towards a Reference Architecture to Provision Tools as a Service for Global Software Development," *WICSA 2014,* 2014.
[54] J. D. Herbsleb, "Global Software Engineering: The Future of Socio-technical Coordination," in *Future of Software Engineering, 2007. FOSE '07*, 2007, pp. 188-198.
[55] W. Maalej, "Task-First or Context-First? Tool Integration Revisited," in *Automated Software Engineering, 2009. ASE '09. 24th IEEE/ACM International Conference on*, 2009, pp. 344-355.
[56] H. Zhang and M. Ali Babar, "Systematic reviews in software engineering: An empirical investigation," *Information and Software Technology,* vol. 55, pp. 1341-1354, 2013.
[57] H. Zhang and M. A. Babar, "On searching relevant studies in software engineering," in *Proceedings of the 14th international conference on evaluation and assessment in software engineering (EASE)*, 2010.
[58] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," *Journal of systems and software,* vol. 80, pp. 571-583, 2007.
[59] H. Zhang, M. A. Babar, and P. Tell, "Identifying relevant studies in software engineering," *Information and Software Technology,* vol. 53, pp. 625-637, 2011.
[60] G. W. Noblit and R. D. Hare, "Meta-ethnography: synthesizing qualitative studies. Newbury Park: Sage, 1988."

[61] D. S. Cruzes and T. Dybå, "Recommended Steps for Thematic Synthesis in Software Engineering," in *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on*, 2011, pp. 275-284.
[62] J. Samuel T. Redwine and W. E. Riddle, "Software technology maturation," presented at the Proceedings of the 8th international conference on Software engineering, London, England, 1985.
[63] T. Dybå and T. Dingsøyr, "Strength of evidence in systematic reviews in software engineering," presented at the Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, Kaiserslautern, Germany, 2008.
[64] T. Dybå and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," *Information and Software Technology,* vol. 50, pp. 833-859, 8 2008.
[65] A. Celesti, F. Tusa, M. Villari, and A. Puliafito, "How to Enhance Cloud Architectures to Enable Cross-Federation," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, 2010, pp. 337-345.
[66] F. Perez-Sorrosal, M. Patino-Martinez, R. Jimenez-Peris, and B. Kemme, "Elastic SI-Cache: consistent and scalable caching in multi-tier architectures," *The VLDB Journal,* vol. 20, pp. 841-865, 2011.
[67] "Amazon Auto Scaling, http://aws.amazon.com/autoscaling/ [July, 2015]."
[68] "Amazon Elastic Load Balancing, http://aws.amazon.com/elasticloadbalancing/ [July, 2015]."
[69] "Amazon Cloud Watch, http://aws.amazon.com/cloudwatch/ [July, 2015]."
[70] "Amazon Simple Workflow Service, http://aws.amazon.com/swf/ [July, 2015]."
[71] "Google App Engine Architecture, http://web.stanford.edu/class/ee380/Abstracts/081105-slides.pdf [July, 2015]."
[72] K. Roche and J. Douglas, "App Engine Services," in *Beginning Java™ Google App Engine*, ed: Apress, 2010, pp. 169-195.
[73] A. Azeez, S. Perera, D. Gamage, R. Linton, P. Siriwardana, D. Leelaratne*, et al.*, "Multi-tenant SOA Middleware for Cloud Computing," in *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*, ed: IEEE, 2010, pp. 458-465.
[74] E. M. Maximilien, A. Ranabahu, R. Engehausen, and L. Anderson, "IBM altocumulus: a cross-cloud middleware and platform," presented at the Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications, Orlando, Florida, USA, 2009.

[75] L. Shang, S. Petiton, N. Emad, X. Yang, and Z. Wang, "Extending YML to Be a Middleware for Scientific Cloud Computing," in *Cloud Computing*. vol. 5931, M. Jaatun, G. Zhao, and C. Rong, Eds., ed: Springer Berlin Heidelberg, 2009, pp. 662-667.

[76] S. van der Burg, M. de Jonge, E. Dolstra, and E. Visser, "Software deployment in a dynamic cloud: From device to service orientation in a hospital environment," in *ICSE Workshop on Software Engineering Challenges of Cloud Computing, 2009. CLOUD '09*, ed: IEEE, 2009, pp. 61-66.

[77] I. Gorton, L. Yan, and Y. Jian, "Exploring Architecture Options for a Federated, Cloud-Based System Biology Knowledgebase," in *2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, ed: IEEE, 2010, pp. 218-225.

[78] J. Alvares de Oliveira and T. Ledoux, "Self-optimisation of the energy footprint in service-oriented architectures," presented at the Proceedings of the 1st Workshop on Green Computing, Bangalore, India, 2010.

[79] T. C. Chieu, A. Mohindra, A. A. Karve, and A. Segal, "Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment," in *IEEE International Conference on e-Business Engineering, 2009. ICEBE '09*, ed: IEEE, 2009, pp. 281-286.

[80] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslen, *Experimentation in software engineering: an introduction*: Kluwer Academic Publishers, 2000.

[81] A. Nowak, T. Binz, C. Fehling, O. Kopp, F. Leymann, and S. Wagner, "Pattern-driven green adaptation of process-based applications and their runtime infrastructure," *Computing,* vol. 94, pp. 463-487, 2012.

[82] Y.-B. Han, J.-Y. Sun, G.-L. Wang, and H.-F. Li, "A cloud-based bpm architecture with user-end distribution of non-compute-intensive activities and sensitive data," *Journal of Computer Science and Technology,* vol. 25, pp. 1157-1167, 2010.

[83] H. Flores and S. N. Srirama, "Mobile cloud middleware," *Journal of Systems and Software,* vol. 92, pp. 82-94, 2014.

[84] X. Hu, T. H. Chu, H. C. Chan, and V. C. Leung, "Vita: A crowdsensing-oriented mobile cyber-physical system," *Emerging Topics in Computing, IEEE Transactions on,* vol. 1, pp. 148-165, 2013.

[85] L. Gkatzikis and I. Koutsopoulos, "Migrate or not? exploiting dynamic task migration in mobile cloud computing systems," *Wireless Communications, IEEE,* vol. 20, pp. 24-32, 2013.

[86] R. Martignoni, "Global Sourcing of Software Development - A Review of Tools and Services," in *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on*, 2009, pp. 303-308.

[87] J. Whitehead, "Collaboration in Software Engineering: A Roadmap," presented at the 2007 Future of Software Engineering, 2007.
[88] Muhammad Ali Babar, Torgeir Dingsøyr, Patricia Lago, and H. v. Vliet, "Software Architecture Knowledge Management: Theory and Practice," *Springer,* 2009.
[89] M. Völter, T. Stahl, J. Bettin, A. Haase, and S. Helsen, *Model-driven software development: technology, engineering, management*: John Wiley & Sons, 2013.
[90] P. Dourish and V. Bellotti, "Awareness and coordination in shared workspaces," presented at the Proceedings of the 1992 ACM conference on Computer-supported cooperative work, Toronto, Ontario, Canada, 1992.
[91] B. Sodhi and T. V. Prabhakar, "Application architecture considerations for cloud platforms," in *2011 Third International Conference on Communication Systems and Networks (COMSNETS)*, ed: IEEE, 2011, pp. 1-4.
[92] E. J. Domingo, J. T. Nino, A. L. Lemos, M. L. Lemos, R. C. Palacios, and J. M. G. BerbiÃÅs, "CLOUDIO: A Cloud Computing-Oriented Multi-tenant Architecture for Business Information Systems," in *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*, ed: IEEE, 2010, pp. 532-533.
[93] R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, and R. Buyya, "The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid Clouds," *Future Generation Computer Systems,* vol. 28, pp. 861-870, 2012.
[94] R. Han, M. M. Ghanem, L. Guo, Y. Guo, and M. Osmond, "Enabling cost-aware and adaptive elasticity of multi-tier cloud applications," *Future Generation Computer Systems,* vol. 32, pp. 82-98, 2014.
[95] I. Brandic, D. Music, and S. Dustdar, "Service mediation and negotiation bootstrapping as first achievements towards self-adaptable grid and cloud services," presented at the Proceedings of the 6th international conference industry session on Grids meets autonomic computing, Barcelona, Spain, 2009.
[96] E. Y. Nakagawa, E. F. Barbosa, and J. C. Maldonado, "Exploring ontologies to support the establishment of reference architectures: An example on software testing," in *Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on*, 2009, pp. 249-252.
[97] S. Martínez-Fernández, C. Ayala, X. Franch, and H. M. Marques, "Artifacts of software reference architectures: a case study," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, 2014, p. 42.
[98] S. I. Hashmi, V. Clerc, M. Razavian, C. Manteli, D. A. Tamburri, P. Lago*, et al.*, "Using the Cloud to Facilitate Global Software

Development Challenges," in *Global Software Engineering Workshop (ICGSEW), 2011 Sixth IEEE International Conference on*, 2011, pp. 70-77.

[99] J. A. Osorio, M. R. Chaudron, and W. Heijstek, "Moving from waterfall to iterative development: An empirical evaluation of advantages, disadvantages and risks of RUP," in *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, 2011, pp. 453-460.

[100] I. Steinmacher, A. P. Chaves, and M. A. Gerosa, "Awareness support in global software development: a systematic review based on the 3C collaboration model," presented at the Proceedings of the 16th international conference on Collaboration and technology, Maastricht, The Netherlands, 2010.

[101] C. Gutwin, R. Penner, and K. Schneider, "Group awareness in distributed software development," in *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, 2004, pp. 72-81.

[102] C. P. Bezemer, A. Zaidman, B. Platzbeecker, T. Hurkmans, and A. t Hart, "Enabling multi-tenancy: An industrial experience report," in *Software Maintenance (ICSM), 2010 IEEE International Conference on*, 2010, pp. 1-8.

[103] "Visio, http://office.microsoft.com/en-001/visio/ [July, 2015]."

[104] "ArgoUML, http://argouml.tigris.org/ [July, 2015]."

[105] "Microsoft Office, http://www.microsoft.com/Office [July, 2015]."

[106] "Google Docs, http://www.docs.google.com [July, 2015]."

[107] M. A. Babar and I. Gorton, "A Tool for Managing Software Architecture Knowledge," presented at the Proceedings of the Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent, 2007.

[108] B. Decker, E. Ras, J. Rech, B. Klein, and C. Hoecht, "Self-organized reuse of software engineering knowledge supported by semantic wikis," in *Proceedings of the Workshop on Semantic Web Enabled Software Engineering (SWESE)*, 2005.

[109] A. Segev and Q. Z. Sheng, "Bootstrapping ontologies for web services," *Services Computing, IEEE Transactions on,* vol. 5, pp. 33-44, 2012.

[110] Y. Zhao, J. Dong, and T. Peng, "Ontology Classification for Semantic-Web-Based Software Engineering," *Services Computing, IEEE Transactions on,* vol. 2, pp. 303-317, 2009.

[111] V. Uren, P. Cimiano, J. Iria, S. Handschuh, M. Vargas-Vera, E. Motta, *et al.*, "Semantic annotation for knowledge management: Requirements and a survey of the state of the art," *Web Semantics: Science, Services and Agents on the World Wide Web,* vol. 4, pp. 14-28, 1 2006.

[112] J. R. Hilera, Ferna, x, and L. ndez-Sanz, "Developing Domain-Ontologies to Improve Sofware Engineering Knowledge," in *Software Engineering Advances (ICSEA), 2010 Fifth International Conference on*, 2010, pp. 380-383.
[113] S. Nešić, "Semantic Document Model to Enhance Data and Knowledge Interoperability," in *Web 2.0 & Semantic Web*. vol. 6, V. Devedžić and D. Gaševic, Eds., ed: Springer US, 2009, pp. 135-160.
[114] D. Baxter, J. Gao, K. Case, J. Harding, B. Young, S. Cochrane*, et al.*, "A framework to integrate design knowledge reuse and requirements management in engineering design," *Robotics and Computer-Integrated Manufacturing,* vol. 24, pp. 585-593, 8 2008.
[115] D. Baxter, J. Gao, K. Case, J. Harding, B. Young, S. Cochrane*, et al.*, "An engineering design knowledge reuse methodology using process modelling," *Research in Engineering Design,* vol. 18, pp. 37-48, 2007/05/01 2007.
[116] O. Zimmermann, C. Miksovic, and J. M. Küster, "Reference architecture, metamodel, and modeling principles for architectural knowledge management in information technology services," *Journal of Systems and Software,* vol. 85, pp. 2014-2033, 9 2012.
[117] F. Zahedi, "The analytic hierarchy process-a survey of the method and its applications," *interfaces,* vol. 16, pp. 96-108, 1986.
[118] F. Baader, I. Horrocks, and U. Sattler, "Description Logics," in *Handbook on Ontologies*, S. Staab and R. Studer, Eds., ed: Springer Berlin Heidelberg, 2004, pp. 3-28.
[119] "Dublin Core Metadata Initiative, http://dublincore.org/documents/abstract-model/ [July, 2015]."
[120] "IEEE Standard for Learning Object Metadata," *IEEE Std 1484.12.1-2002,* pp. i-32, 2002.
[121] "SPARQL, http://www.w3.org/TR/sparql11-query/ [July, 2015]."
[122] M. Boškovic, E. Bagheri, G. Grossmann, D. Gaševic, and M. Stumptner, "Towards integration of semantically enabled service families in the cloud," *WS2,* p. 58, 2011.
[123] M.-C. Valiente, E. Garcia-Barriocanal, and M.-A. Sicilia, "Applying ontology-based models for supporting integrated software development and it service management processes," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on,* vol. 42, pp. 61-74, 2012.
[124] A. April, J.-M. Desharnais, and R. R. Dumke, "A Formalism of Ontology to Support a Software Maintenance Knowledge-based System," in *SEKE*, 2006, pp. 331-336.
[125] B. A. Kitchenham, G. H. Travassos, A. von Mayrhauser, F. Niessink, N. F. Schneidewind, J. Singer*, et al.*, "Towards an ontology of software maintenance," *Journal of Software Maintenance,* vol. 11, pp. 365-389, 1999.

[126] L. Liao, Y. Qu, and H. K. Leung, "An Ontology-based Approach to Express Software Processes."
[127] I. N. Athanasiadis, F. Villa, and A.-E. Rizzoli, "Enabling knowledge-based software engineering through semantic-object-relational mappings," in *Proceedings of the 3rd International Workshop on Semantic Web Enabled Software Engineering*, 2007.
[128] D. Ameller and X. Franch, "Ontology-Based Architectural Knowledge Representation: Structural Elements Module," in *Advanced Information Systems Engineering Workshops*. vol. 83, C. Salinesi and O. Pastor, Eds., ed: Springer Berlin Heidelberg, 2011, pp. 296-301.
[129] B. Antunes, P. Gomes, and N. Seco, "SRS: a software reuse system based on the semantic web," in *3rd International Workshop on Semantic Web Enabled Software Engineering (SWESE)*, 2007.
[130] H.-J. Happel, A. Korthaus, S. Seedorf, and P. Tomczyk, "KOntoR: an ontology-enabled approach to software reuse," in *In: Proc. Of The 18Th Int. Conf. On Software Engineering And Knowledge Engineering*, 2006.
[131] R. Witte, Y. Zhang, and J. Rilling, "Empowering software maintainers with semantic web technologies," in *The Semantic Web: Research and Applications*, ed: Springer, 2007, pp. 37-52.
[132] Y. Zhang, R. Witte, J. Rilling, and V. Haarslev, "An ontology-based approach for traceability recovery," in *3rd International Workshop on Metamodels, Schemas, Grammars, and Ontologies for Reverse Engineering (ATEM 2006), Genoa*, 2006, pp. 36-43.
[133] R. C. de Boer, P. Lago, A. Telea, and H. Van Vliet, "Ontology-driven visualization of architectural design decisions," in *Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on*, 2009, pp. 51-60.
[134] R. C. de Boer and H. Van Vliet, "QuOnt: an ontology for the reuse of quality criteria," in *Sharing and Reusing Architectural Knowledge, 2009. SHARK '09. ICSE Workshop on*, 2009, pp. 57-64.
[135] A. Tang, L. Peng, and H. van Vliet, "Software Architecture Documentation: The Road Ahead," in *Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on*, 2011, pp. 252-255.
[136] K. A. de Graaf, P. Liang, A. Tang, W. R. van Hage, and H. van Vliet, "An exploratory study on ontology engineering for software architecture documentation," *Computers in Industry*.
[137] K. A. de Graaf, A. Tang, L. Peng, and H. Van Vliet, "Ontology-based Software Architecture Documentation," in *Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on*, 2012, pp. 121-130.
[138] S. Henninger and P. Ashokkumar, "An ontology-based metamodel for software patterns," 2006.

[139] H. Zhou, H. Yang, and A. Hugill, "An Ontology-Based Approach to Reengineering Enterprise Software for Cloud Computing," in *Computer Software and Applications Conference (COMPSAC), 2010 IEEE 34th Annual*, 2010, pp. 383-388.
[140] H. Kaiya and M. Saeki, "Ontology based requirements analysis: lightweight semantic processing approach," in *Quality Software, 2005.(QSIC 2005). Fifth International Conference on*, 2005, pp. 223-230.
[141] S. C. Brandt, J. Morbach, M. Miatidis, M. Theißen, M. Jarke, and W. Marquardt, "An ontology-based approach to knowledge management in design processes," *Computers & Chemical Engineering,* vol. 32, pp. 320-342, 2008.
[142] J. Rilling, Y. Zhang, W. J. Meng, R. Witte, V. Haarslev, and P. Charland, "A unified ontology-based process model for software maintenance and comprehension," in *Models in Software Engineering*, ed: Springer, 2007, pp. 56-65.
[143] P. Wongthongtham, E. Chang, T. S. Dillon, and I. Sommerville, "Software Engineering Ontologies and Their Implementation," in *IASTED Conf. on Software Engineering*, 2005, pp. 208-213.
[144] R. L. Grossman, "The Case for Cloud Computing," *IT Professional,* vol. 11, pp. 23-27, 2009.
[145] "Cloud9 IDE, https://c9.io/ [July, 2015]."
[146] "Griffy, https://www.gliffy.com/ [July, 2015]."
[147] P. Yara, R. Ramachandran, G. Balasubramanian, K. Muthuswamy, and D. Chandrasekar, "Global Software Development with Cloud Platforms," in *Software Engineering Approaches for Offshore and Outsourced Development*. vol. 35, O. Gotel, M. Joseph, and B. Meyer, Eds., ed: Springer Berlin Heidelberg, 2009, pp. 81-95.
[148] N. Harrison, P. Avgeriou, and U. Zdun, "Using patterns to capture architectural decisions," *Software, IEEE,* vol. 24, pp. 38-45, 2007.
[149] M. N. Huhns and M. P. Singh, "Service-oriented computing: Key concepts and principles," *Internet Computing, IEEE,* vol. 9, pp. 75-81, 2005.
[150] R. T. Fielding, "Architectural styles and the design of network-based software architectures," University of California, Irvine, 2000.
[151] "Resource Description Framework (RDF), http://www.w3.org/RDF/ [July, 2015]."
[152] G. Katsaros, G. Kousiouris, S. V. Gogouvitis, D. Kyriazis, A. Menychtas, and T. Varvarigou, "A Self-adaptive hierarchical monitoring mechanism for Clouds," *Journal of Systems and Software,* vol. 85, pp. 1029-1041, 2012.
[153] A. Almutairi, M. Sarfraz, S. Basalamah, W. Aref, and A. Ghafoor, "A distributed access control architecture for cloud computing," 2011.

[154] J. Bernal Bernabe, J. M. Marin Perez, J. M. Alcaraz Calero, F. J. Garcia Clemente, G. Martinez Perez, and A. F. Gomez Skarmeta, "Semantic-aware multi-tenancy authorization system for cloud architectures," *Future Generation Computer Systems,* vol. 32, pp. 154-167, 2014.
[155] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*: Pearson Education, 1994.
[156] Y. Badr and G. Caplat, "Software-as-a-Service and Versionology: Towards Innovative Service Differentiation," in *2010 24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, ed: IEEE, 2010, pp. 237-243.
[157] "Amazon EC2 APIs, http://aws.amazon.com/sdk-for-java/ [July 2015]."
[158] "Apache Ant, http://ant.apache.org/ [July 2015]."
[159] B. Leiba, "Oauth web authorization protocol," *IEEE Internet Computing,* pp. 74-77, 2012.
[160] "JAX-RS, http://jax-rs-spec.java.net/ [July, 2015]."
[161] "JAX-WS, https://jax-ws.java.net/ [July, 2015]."
[162] "Apache Jena Framework, https://jena.apache.org/ [July, 2015]."
[163] A. J. Riel, *Object-oriented design heuristics* vol. 338: Addison-Wesley Reading, 1996.
[164] "jBPM, http://www.jbpm.org/ [July, 2015]."
[165] "GlassFish, https://glassfish.java.net/downloads/3.1.2.2-final.html [July, 2015]."
[166] "Amazon MySQL RDS, http://aws.amazon.com/rds/mysql/ [July, 2015]."
[167] "Java Persistence APIs, http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html [July, 2015]."
[168] "Amazon, http://aws.amazon.com/s3/ [July, 2015]."
[169] "UML2OWL, http://sourceforge.net/projects/uml2owl/ [July, 2015]."
[170] "Apache Jena Persistence, https://jena.apache.org/documentation/tdb/index.html [July, 2015]."
[171] P. Clements and L. Bass, "Relating business goals to architecturally significant requirements for software systems," DTIC Document2010.
[172] R. A. Calvo, S. T. O'Rourke, J. Jones, K. Yacef, and P. Reimann, "Collaborative Writing Support Tools on the Cloud," *Learning Technologies, IEEE Transactions on,* vol. 4, pp. 88-97, 2011.
[173] L. B. R. Oliveira and E. Y. Nakagawa, "A service-oriented reference architecture for software testing tools," in *Software Architecture*, ed: Springer, 2011, pp. 405-421.
[174] R. Wolvers and T. Seceleanu, "Embedded Systems Design Flows: Integrating Requirements Authoring and Design Tools," in *Software*

Engineering and Advanced Applications (SEAA), 2013 39th EUROMICRO Conference on, 2013, pp. 244-251.

[175] M. Biehl, J. De Sosa, M. Torngren, and O. Diaz, "Efficient Construction of Presentation Integration for Web-Based and Desktop Development Tools," in *Computer Software and Applications Conference Workshops (COMPSACW), 2013 IEEE 37th Annual*, 2013, pp. 697-702.

[176] R. Khare, M. Guntersdorfer, P. Oreizy, N. Medvidovic, and R. N. Taylor, "xADL: enabling architecture-centric tool integration with XML," in *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on*, 2001, p. 9.

[177] M. Ali Babar, "A framework for groupware-supported software architecture evaluation process in global software development," *Journal of Software: Evolution and Process,* vol. 24, pp. 207-229, 2012.

[178] R. Comerford, "Mecha... what?[mechatronics]," *Spectrum, IEEE,* vol. 31, pp. 46-49, 1994.

[179] S. K. Garg, S. Versteeg, and R. Buyya, "A framework for ranking of cloud computing services," *Future Generation Computer Systems,* vol. 29, pp. 1012-1023, 6 2013.

[180] K. Lee, K. C. Kang, and J. Lee, "Concepts and Guidelines of Feature Modeling for Product Line Software Engineering," presented at the Proceedings of the 7th International Conference on Software Reuse: Methods, Techniques, and Tools, 2002.

[181] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier*, et al.*, *Aspect-oriented programming*: Springer, 1997.

[182] J. Rao and X. Su, "A survey of automated web service composition methods," in *Semantic Web Services and Web Process Composition*, ed: Springer, 2005, pp. 43-54.