Future Generation Computer Systems

A Reference Architecture for provisioning of Tools as a Service: Meta-model, Ontologies and Design Elements

Muhammad Aufeef Chauhan[a,*], Muhammad Ali Babar[a,b], Quan Z. Sheng[b]

[a] *Software and Systems Section, IT University of Copenhagen, Copenhagen, Denmark*
[b] *The University of Adelaide, Adelaide, Australia*

Keywords:
Cloud computing
Software reference architecture
Tools as a Service (TaaS)
Meta-model
Ontologies

A B S T R A C T

Software Architecture (SA) plays a critical role in designing, developing and evolving cloud-based platforms that can be used to provision different types of services for consumers on demand. In this paper, we present a Reference Architecture (RA) for designing cloud-based Tools as a service SPACE (TSPACE), which can provision a bundled suite of tools following the Software as a Service (SaaS) model. The reference architecture has been designed by leveraging information structuring approaches and by using well-known architecture design principles and patterns. The RA has been documented using view-based approach and has been presented in terms of its context, goals, the RA meta-model, information structuring and relationship models using ontologies and components of the RA. We have demonstrated the feasibility and applicability of the RA with the help of a prototype and have used the prototype to provision software architecting tools. We have also evaluated the RA in terms of effectiveness of the design decisions and the RA's completeness and feasibility using scenario-based architecture evaluation method. The proposed TSPACE RA can provide valuable insights to information structure approaches and guidelines for designing and implementing TSPACE for various domains.

1. Introduction

Provisioning of tools in a Tools as a service SPACE (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 facilitating the selection and provisioning of the tools but also needs to provide seamless operations of the tools in terms of distribution of the activities over various tools and integration among the artifacts that are generated and 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, majority of the tools that are used for architecture modeling such as Microsoft Visio[1] and ArgoUML[2] are developed on top of desktop-based paradigm. The desktop and cloud-based word processing tools (e.g., Microsoft Office Suite[3] and Google Docs[4]) and specialized Web based applications (e.g., PakMe [1]) 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 can facilitate the selection of appropriate tools from a pool of available tools and can support a seamless integration among the selected tools. Involvement of the heterogeneous tools requires a solution that is applicable and extendable for various types of the tools, irrespective of the technological paradigms and the tools' vendors.

We have leveraged semantic integration technologies for addressing the abovementioned 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 [2].

* Corresponding author.
E-mail addresses: muac@itu.dk (M.A. Chauhan), ali.babar@adelaide.edu.au (M.A. Babar), michael.sheng@adelaide.edu.au (Q.Z. Sheng).

[1] www.microsoftstore.com/Visio.
[2] http://en.wikipedia.org/wiki/ArgoUML.
[3] www.microsoft.com/Office.
[4] docs.google.com.

Ontologies also facilitate communication, integration and reasoning [2]. Our ontology-based solution enables the provisioning of Tools as a Service (TaaS) for performing different activities using appropriate tools hosted on clouds without requiring the individual tools to focus 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 selected 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 projects 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. The integration mechanism should also provide a support for additional collaboration and awareness activities among the users who perform the activities using the different tools.

Our proposed ontologies provide solution to three main lifecycle phases of the TSPACE. Firstly, the solution supports selection of the tools that are to be provisioned as part of the TSPACE. Once TSPACE is enacted, the solution provides support for semantic integration among the heterogeneous artifacts that are produced and maintained using different tools. Finally, the solution provides support for awareness of the activities that are performed by the stakeholders using the 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 a 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 have to make the final decisions. The main contributions of the research reported in this article are:

- The TSPACE ontologies that can be used to capture concepts of TSPACE, including Capability Ontology, Tools and Artifacts Ontology, Change Ontology and Annotation Ontology.
- A meta-model to characterize TSPACE and to design concrete architecture for providing TSPACE, and the structure of a set of ontologies that formalizes the tools selection, tools provisioning and semantic integration among the artifacts consumed or generated by the hosted tools.
- A detailed description of the TSPACE Reference Architecture (RA) by using multiple levels of abstractions [3] and rationalizing the incorporation of different modules and components in the RA that are described in terms of development view, logical view, process view and deployment view as recommended by view based approaches [3].
- A detailed description of the use of well-known design principles and architectural patterns [4] for designing and reasoning architectures for TSPACE and a selected set of potential solutions to implement the RA.

The organization of the remainder of this paper is as follows. Section 2 describes the TSPACE RA requirements and documentation approach. Section 3 provides details on the TSPACE RA development approach and architecture meta-model. Section 4 elaborates the TSPACE ontologies and Section 5 provides details on the TSPACE RA design. Section 6 presents an overview of TSPACE prototype along with details on evaluation. Section 7 discusses related work and Section 8 concludes the paper.

2. TSPACE RA requirements and documentation approach

This section presents a brief background, functional and non-functional requirements of the TSPACE RA. 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 a TSPACE RA. The reported requirements are based on our previous work on a TaaS infrastructure [5] and a review of the literature on important quality characteristics of the cloud-based systems [6].

2.1. Functional requirements

We have identified the Functional Requirements (FRs) based on the key features required by the RA according to different lifecycle phases of a TSPACE, i.e., tools enactment and provisioning, semantic integration among the artifacts associated with tools after enactment and awareness of the stakeholders' activities during tools' lifecycle. Following are the functional requirements that have been enhanced based on our earlier work [5] in this line of research.

- *FR1—Enactment and provisioning of a TSPACE and associated tools according to the requirements of different activities of a project*: While provisioning tools, the architectural support should also consider the specific location and resource requirements' parameters of the tools.
- *FR2—Semantic integration among artifacts maintained by the tools constituting a TSPACE after enactment*: The TSPACE consists of multiple tools that may have their proprietary formats to store artifacts. The TSPACE architecture should support semantic integration among artifacts generated and maintained by the different tools.
- *FR3—Process centric integration to support collaboration among the tools*: The tools provisioned by TSPACE can also require alignment with organizational process and required support for process centric collaboration among the tools so that artifacts can be exchanged by the tools following specific processes.
- *FR4—Awareness of the operations that are performed on the artifacts during the lifecycle of a TSPACE instance using multiple tools*: Multiple artifacts are produced or consumed during the lifecycle of a specific project for which a TSPACE is instantiated. Hence, there is a need to raise awareness about users' activities associated with the operations that are performed on the artifacts.

2.2. Quality requirements

The quality (i.e., non-functional) requirements of a system are classified into two categories: (i) design time requirements that are discernable while a system is being designed and (ii) runtime requirements that are discernable once a system is operational [7,8]. The following are the design time and runtime quality requirements for a TSPACE:

- *QR1—Automated Provisioning*: A RA shall support automated provisioning of a TSPACE so that the required tools can be acquired automatically for a project based on the constraints on the location of the tools.
- *QR2—Flexibility*: As selection of the tools in a specific instance of a TSPACE depends upon the activities to be performed within a project, a RA shall be flexible enough to provide semantic integration, awareness and traceability support for different types of the tools.

Table 1
Reference architecture documentation requirements and the corresponding solutions.

Dimension	Sub-dimension	TSPACE 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 implementing a prototype.

- *QR*3—*Interoperability*: A RA shall provide semantic integration, awareness, and traceability support for different types of artifacts (e.g., knowledge management, textual documentation and UML modeling tools used for software architecture related activities).
- *QR*4—*Completeness, Feasibility and Applicability*: Bass et al. [7] have presented a number of general quality requirements including completeness, feasibility and applicability. Completeness of a TSPACE RA is important so that it can serve as a guiding model for designing a specific instance of a TSPACE. It should be feasible that a RA is implementable. The applicability quality characteristic is also important so that a RA can be used to design and evaluate a concrete architecture of a TSPACE.
- *QR*5—*Modifiability and Integration*: The tools associated with a TSPACE may come from different vendors. Those tools can be provisioned using public, private or hybrid deployment models. Hence, a RA shall support modifiability and seamless integration among different modules of a TSPACE with the provisioned tools that can be provided by different vendors.
- *QR*6—*Multi-tenancy*: Being a Cloud-based infrastructure, a TSPACE needs to be a multi-tenant platform (with architectural support). Each TSPACE instance can have its own set of tools and rules for awareness and traceability. A particular tenant should be able to access all its specified features and configurations.

2.3. Reference architecture documentation requirements

Since a RA provides valuable guidelines for designing a concrete architecture, it is important to describe a RA as comprehensively as possible and in an easy-to-understand manner. We describe the proposed RA using a systematic approach that advocates the use of context, goal and design dimensions of a RA [9,10]. The context dimension covers the purpose, the development organization, and maturity stage (e.g., preliminary or classic) of a RA [9]. The goal dimension encompasses business goals and quality attributes as well as the purpose of defining a RA (e.g., to standardize concrete architecture or to facilitate design of concrete architecture). The design dimension elaborates whether the RA is concrete or abstract; and whether the RA has been described using formal, semiformal or informal approaches. Avgeriou et al. [10] propose that a RA description should address three main constituents: (i) description of the approach used to document the RA, (ii) guidelines on instantiation of the RA, and (iii) evaluation of the RA corresponding to desired quality attributes. Table 1 lists different dimensions of the RA documentation and our proposed TSPACE solution.

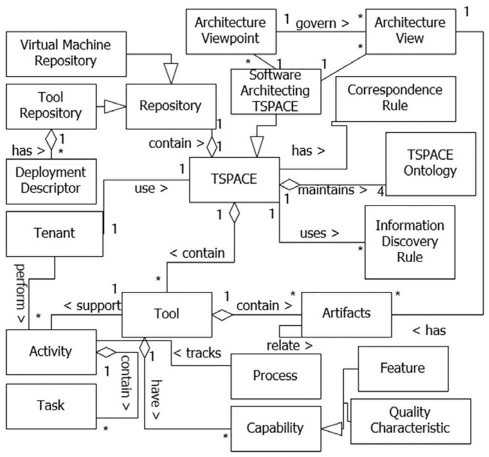


Fig. 1. TSPACE meta-model.

3. TSPACE reference development approach and architecture meta-model

One of the core elements of the proposed reference architecture is a meta-model, which characterizes the elements of a TSPACE and the relations among the elements (Fig. 1). TSPACE meta-model drives the detailed design of TSPACE reference architecture. 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 [11] and ISO/IEC/IEEE 42010:2011 [12]. The extended TSPACE meta-model is shown in Fig. 1. The meta-model shows an abstract TSPACE and its specialization for architecting TSPACE (i.e., TSPACE instance for the tools that are used for software architecting activities). TSPACE meta-model provides a reference points for detailed design of the TSPACE ontologies (to be described in Section 4) and TSPACE RA (to be described in Section 5).

The details on different elements of the meta-model are as follows. A user (tenant) can associate the required tools with a TSPACE in two ways: (i) the tools can be provisioned by third party vendors and are integrated with TSPACE via plug-ins, and (ii) 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 Open Source community. These tools need architecture 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 performed 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 Fig. 1 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.

4. TSPACE ontologies

As described in the introduction that our research on TSPACE has been motivated by the need of having an 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 artifacts using the tools, and support to work with decentralized artifacts using the tools [5]. The key quality requirements of TSPACE require support for bundling multiple tools together in a suite because different stakeholders may have different requirements of tools to perform specific activities. In order to provision TaaS, TSPACE should provide support for mechanisms through which (i) capabilities (functional and non-functional characteristics) supported by the tools and required by the stakeholders can be captured, (ii) related artifacts and data elements maintained among different tools can be associated with each other, and (iii) an awareness mechanism through which notifications of operations and changes can be propagated across tools, which are provisioned as a part of TSPACE.

An ontology is defined as "a formal and explicit specification of a shared conceptualization" and consists of shared vocabulary that can be used to model a specific domain [13]. Ontologies are widely used to define semantic relationships among data and to maintain knowledge of semantic relationships. The knowledge is often maintained using Web-based application such as Semantic Wikis [14]. Annotating digital documents is a common strategy to adapt digital documents to the Web [15]. Ontologies are an effective way of modeling, sharing and reusing organizational knowledge [16].

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

$$\mathrm{TspaceOnt} = \langle \mathrm{CapOnt}, \mathrm{ArtToolOnt}, \mathrm{ChaOnt}, \mathrm{AnnOnt} \rangle.$$

A brief description of each specialization is as follows:

- *Capability Ontology* (*CapOnt*) is used to capture 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 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 the context of multiple artifacts. *AnnOnt* is used to annotate artifacts that are produced or consumed in a TSPACE instance, establish 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. The details of the ontologies and their constituting elements are to be described in the following sections.

The strategy to build ontologies for a specific domain is a critical step. Two different approaches are used to build ontologies: *manual* and *automated* [17]. The manual approach is based on expert knowledge whilst the automated approach is based on information extraction and natural language processing techniques [17]. The automated ontology generation approach is used to extract concepts from the data and structure the concepts in hierarchical order [18], however the automated techniques cannot be used to identify the complex relations between the concepts associated with a particular domain.

Because of the 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 semiautomated 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 paper, we focus on software architecting domain, and our abstract ontology model is based on conceptual architecture documentation meta-models IEEE 1471-2000 [11] and ISO/IEC/IEEE 42010:2011 [12]. 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.

4.1. 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 software architecture such as documentation of architecture design decisions [7], tradeoffs made during the design, architecture patterns and styles [19] that are chosen to implement the design decisions and models of the architecture using different views [3]. It is important to have a consolidated view of different activities carried out using different tools. The activities, tasks and artifacts should be linked to a TSPACE instance for establishing and maintaining relations among them.

There are some important aspects of the activities and the processes that need to be considered while defining ontologies and annotation. Artifacts and process, reuse and management should be treated as a process not as an event [20]. There is also a need to record and track actions and events throughout software engineering process [20]. Process, task and product knowledge are considered key elements to reuse system design [21]. 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 [22]. It facilitates information discovery and analysis. In the following subsections, we describe in details of 4-tuple elements of TspaceOnt.

4.1.1. 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 Fig. 2. 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 with *capableOf* relationship. The members of functional and non-functional capabilities are associated with root elements with the *support* relationship.

Fig. 2(a) shows 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 has a capability to provide a number of runtime quality attributes (such as secured access, support for multi-tenancy and location specific enactment) that is represented as *Quality Capability*.

Fig. 2(b), (c) and (d) show the 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 [1]) and architecture modeling tools (Microsoft Visio). In the diagrams, only some of the functional and quality capabilities are presented. Individual capabilities of the tools are combined to formulate the aggregated capability of TSPACE, as shown in Fig. 2(e). 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). The capability ontology structure presented in Fig. 2(a) can also be used to seek input of the users required capabilities in a TSPACE instance. Fig. 2(f) 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.

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: (i) the tools that are enacted and provisioned by TSPACE as part of a TSPACE instance on public or private IaaS cloud and (ii) the tools that are enacted by third party tool providers and are integrated with a TSPACE instance by providing support for data integration using 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 $l \cap m$ 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 $n \cap o$ 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) [23] is applied as shown in the 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 the 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

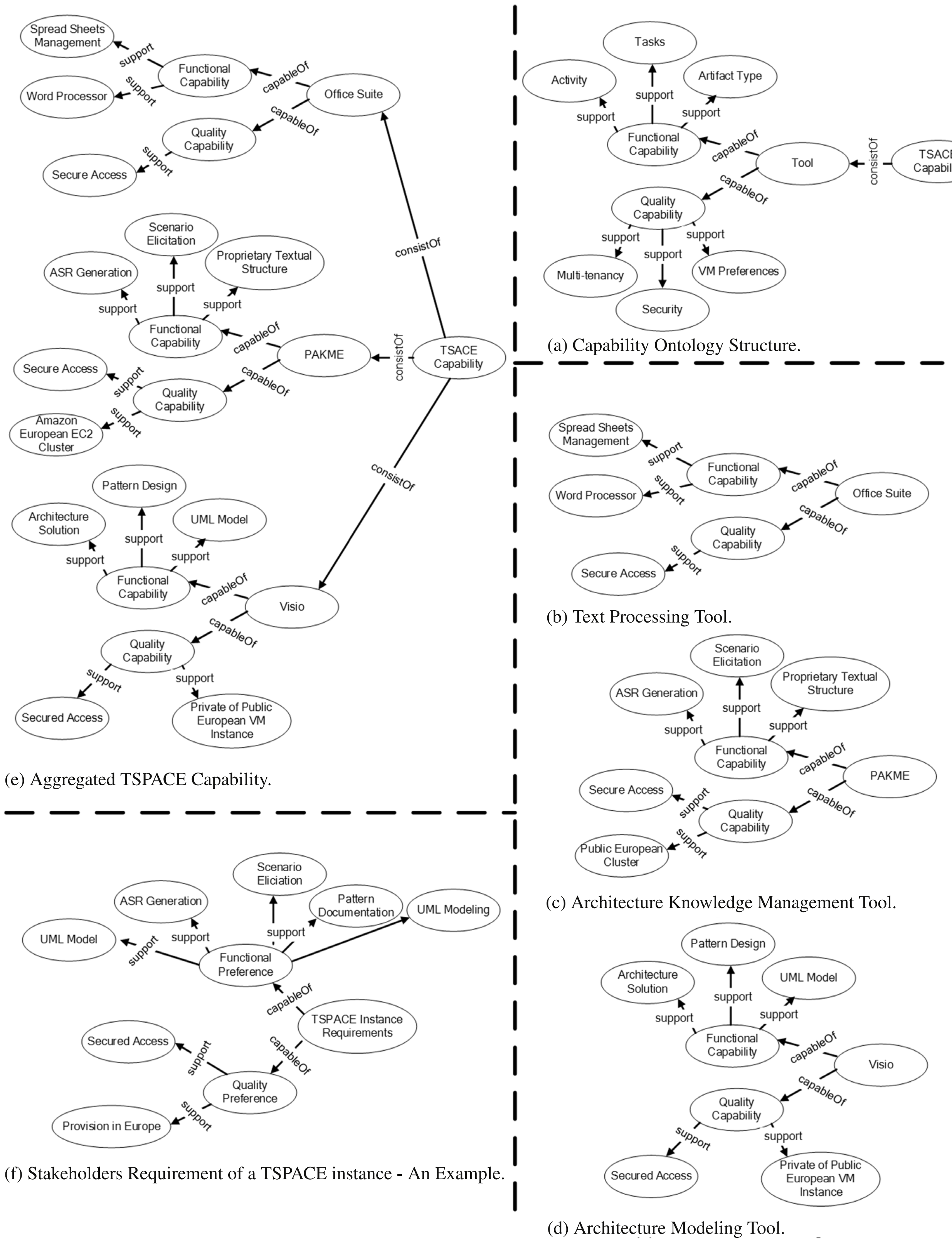


Fig. 2. Capability ontology structure.

indicates not important and nine indicates very important. Tools providing a closer match with the desired features and qualities 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 the tools as a service model, especially when third party *aaS model tools are to be integrated with TSPACE. The capability ontology of each tool (functional/quality feature set) can be populated

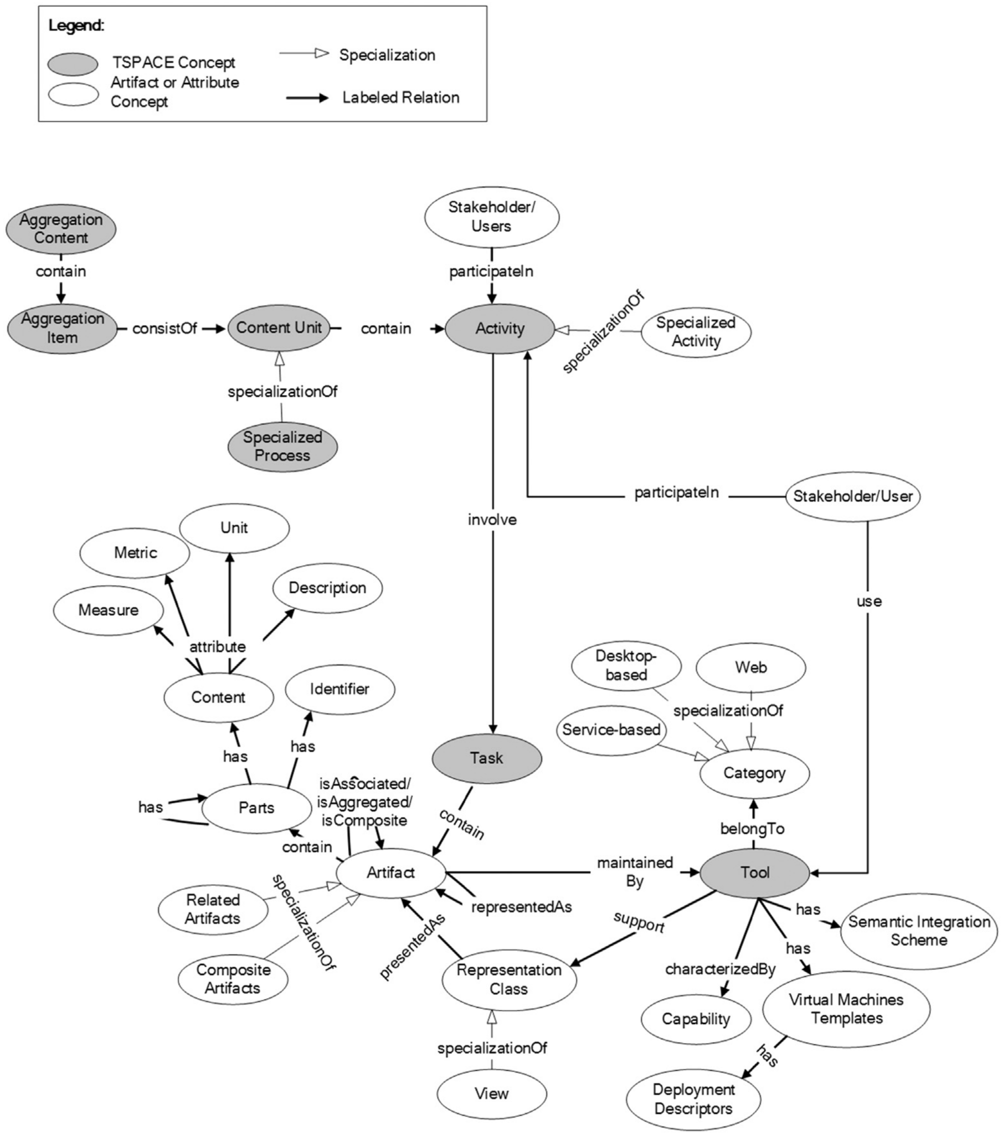


Fig. 3. TSPACE abstract tool and artifact ontology.

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) [18].

4.1.2. 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 the 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 Fig. 3 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 consists of a number of tasks and each task can involve users working on at least one artifact. The artifacts are organized in 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: (i) a unique identifier that identifies an artifact in TSAPCE and (ii) contents of the artifact. The artifact contents can have multiple sub attributes. Description of the contents of the artifacts include 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

Table 2
TSPACE relations to manage the tools and artifacts.

Relationship	Description	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.	isAssociated	Association relationship between two CEs that are at same level of abstraction.
consistOf	Relationship between a parent CU and a child CU.	isAggregated	Aggregation relationship between two CEs such that the one being aggregated can exist without the aggregator.
contain	Containment relationship between parent content element (CE) and child CE such that parents and child are at different levels of abstraction.	isComposite	Composition relationship between two CEs such that the one being composed cannot exist without the composer.
containedBy	Containment relationship between child CE and parent CE. It is inverse of the contain relation.	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.
specializationOf	Specialization of a generalized CE into a specialized CE.	representedAs	Representation of one type of CE with another type of CE with same form. E.g., using textual description.
has	Association between an actor (stakeholder) and a CE.	attribute	An attribute of a CE that represents its property.
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.	support	A particular view that is supported by the respective tool e.g., a scenario view or a use case view.

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 may exist between different elements of TSPACE are listed in Table 2 and are explained in the remainder of this subsection with the help of *ArtToolOnt* in software architecting domain.

The tools that are available for provisioning in TSPACE have capabilities 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. Fig. 3 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 the software architecting domain, architecture views and architecture viewpoints are used [7] and Representation Class concept that is shown in Fig. 3 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 Fig. 4. Containment relationship between different types of elements of TSPACE and the tools that contain the elements is established via the *maintainedBy* property. Specializations of tools are represented via the *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 Fig. 4, *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.

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 Fig. 3, *Artifact*, its specializations and its sub-concepts are the examples of CEs. Fig. 4 shows an instance of CEs with references to architecture design of the TSPACE. In Fig. 4, each of the sub-concepts represents content elements at a 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. 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 Fig. 4 *Requirements and Scenarios* contain two primary sub-concepts: *quality attributes* and *architecture concerns*. Quality attributes may have many specializations. The specialization of a quality attribute is represented via the *specializationOf* relationship. There can be nested specializations and can be represented via the *specializationOf* relationship. Architecture concerns consist of description, metrics and metrics' values. Architecture concerns are framed by architecture viewpoint and are represented by the *framedBy* relationship. An architecture framework aggregates architecture viewpoints. Aggregation relationship of architecture framework with architecture viewpoint and correspondence rules is represented via the *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 the *specializationOf* property.

There can be more specialization of architecture knowledge, although they are not depicted in the figures. Architecture is modeled using different views and is a representation of different viewpoints. This relation is depicted as the *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 [24]. 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 the *presentedAs* property. There can be general association between CEs or different elements of the CEs; association of architecture styles and patterns with models and diagrams. This type of general association is represented by the *associatedWith* property. The high-level relationships of ArtToolOnt ontologies discussed above are listed in Table 2.

4.1.3. *Change ontology (ChaOnt)*

The change ontology tracks changes in the TSPACE's content elements (CE) and relationship between CEs. We extend the change

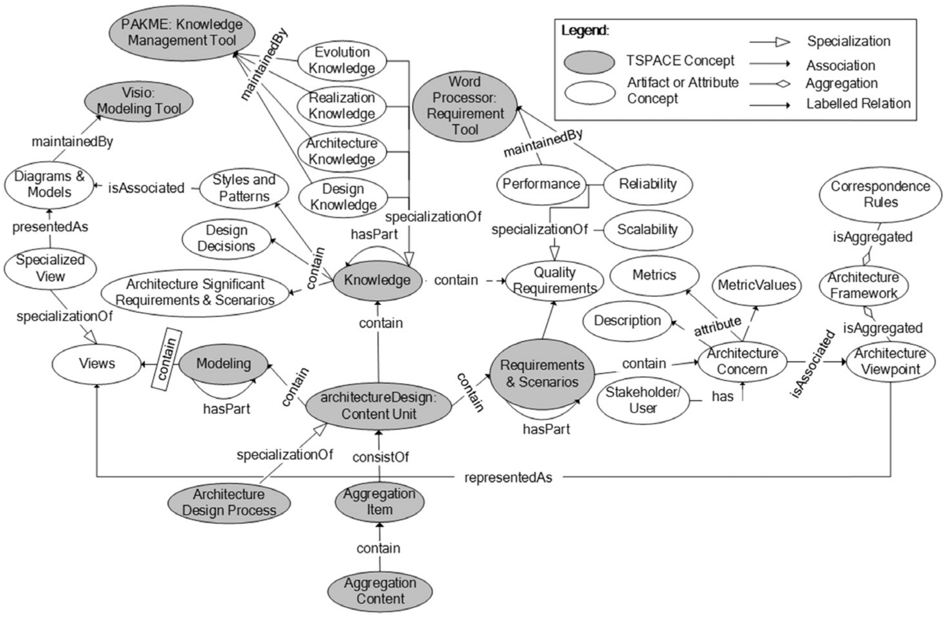


Fig. 4. TSPACE tool and artifact ontology instance example.

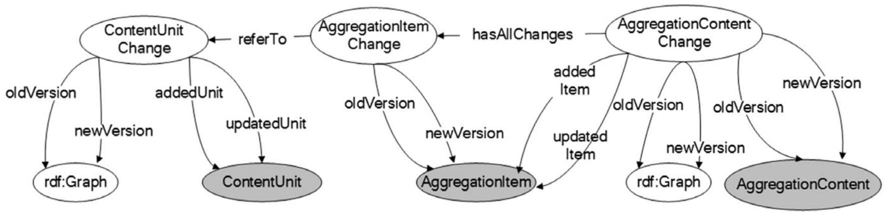


Fig. 5. TSPACE change ontology.

ontology of the semantic document model reported in [17] for the elements of TSPACE. Pictorial representation of the root-level change ontology is presented in Fig. 5. 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.

4.1.4. 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 Fig. 6. We have extended the annotation ontology of the semantic document model reported in [17] for the TSPACE.

Our approach for enabling TaaS leverages annotation ontology, for semantic integration among artifacts maintained by different types of tools used in a specific instance of the TSPACE. The annotation ontology supports plug-ins and data collection probes. The plug-ins, add-ins and probes discover CEs at different levels of abstraction with the help of Annotation Ontology. By introducing 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.

Context Independent Annotation. Artifacts and Tools Ontology that is discussed in Section 4.1.2 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. Semantic document model (SDM) to enhance data and knowledge interoperability for text documents [17] 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

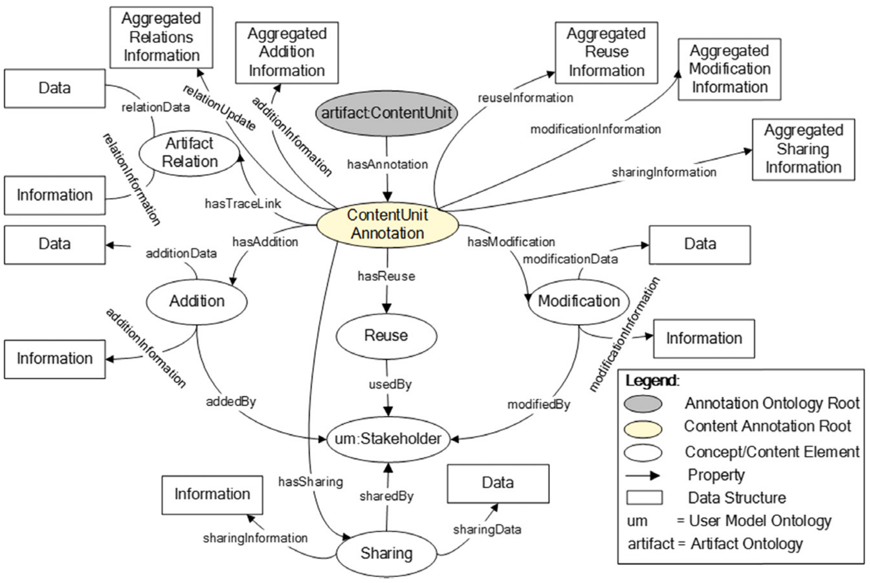


Fig. 6. TSPACE annotation ontology.

which the TSPACE is to be used; hence, in this paper 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 to have the TSPACE is to provide a customizable and semantically integrated suite of tools by bundling multiple tools together [5]. 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 type of interaction, stakeholder who is participating in the interaction and 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 that are corresponding to aggregated information of each concept. Aggregated information is maintained to have a consolidated view of stakeholders' activities on the artifacts that are being used as part of an activity (e.g., software architecture design). Fig. 6 shows pictorial representation of annotation ontology and usage metadata.

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 (Fig. 3) concept defined by the TSPACE ontology. We have introduced the DesignSpaceAnnotation concept in our annotation ontology that acts as a metadata binder for AggregationItem (Fig. 6). 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, and (ii) TraceElement to identify dependencies and trace links between content elements of the TSPACE.

4.2. Use of ontologies for notifications and information extraction

Annotation ontology along with change ontology also supports notifications for collaborative activities that are being performed

Table 3
Sample rules for TSPACE notifications.

Notification	Formation
R1: Addition	$\forall_{x,y}$: $\langle\text{NotifyAddition}\rangle_y \implies \text{Parent}_{x,y} \wedge \langle\text{Addition}\rangle_x$
R2: Modification	$\forall_{x,y}$: $\langle\text{NotifyUpdate}\rangle_y \implies \text{Parent}_{x,y} \wedge \langle\text{Modification}\rangle_x$
R3: Sharing	$\forall_x$: $\langle\text{NotifySharing}\rangle_x \implies$ $\langle\text{Select}\rangle\langle\text{User1}\rangle_x \wedge \langle\text{Select}\rangle\langle\text{User2}\rangle_x$
R4: Conflict	$\forall_{x,y}$: $\langle\text{NotifyConflict}\rangle_y \implies$ $\text{Parent}_{z,x} \wedge \text{Parent}_{z,y} \wedge \langle\text{Modification}\rangle_x \rightarrow \langle\text{Conflict}\rangle_y$

using multiple tools in a TSPACE. The rules use elements of annotation and change ontology to raise awareness on 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, $\text{Relation}_{x,y}$ be a relationship that exists between x and y (e.g., $\text{Parent}_{x,y}$ represents x is parents of y), $\langle\text{Action}\rangle_x$ is an action triggered for x, and $\langle\text{Select}\rangle\langle U\rangle_x$ as selection of a content element to be used in a particular activity by a user U. Table 3 shows some sample notifications rules corresponding to the addition, modification, conflict identification and sharing of the content elements. These rules can be implemented using SPARQL[5] queries in combination with complimentary algorithms.

Algorithm 2 depicts the details of the algorithm to fire addition and modification notifications when a content element of any of the parent (of the content element) that is under investigation is modified, or additional attributes are added.

```
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] http://www.w3.org/TR/sparql11-query/.

5. TSPACE architecture design and decomposition of architecture elements

We have designed the 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 [7] and several architectural styles. TSPACE reference architecture consists of two abstraction layers. The 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 integrability. 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 for managing process-centric integration to facilitate exchange of the artifacts among the tools.

We present TSPACE reference architecture at two 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 Fig. 7 shows the notations that are used in the diagrams of the reference architecture.

According to the functional requirements (Section 2.1), 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. Fig. 7 provides an overall representation of the reference architecture (the development view). The modules at the first level of decomposition are organized following the layered architecture style [7]. 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.

The *Tools Selection and Provisioning Manager* enables users to select the tools that are suitable for the activities to be performed. The *Integration Manager* supports process centric and semantic integration among the tools and the artifacts that are maintained by the provisioned tools. The *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 corresponding domain in which the reference architecture is used. The *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 3). All the tools constituting a TSPACE and the relevant artifacts are annotated using the Annotation Ontology of the semantic integration model (Section 4.1.4).

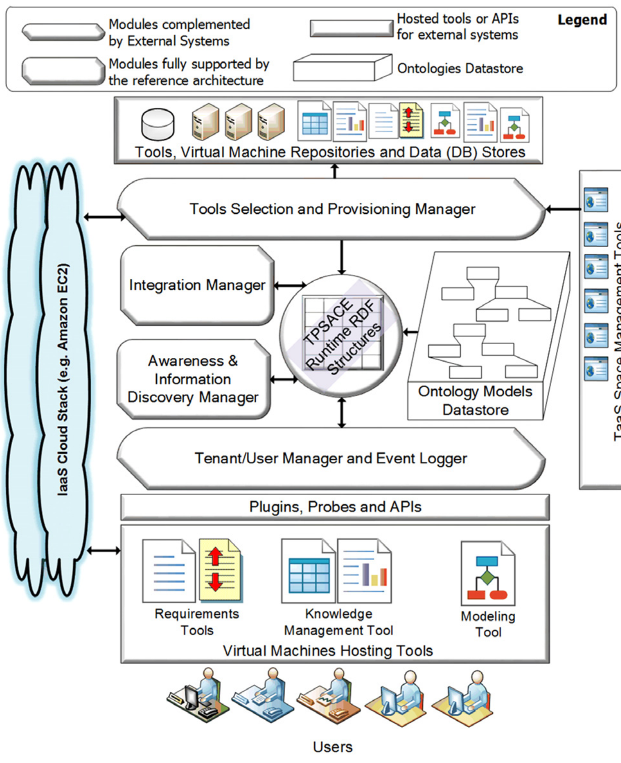

Modules complemented by External Systems
Modules fully supported by the reference architecture
Hosted tools or APIs for external systems
Ontologies Datastore
Legend
Tools, Virtual Machine Repositories and Data (DB) Stores
IaaS Cloud Stack (e.g. Amazon EC2)
Tools Selection and Provisioning Manager
Integration Manager
TPSACE Runtime RDF Structures
Awareness & Information Discovery Manager
Ontology Models Datastore
Tenant/User Manager and Event Logger
Plugins, Probes and APIs
Requirements Tools
Knowledge Management Tool
Modeling Tool
Virtual Machines Hosting Tools
TaaS Space Management Tools
Users

Fig. 7. TSPACE architecture—first level decomposition.

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 [25] to support integration among components and modifiability (QR5). We have also described the collaboration (using collaboration diagrams) between the components of each module to show data exchange between the components.

5.1. Details of the reference architecture components

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 avoid cluttering. The diagrams in this section are represented using Unified Modeling Language (UML).

5.1.1. Decomposition of tools selection and provisioning manager

The 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. Fig. 8 shows the details of the components and classes encompassing the *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.

The TSPACE repository consists of Virtual Machine Templates and Elements. *Elements* are an abstract representation of *Tools* and *OperationalServices*. As indicated earlier in this paper, 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

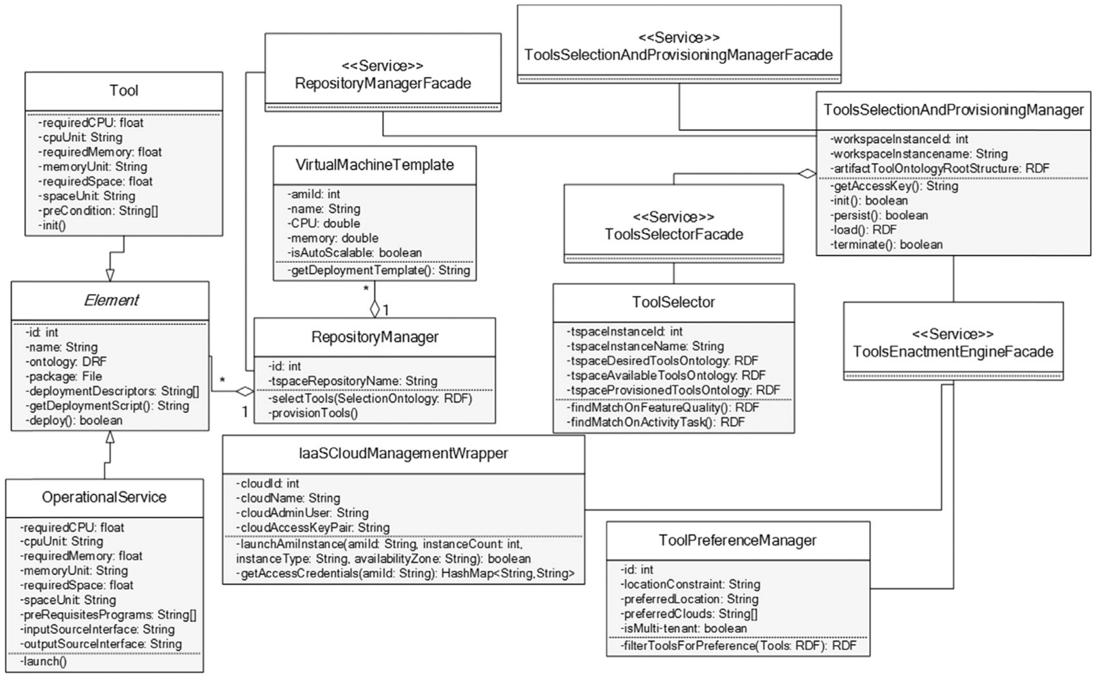

Tool
-requiredCPU: float
-cpuUnit: String
-requiredMemory: float
-memoryUnit: String
-requiredSpace: float
-spaceUnit: String
-preCondition: String[]
-init()
<<Service>> RepositoryManagerFacade
<<Service>> ToolsSelectionAndProvisioningManagerFacade
ToolsSelectionAndProvisioningManager
-workspaceInstanceId: int
-workspaceInstancename: String
-artifactToolOntologyRootStructure: RDF
-getAccessKey(): String
-init(): boolean
-persist(): boolean
-load(): RDF
-terminate(): boolean
VirtualMachineTemplate
-amiId: int
-name: String
-CPU: double
-memory: double
-isAutoScalable: boolean
-getDeploymentTemplate(): String
<<Service>> ToolsSelectorFacade
Element
-id: int
-name: String
-ontology: DRF
-package: File
-deploymentDescriptors: String[]
-getDeploymentScript(): String
-deploy(): boolean
RepositoryManager
-id: int
-tspaceRepositoryName: String
-selectTools(SelectionOntology: RDF)
-provisionTools()
ToolSelector
-tspaceInstanceId: int
-tspaceInstanceName: String
-tspaceDesiredToolsOntology: RDF
-tspaceAvailableToolsOntology: RDF
-tspaceProvisionedToolsOntology: RDF
-findMatchOnFeatureQuality(): RDF
-findMatchOnActivityTask(): RDF
<<Service>> ToolsEnactmentEngineFacade
*
1
*
1
OperationalService
-requiredCPU: float
-cpuUnit: String
-requiredMemory: float
-memoryUnit: String
-requiredSpace: float
-spaceUnit: String
-preRequisitesPrograms: String[]
-inputSourceInterface: String
-outputSourceInterface: String
-launch()
IaaSCloudManagementWrapper
-cloudId: int
-cloudName: String
-cloudAdminUser: String
-cloudAccessKeyPair: String
-launchAmiInstance(amiId: String, instanceCount: int, instanceType: String, availabilityZone: String): boolean
-getAccessCredentials(amiId: String): HashMap<String,String>
ToolPreferenceManager
-id: int
-locationConstraint: String
-preferredLocation: String
-preferredClouds: String[]
-isMulti-tenant: boolean
-filterToolsForPreference(Tools: RDF): RDF

Fig. 8. Tools selection and provisioning—detailed design.

ToolSelector by establishing a closest match between the required tools and the tools that are available for provisioning. *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:

- If the tools have deployment scripts associated with them and the tools can be deployed remotely (e.g., using Apache Ant[6] scripts), then the tools are deployed on virtual machines and provisioned.
- 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 *ToolSelector* 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, *ToolsSelectionAndProvisioningManager* 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 Fig. 9, which is at higher level of class hierarchy that is shown in Fig. 8. Different elements of initialization factory are shown in Fig. 9(a). For example *TspaceManager* is composed of *TspaceInitializer* and uses its methods to launch tools and ontology instances of a TSPACE instance. Details of the methods and cardinality between different elements are shown in the figure.

6 http://ant.apache.org/.

5.1.2. 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 Fig. 10. In the figure, <<TSPACE>> stereotype shows elements of TSPACE whereas <<Tool>> stereotype shows element of the tools that interact with TSPACE elements.

As TSPACE integration consists of semantic integration and process centric integration, Fig. 10 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 Fig. 10. *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 Fig. 10 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 Fig. 10. 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 notification accordingly. Detail of the methods and cardinality between different elements is shown in Fig. 10.

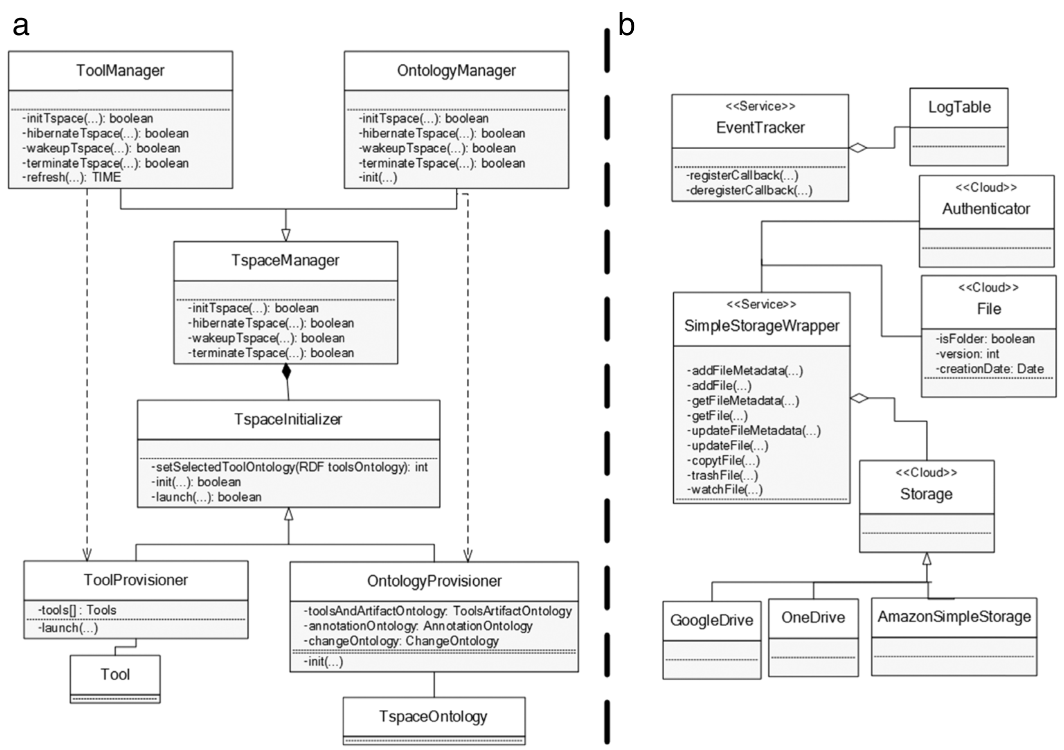


Fig. 9. TSPACE provisioning—initialization factory.

Details of the Simple Storage Manager are shown in Fig. 9(b). In the figure, <<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.

5.1.3. *Decomposition of collaboration and awareness manager*

The detail of Collaboration and Awareness Manager is shown in Fig. 11. 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*. *NotificationRules* and *InformationDiscoveryRules* are used to fetch the desired information from ontology RDFs using SPARQL queries. The *NotificationManager*, the *InformationDiscoveryManager* and the *AnnotationManager* interact with SparqlQueryExecuter to execute the queries on RDF data stores.

NotificationManager can handle two types of notifications: (i) *ChangeNotification* that is triggered when a change is made in semantically integrated artifacts and (ii) *ConflictNotification* when semantically integrated artifacts may present conflicting information. *AnnotationManager* facilitates to semantically integrate the artifacts using annotations. *NotificationManager* interacts with *InformationDiscoveryManager* for SPARQL query execution and transformation of the extracted information to higher levels of abstractions. Detail of the methods and cardinality between different elements is shown in Fig. 11.

5.1.4. *Decomposition of multi-tenancy and authentication*

The core of TSPACE multi-tenancy in combination with tenant and user authentication is presented in Fig. 12. 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. 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 [26] 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 sent by the tools to TSPACE. *ScalabilityController* (e.g., Amazon cloud watch and elastic load balancer[7,8,9]) is an external IaaS monitor that is used to replicate the queues according to defines 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 is shown in Fig. 12.

5.2. *Design decisions summary*

A number of design decisions have been taken to incorporate functional and quality requirements in the TSPACE reference architecture. A summary of the design decisions is presented in Table 4.

7 http://aws.amazon.com/autoscaling/.
8 http://aws.amazon.com/cloudwatch/.
9 http://aws.amazon.com/elasticloadbalancing/.

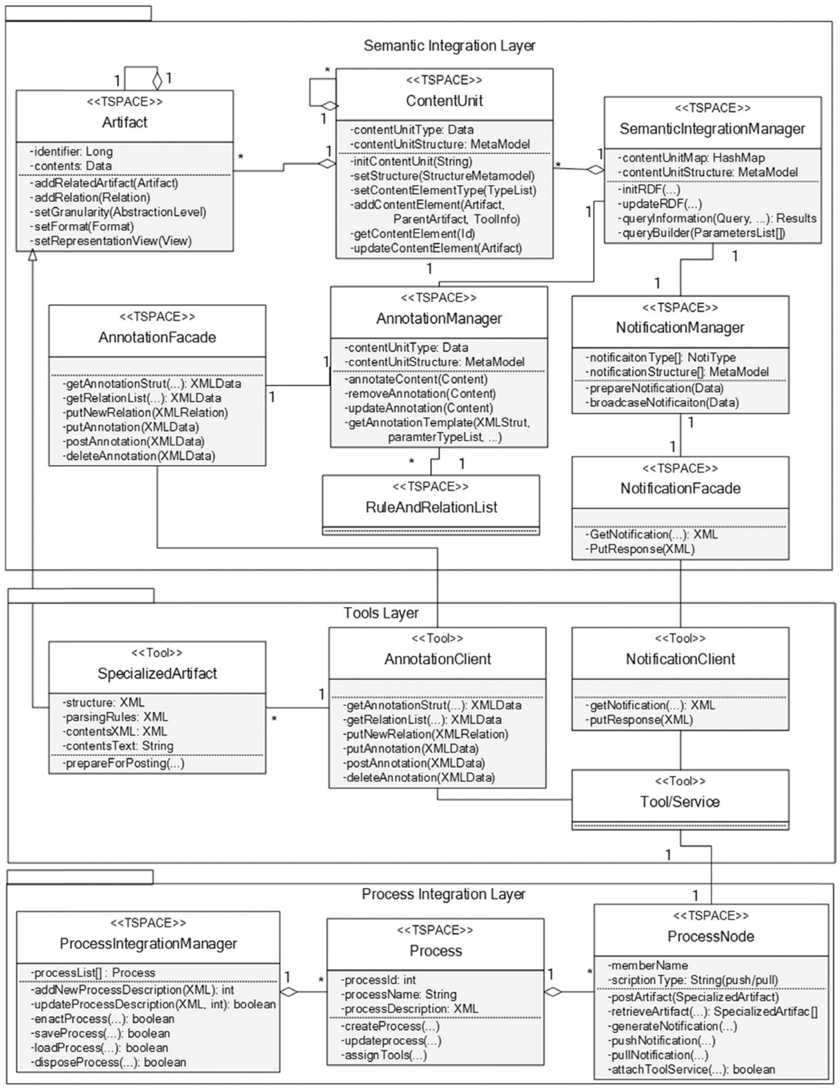

Semantic Integration Layer
<<TSPACE>> Artifact
-identifier: Long
-contents: Data
-addRelatedArtifact(Artifact)
-addRelation(Relation)
-setGranularity(AbstractionLevel)
-setFormat(Format)
-setRepresentationView(View)
<<TSPACE>> ContentUnit
-contentUnitType: Data
-contentUnitStructure: MetaModel
-initContentUnit(String)
-setStructure(StructureMetamodel)
-setContentElementType(TypeList)
-addContentElement(Artifact, ParentArtifact, ToolInfo)
-getContentElement(Id)
-updateContentElement(Artifact)
<<TSPACE>> SemanticIntegrationManager
-contentUnitMap: HashMap
-contentUnitStructure: MetaModel
-initRDF(...)
-updateRDF(...)
-queryInformation(Query, ...): Results
-queryBuilder(ParametersList[])
<<TSPACE>> AnnotationFacade
-getAnnotationStrut(...): XMLData
-getRelationList(...): XMLData
-putNewRelation(XMLRelation)
-putAnnotation(XMLData)
-postAnnotation(XMLData)
-deleteAnnotation(XMLData)
<<TSPACE>> AnnotationManager
-contentUnitType: Data
-contentUnitStructure: MetaModel
-annotateContent(Content)
-removeAnnotation(Content)
-updateAnnotation(Content)
-getAnnotationTemplate(XMLStrut, paramterTypeList, ...)
<<TSPACE>> NotificationManager
-notificaiton Type[]: NotiType
-notificationStructure[]: MetaModel
-prepareNotification(Data)
-broadcaseNotificaiton(Data)
<<TSPACE>> RuleAndRelationList
<<TSPACE>> NotificationFacade
-GetNotification(...): XML
-PutResponse(XML)
Tools Layer
<<Tool>> SpecializedArtifact
-structure: XML
-parsingRules: XML
-contentsXML: XML
-contentsText: String
-prepareForPosting(...)
<<Tool>> AnnotationClient
-getAnnotationStrut(...): XMLData
-getRelationList(...): XMLData
-putNewRelation(XMLRelation)
-putAnnotation(XMLData)
-postAnnotation(XMLData)
-deleteAnnotation(XMLData)
<<Tool>> NotificationClient
-getNotification(...): XML
-putResponse(XML)
<<Tool>> Tool/Service
Process Integration Layer
<<TSPACE>> ProcessIntegrationManager
-processList[] : Process
-addNewProcessDescription(XML): int
-updateProcessDescription(XML, int): boolean
-enactProcess(...): boolean
-saveProcess(...): boolean
-loadProcess(...): boolean
-disposeProcess(...): boolean
<<TSPACE>> Process
-processId: int
-processName: String
-processDescription: XML
-createProcess(...)
-updateprocess(...)
-assignTools(...)
<<TSPACE>> ProcessNode
-memberName
-scriptionType: String(push/pull)
-postArtifact(SpecializedArtifact)
-retrieveArtifact(...): SpecializedArtifac[]
-generateNotification(...)
-pushNotification(...)
-pullNotification(...)
-attachToolService(...): boolean
1
*

Fig. 10. Integration manager—detailed design.

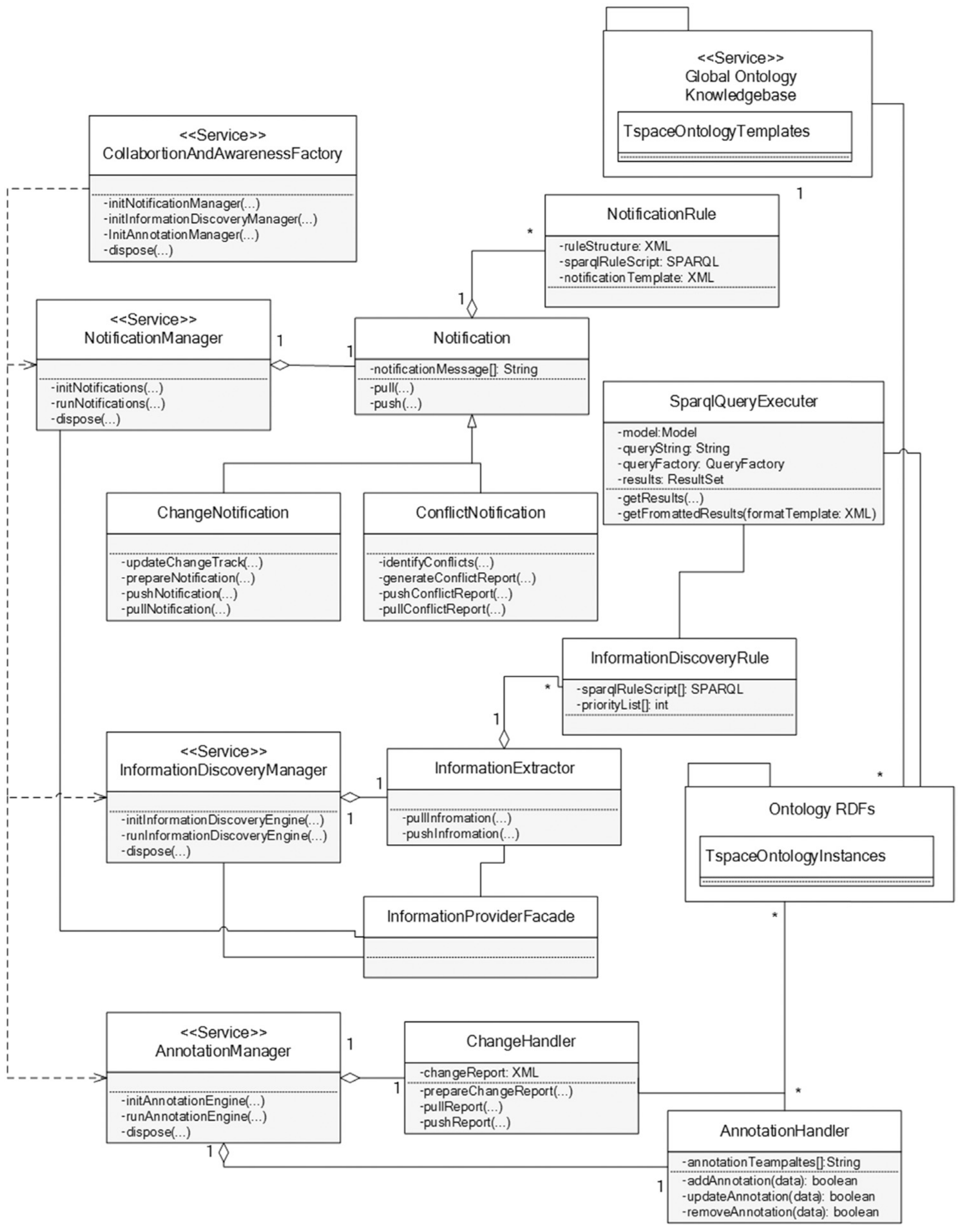
6. Prototype and evaluation of TSPACE reference architecture

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[10] and JAX-WS[11] service technologies. We have used Apache Jena Framework[12] to implement the semantic integration in the prototype. Persistence of TSPACE is handled following principles of Object Oriented Paradigm [27] and Java Persistence APIs (JPA) have been used to store data objects in the underlying database that is used for persistence.

We have used jBPM core library[13] 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.[14] Amazon IaaS cloud[15] has been used to deploy TSPACE prototype and the tools that are provisioned by TSPACE. The core components of TSPACE have been deployed on Amazon EC2.

Windows Server 2012 instance[16] with 8 GB of RAM and 2.4 GHz Intel Xeon processor. Amazon Cloud Watch [8] and Elastic Load Balancer [9] have been attached with the core services to enable autoscaling. Amazon EC2 instances and Amazon Machine Image (AMI) templates [16] have been used to host the tools that are provisioned by TSPACE. Amazon RDS for MySQL[17] have been used for persistence of the data objects. There is a Java Persistence API (JPA)[18] 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 Fig. 1 and ontology as discussed in Section 3) of TSPACE reference architecture as persistence objects and JPA's object to relational mapping features are used. We have also used

10 http://jax-rs-spec.java.net/.
11 https://jax-ws.java.net/.
12 https://jena.apache.org/.
13 http://www.jbpm.org/.
14 https://glassfish.java.net/downloads/3.1.2.2-final.html.
15 http://aws.amazon.com/.
16 http://aws.amazon.com/ec2/.
17 http://aws.amazon.com/rds/mysql/.
18 http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html.



<<Service>> Global Ontology Knowledgebase
TspaceOntologyTemplates
<<Service>> CollabortionAndAwarenessFactory
-initNotificationManager(...)
-initInformationDiscoveryManager(...)
-InitAnnotationManager(...)
-dispose(...)
NotificationRule
-ruleStructure: XML
-sparqlRuleScript: SPARQL
-notificationTemplate: XML
<<Service>> NotificationManager
-initNotifications(...)
-runNotifications(...)
-dispose(...)
Notification
-notificationMessage[]: String
-pull(...)
-push(...)
SparqlQueryExecuter
-model:Model
-queryString: String
-queryFactory: QueryFactory
-results: ResultSet
-getResults(...)
-getFromattedResults(formatTemplate: XML)
ChangeNotification
-updateChangeTrack(...)
-prepareNotification(...)
-pushNotification(...)
-pullNotification(...)
ConflictNotification
-identifyConflicts(...)
-generateConflictReport(...)
-pushConflictReport(...)
-pullConflictReport(...)
InformationDiscoveryRule
-sparqlRuleScript[]: SPARQL
-priorityList[]: int
<<Service>> InformationDiscoveryManager
-initInformationDiscoveryEngine(...)
-runInformationDiscoveryEngine(...)
-dispose(...)
InformationExtractor
-pullInfromation(...)
-pushInfromation(...)
Ontology RDFs
TspaceOntologyInstances
InformationProviderFacade
<<Service>> AnnotationManager
-initAnnotationEngine(...)
-runAnnotationEngine(...)
-dispose(...)
ChangeHandler
-changeReport: XML
-prepareChangeReport(...)
-pullReport(...)
-pushReport(...)
AnnotationHandler
-annotationTeampaltes[]:String
-addAnnotation(data): boolean
-updateAnnotation(data): boolean
-removeAnnotation(data): boolean

Fig. 11. Collaboration and awareness manager—detailed design.

Table 4
Summary of architecture design strategies.

Design decisions	Benefits	Adaptation tactics
Using ontologies to formalize TSPACE constituents (tools, artifacts and operations)	Makes RA dynamic and flexible to support different types of tools	Ontology model can easily be tailored and enhanced for other domains.
Using SOA for TSPACE façade	Positively addresses modifiability of TSPACE RA and its integration with different types of tools	Glue code components can be written to support tools for accessing TSPACE façade.
Centralized repository pattern to share abstract ontology templates	Provides single point of access to the TSPACE ontologies and positively addresses TSPACE flexibility and adaptability	New ontology templates can be added to support specific needs of a TSPACE instance.
Two staged pipes and filters pattern	Positively addresses performance and multi-tenancy	Support for flexible and customizable tenants' identification rules can be provided using SPARQL.
Loosely coupled layered architecture	Positively addresses modifiability and evolvability of TSPACE components	Data caching and indexing can be incorporated to mitigate negative impact of layers on performance.
Use of ontologies as a baseline for tenant specific rules for semantic integration, information discovery and awareness of the operations	Positively addresses multi-tenancy by providing support for tenant specific integration and information discovery rules	More sophisticated support for tenant isolation and indexing can be provided by using approaches such as presented in [28,29].

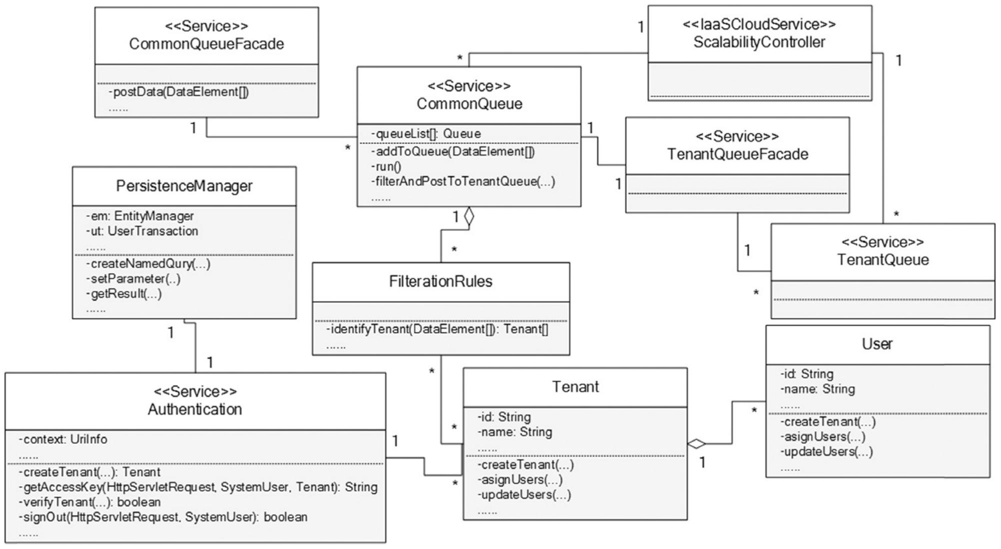


Fig. 12. Multi-tenancy and authentication—detailed design.

Amazon's Simple Storage Service (S3)[19] for storing plain artifacts and data.

We have adopted a multi-facet strategy for the evaluation of TSPACE reference architecture. We have demonstrated the feasibility and applicability of the reference architecture by implementing its prototype. We have combined classical reasoning approach with existing architecture evaluation methods to evaluate effectiveness and completeness of the reference architecture.

6.1. *Evaluation for feasibility using prototype tools*

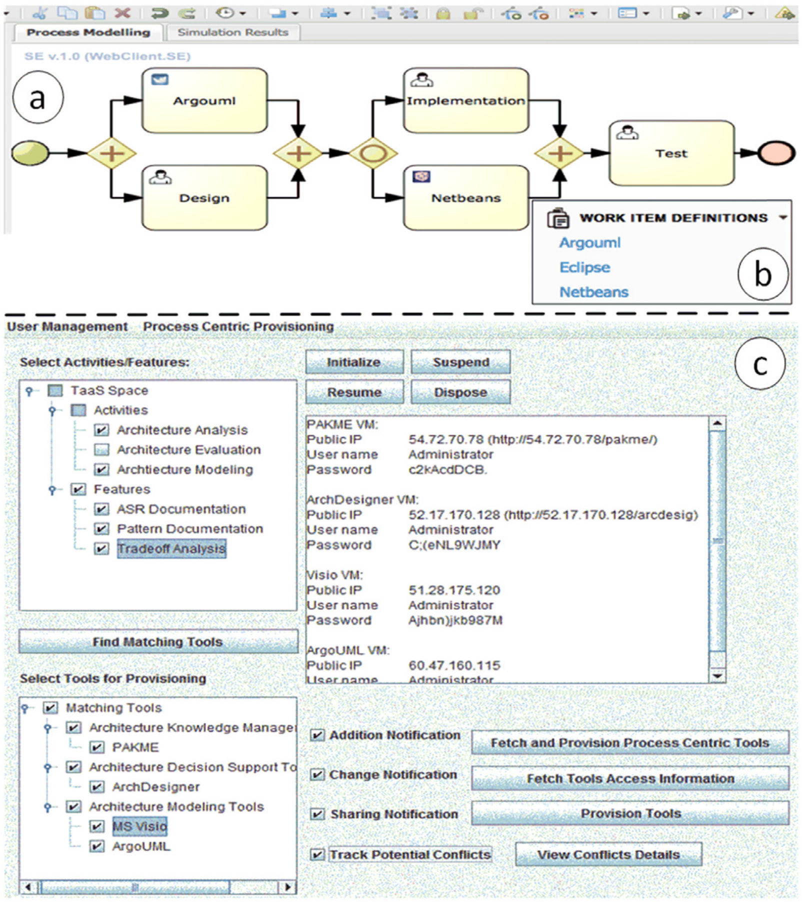
We have developed management applications for TSPACE and have integrated a selected set of tools using TSPACE prototype. Fig. 13 shows management interfaces for TSPACE prototype. Process centric tools provisioning can be achieved by opening process manager from *Process Centric Provisioning* menu of Fig. 13(c). We have enhanced KIE Workbench workflow process modeler[20] to introduce the notion of tools in process workflow designer of KIE. Fig. 13(a) shows GUI that is used for process centric tools provisioning. Every node of the process has two elements: (i) process node stage (e.g., design node or implementation node) and (ii) tools. A tenant can be assigned to each process node stage. All the users belonging to a specific tenant can have access to the attached tool. Once a process is enacted, tools are provisioned on underlying IaaS cloud and their collaboration scheme is defined by the process collaboration engine according to the defined process flow (as described in Section 5.1.2).

Process-centric Tools Provisioning and Integration Manager takes care of exchange of the artifacts among different nodes of the process (as discussed in Section 5.1.2). Fig. 13(a) shows a simple scenario in which a UML design tool ArgoUML, an IDE Netbeans and a test instance is provisioned as part of the process centric provisioning. ArgoUML generates skeleton of the classes, Netbeans IDE takes skeleton of the classes generated by ArgoUML (as input from preceding node of the process) and allow user to write code, and a test instance that takes output of the program and allow testers to run test cases on it.

Fig. 13(c) shows main Graphical User Interface (GUI) of TSPACE for semantically integrated tools provisioning. The GUI is used to specify tools requirements in a TSPACE instance and specify notification. 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. Semantically integrated tools can be provisioned in two different ways using the GUI shown in Fig. 13(c). First way is that the desired activities and features can be selected and TSPACE provides the closest match of the tools that are available in a TSPACE. 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 Fig. 13(c). TSPACE selects the tools that qualify the tools requirements criteria by looking for the closest match of the features that are supported by a tool and required by a tenant, and returns the list of tools that are available for provisioning as shown in bottom left of Fig. 13(c). As figure shows, PAKME, ArchDesigner, Microsoft Visio and ArgoUML qualify search criteria corresponding to specified activities and features shown in Fig. 13(c). After selecting the tools and pressing the *Initialize* button, the tools are provisioned using Amazon Machine Images (AMIs) and access information of the tools is presented. Second way is that the tools can also be directly selected for provisioning from the tree shown in bottom left of Fig. 13(c). When the tools are provisioned, the ontologies are initialized that are subsequently populated as the users perform different activities and operations using the tools (as discussed in Section 5.1.1).

Fig. 14 shows the selected tools used with the prototype implementation of TSPACE. Fig. 14(a) shows semantic relation between the artifacts and corresponding tools with respect to the ontologies described in Section 4. Whereas Fig. 14(b), Fig. 14(c) and Fig. 14(d) show the modified versions of PAKME, ArchDesigner and ArgoUML tools that are provisioned by TSPACE as semantically integrated tools. As our main focus has been to demonstrate the feasibility of TSPACE for software architecting tools, we have selected an architecture knowledge management tool (PAKME), architecture decision support tool (ArchDesigner) and an architecture modeling tool (ArgoUML). The diagrams demonstrate how semantically integrated tools can facilitate users to use PAKME to capture architecture design decisions, use ArchDesigner to evaluate and rank the design decisions and use ArgoUML to make architecture diagrams corresponding to the finalized design decisions. Notifications on actions performed on

[19] http://aws.amazon.com/s3/.
[20] https://docs.jboss.org/jbpm/userguide/wb.Workbench.html.



```
PAKME VM:
Public IP       54.72.70.78 (http://54.72.70.78/pakme/)
User name       Administrator
Password        c2kAcdDCB.

ArchDesigner VM:
Public IP       52.17.170.128 (http://52.17.170.128/arcdesig)
User name       Administrator
Password        C;(eNL9WJMY

Visio VM:
Public IP       51.28.175.120
User name       Administrator
Password        Ajhbn)jkb987M

ArgoUML VM:
Public IP       60.47.160.115
User name       Administrator
```

Fig. 13. TSPACE administration GUIs.

the semantically related artifacts are generated and transmitted with the help of Change and Annotation ontologies (as described in Sections 4.1.3 and 4.1.4).

6.2. *Evaluation for completeness of TSPACE reference architecture*

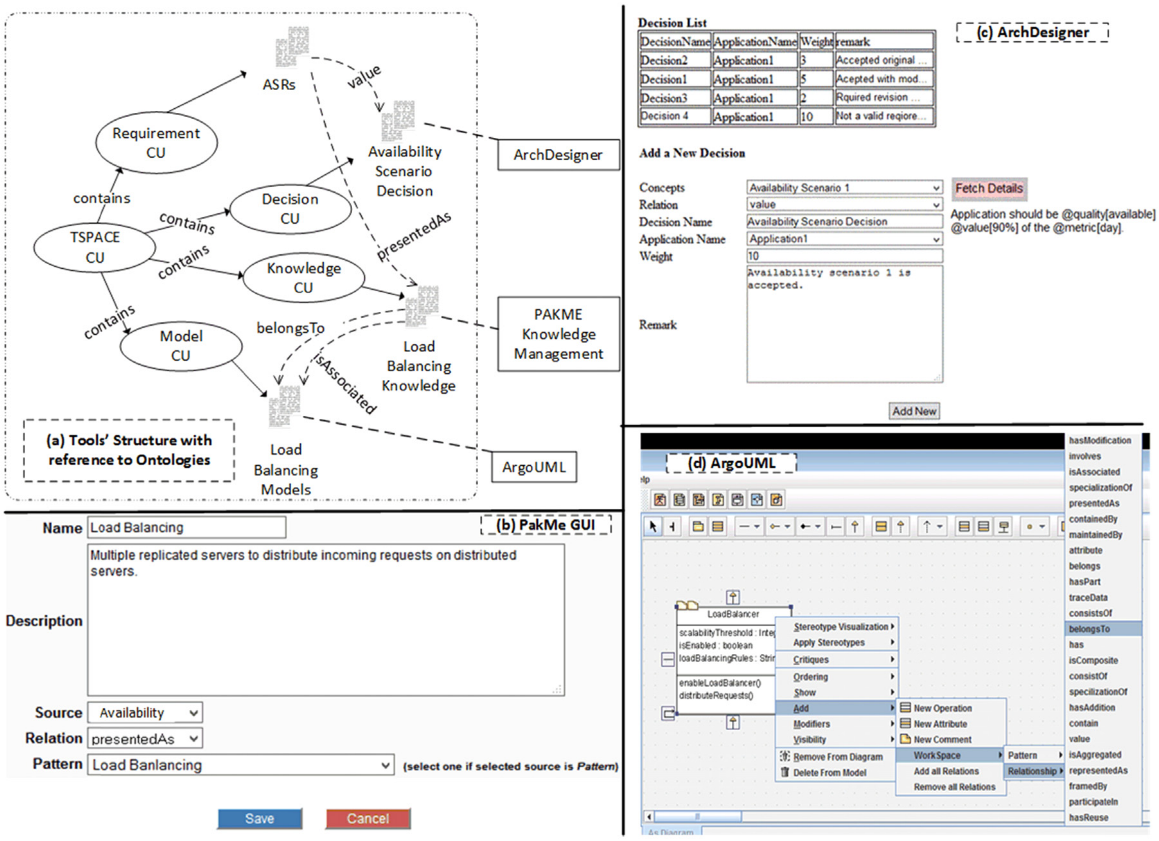
We have evaluated the completeness of TSPACE reference architecture for functional requirements (FR1, FR2, FR3 and FR4) and have used scenarios based evaluation for non-functional requirements (QR1, QR2, QR3, QR4, QR5 and QR6). We report the key reasoning points and the outcomes of the evaluation decisions. Table 5 shows the mapping between the lifecycle phases, functional requirements and corresponding components from decomposed architectural representations. It is clear from Table 5 that different parts of the reference architecture provide support for all the phases and corresponding requirements (Req).

We have presented TSPACE reference architecture in terms of its goals [30] that 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 interaction between the components of the reference architecture. It covers all the important dimensions for reporting a reference architecture as per [9]. It also positively addresses the completeness of the reference architecture (QR4). Our decision of using a layered approach supports separation of concerns among the components and high degree of modifiability (QR5). 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 (QR2) and integration (QR5). Façade pattern is used at the interface layer to provide interoperability (QR3) 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 (QR6) in the ontology-based semantic integration. The adoption of ontology-based approach for tools selection, provisioning, integration, collaboration and awareness enables the reference architecture to be applicable (QR4) to heterogeneous tools and activities. Although security is not explicitly considered, it is addressed with the help of the Authentication component.

6.3. *Evaluation of awareness support*

TSPACE can be used to raise awareness for different types of the operations that are performed on the artifacts by the users (stakeholders) using the provisioned tools. The awareness operations are supported using the ontologies described in Section 4. The awareness support of TSPACE can also be used to identify conflicts. For example, for software architecture knowledge management tools, the different types of knowledge conflicts can occur when the users use different metric or metric measurement values for same quality attribute while capturing non-functional quality requirements.

Fig. 15 shows the requirements capturing GUI of the modified PAKME tool used in the TSPACE prototype. In Fig. 15, two availability requirements and two scalability requirements (non-functional requirements) are presented where respective parts of the requirements are annotated with quality, value and metric annotations. Availability scenarios shown in Fig. 15(a)



Name	Load Balancing
Description	Multiple replicated servers to distribute incoming requests on distributed servers.
Source	Availability
Relation	presentedAs
Pattern	Load Banlancing (select one if selected source is *Pattern*)

Decision List

DecisionName	ApplicationName	Weight	remark
Decision2	Application1	3	Accepted original ...
Decision1	Application1	5	Acepted with mod...
Decision3	Application1	2	Rquired revision ...
Decision 4	Application1	10	Not a valid reqiore...

Add a New Decision

Concepts	Availability Scenario 1
Relation	value
Decision Name	Availability Scenario Decision
Application Name	Application1
Weight	10
Remark	Availability scenario 1 is accepted.

Application should be @quality[available] @value[90%] of the @metric[day].

Fig. 14. Semantic integration structure and prototype tools.

Table 5
Activities, requirements an component mapping.

Life cycle phase	Req. ID	RA components
Tools registration	FR1	Tools/Capability ontology and (Tools') Repository manager.
Tools selection	FR1	Tools/Capability ontology and tools selector.
Enactment and provisioning	FR1, QR1	Tools/Capability ontology, Tool preference manager, Tools enactment engine and IaaS cloud management wrapper.
Semantic and process centric integration	FR2, FR3, QR5	Artifact ontology, Global ontology knowledgebase, Common queue and tenant queues, Filtration rules, Ontology RDFs, Annotation manager, Semantic integration manager and information discovery manager.
Relationship management	FR2	Annotation ontology, Ontology RDFs and semantic integration manager.
Awareness of the operations on the artifacts	FR4	Annotation ontology, Change ontology, Information discovery manager, SPQRQL query executor, Notification manager and notification rules.

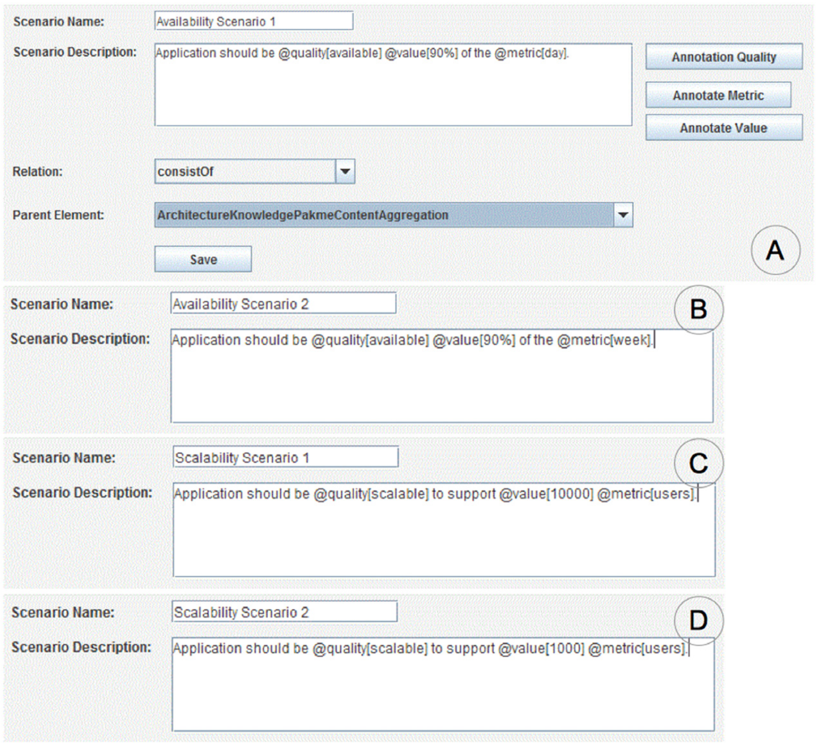
and Fig. 15(b) have different metric units but same values. Scalability scenarios depicted in Fig. 15(c) and Fig. 15(d) have the same metric unit but different values of the metrics. When this information is saved in PAKME, the probes implemented in PAKME send non-functional requirements data and annotations to the TSPACE. When the TSPACE is initialized and tools are launched, different types of notifications can be configured (as shown with check boxes in Fig. 13(c)). 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 Fig. 16. Some sample notification rules are described in Table 3. The conflict notifications that are presented in Fig. 16 are generated by running SPARQL queries and complimentary algorithms, which search for same quality attributes in non-functional requirements but with either different metric units or different metric values.

6.4. Evaluation of TSPACE reference architecture via potential stakeholders

We have used a tailored Architecture Tradeoff Analysis Method (ATAM) [31] to evaluate TSPACE reference architecting by conducting an evaluation session with potential stakeholders. ATAM can be used to analyze software design strategies that are 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 [7]. While analyzing architecture for identifying sensitivity and tradeoff points, architecture scenarios are used.

6.4.1. Evaluation settings

We had 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 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 Ph.D. degree) in Software Engineering/Computer Science. The participants of the evaluation session have detailed knowledge of design and development of cloud-enabled software systems. As TSPACE


Scenario Name: Availability Scenario 1

Scenario Description: Application should be @quality[available] @value[90%] of the @metric[day].

Annotation Quality | Annotate Metric | Annotate Value

Relation: consistOf

Parent Element: ArchitectureKnowledgePakmeContentAggregation

Save

A

Scenario Name: Availability Scenario 2

Scenario Description: Application should be @quality[available] @value[90%] of the @metric[week].

B

Scenario Name: Scalability Scenario 1

Scenario Description: Application should be @quality[scalable] to support @value[10000] @metric[users].

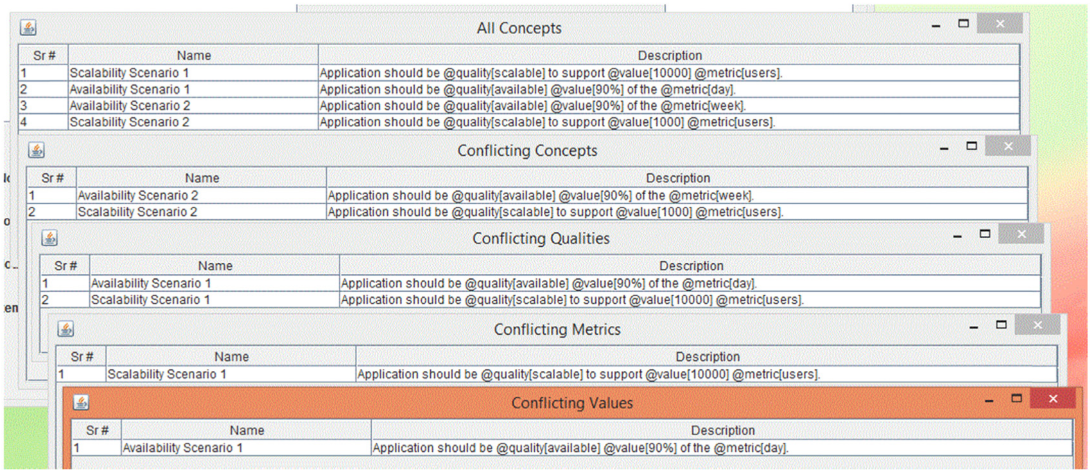
C

Scenario Name: Scalability Scenario 2

Scenario Description: Application should be @quality[scalable] to support @value[1000] @metric[users].

D

Fig. 15. PAKME requirements capturing GUI.


All Concepts

Sr #	Name	Description
1	Scalability Scenario 1	Application should be @quality[scalable] to support @value[10000] @metric[users].
2	Availability Scenario 1	Application should be @quality[available] @value[90%] of the @metric[day].
3	Availability Scenario 2	Application should be @quality[available] @value[90%] of the @metric[week].
4	Scalability Scenario 2	Application should be @quality[scalable] to support @value[1000] @metric[users].

Conflicting Concepts

Sr #	Name	Description
1	Availability Scenario 2	Application should be @quality[available] @value[90%] of the @metric[week].
2	Scalability Scenario 2	Application should be @quality[scalable] to support @value[1000] @metric[users].

Conflicting Qualities

Sr #	Name	Description
1	Availability Scenario 1	Application should be @quality[available] @value[90%] of the @metric[day].
2	Scalability Scenario 1	Application should be @quality[scalable] to support @value[10000] @metric[users].

Conflicting Metrics

Sr #	Name	Description
1	Scalability Scenario 1	Application should be @quality[scalable] to support @value[10000] @metric[users].

Conflicting Values

Sr #	Name	Description
1	Availability Scenario 1	Application should be @quality[available] @value[90%] of the @metric[day].

Fig. 16. Conflict notifications.

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 to analyze all aspects of TSPACE reference architecture.

Expertise of the participants were as follows: two participants (P1 and P2) had expertise in designing and developing workflow based tools and system (both with 11 years of working experience), two participants (P3 and P4) had experience with designing and developing software engineering tools for distributed software development (with 8 and 5 years of experience respectively), and two participants (P5 and P6) had experience with design and development of cloud based and web based applications (with 5 and 2 years of experience respectively). All the participants were given a document describing TSPACE reference architecture requirements, architecture design decisions and different view of the reference architecture a week before the evaluation session.

In the beginning of the evaluation session, an introduction and context of TSPACE reference architecture was presented to the participants. Once 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 that evaluation activity. First 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, which was followed by an exercise of identifying sensitivity and tradeoff points, and building 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.

The participants of the evaluation session were asked to fill in a questionnaire during the session. In the questionnaire, there

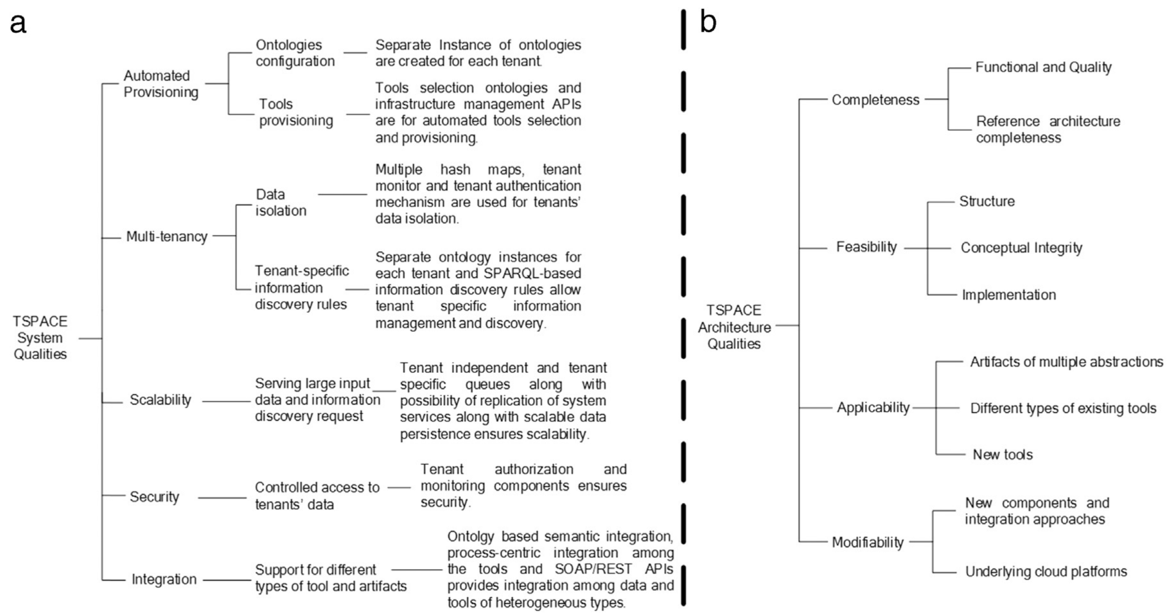

a
TSPACE System Qualities
Automated Provisioning
Ontologies configuration
Separate Instance of ontologies are created for each tenant.
Tools provisioning
Tools selection ontologies and infrastructure management APIs are for automated tools selection and provisioning.
Multi-tenancy
Data isolation
Multiple hash maps, tenant monitor and tenant authentication mechanism are used for tenants' data isolation.
Tenant-specific information discovery rules
Separate ontology instances for each tenant and SPARQL-based information discovery rules allow tenant specific information management and discovery.
Scalability
Serving large input data and information discovery request
Tenant independent and tenant specific queues along with possibility of replication of system services along with scalable data persistence ensures scalability.
Security
Controlled access to tenants' data
Tenant authorization and monitoring components ensures security.
Integration
Support for different types of tool and artifacts
Ontolgy based semantic integration, process-centric integration among the tools and SOAP/REST APIs provides integration among data and tools of heterogeneous types.
b
TSPACE Architecture Qualities
Completeness
Functional and Quality
Reference architecture completeness
Feasibility
Structure
Conceptual Integrity
Implementation
Applicability
Artifacts of multiple abstractions
Different types of existing tools
New tools
Modifiability
New components and integration approaches
Underlying cloud platforms

Fig. 17. Architecture evaluation utility tree.

were two types of questions: (i) the questions (Q1, Q2, Q4, Q6, Q7) those had a qualitative scale to be chosen for their answers and (ii) the questions (Q1-D, Q2-D, Q3-D, Q5-D, Q6-D, Q7-D) those had descriptive answers. Five option value for qualitative scale have used as follows: (a) very low ⇓, (b) low ⇐, (c) medium ⇔, (d) High ⇒ and (e) very high ⇑. Following is the list of questions that were used for the evaluation questionnaire.

- Q1: To what extent (detail) TSPACE functional requirements cover drivers for providing software architecting TaaS?
- Q1-D: What additional requirements do you think should be address by the reference architecture? Please described in sufficient detail?
- Q2: To what extent quality attributes that are considered for the quality of TSPACE as well as quality of the reference architecture are 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 the design decisions that are taken to address the design time and runtime quality of TSPACE reference architecture are 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 the functional requirements and quality characteristics are addressed in TSPACE reference architecture?
- Q6-D: Please provide details on which requirements are not addressed in the reference architecture.
- Q7: To what extent the presented reference architecture addresses challenges associated with TSPACE in general context of TaaS? I.e., For other domains other than software architecting.
- Q7-D: Please provide your comments/details if additional dimensions should be considered in the reference architecture.

Table 6
Quality evaluation results.

Participant ID	Questions			
	Q1	Q2	Q6	Q7
P1	⇒	⇒	⇑	⇒
P2	⇒	⇒	⇑	⇔
P3	⇒	⇒	⇒	⇒
P4	⇒	⇒	⇒	⇔
P5	⇒	⇒	⇒	⇒
P6	⇒	⇑	⇒	⇒

Medium (⇔), High (⇒), Very high (⇑).

6.4.2. Evaluation results

Q1 and Q2 aimed to seek input on functional and quality completeness of TSPACE reference architecture. Q6 aimed to identify to which extent the solutions that have been proposed in TSPACE reference architecture address the stated requirements and quality characteristics. Q7 aimed to identify 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 6. The results show that on average a high value score (corresponding to the questions) was assigned by the participants.

Q4 aimed to identify effectiveness of the important design decisions that had been made during design of TSPACE RA. During the evaluation session, key design strategies (that have been discussed in Section 5) 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 7. The results show that on average all the design decisions were ranked high.

Questions Q1-D and Q2-D were aimed to identify if 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. The RA has been decomposed at three-level to provide more details on the important components of TSPACE reference architecture. There was no concern raised for Q6-D.

Table 7
TSPACE RA design decision ranking.

Design decisions	Participant's score for effectiveness of the design decisions					
	P1	P2	P3	P4	P5	P6
TSPACE ontology meta-model	⇒	⇑	⇑	⇑	⇑	⇔
Using ontologies	⇑	⇒	⇒	⇒	⇒	⇒
Using SOA (SOAP and REST)	⇒	⇒	⇑	⇒	⇑	⇑
Shared repository templates (shared repository pattern)	⇑	⇑	⇑	⇒	⇒	⇑
Tenant neutral queues and tenant specific queues and filters	⇒	⇑	⇒	⇒	⇑	⇑
Layered architecture style	⇒	⇑	⇒	⇑	⇑	⇑

Medium (⇔), High (⇒), Very high (⇑).

Q5-D aimed at identifying the risks, sensitivity points and tradeoff points [7] in TSPACE RA. Utility trees of ATAM [31] were used to facilitate discussion on risks, sensitivity points and tradeoff points. The identified risks and tradeoff points were mainly related with performance of TSPACE. A risk was identified during the evaluation session concerning comprehensiveness of the ontologies to capture artifacts and different types of tools (other than software architecting domain) in TSPACE. This risk is mitigated in the reference architecture by providing flexibility to extend the ontologies and having possibility to add new ontology templates if needed. While discussing sensitivity and tradeoff points, it had been figured out that *Queues* and *Shared Ontology Repositories* (described in Section 5.1) 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). *Queues* and *Shared Ontology Repositories* were also identified as tradeoff points (tradeoff between unified access point and scalability). Tenant independent and tenant specific queues along with respective façade (described in Section 5.1.2) are proposed in TSPACE RA to provide a single point of access and shared ontology repositories are prosed to provide a common ontology knowledgebase.

During the evaluation session, two utility trees were generated. Fig. 17(a) shows system qualities of TSPACE reference architecture, whereas Fig. 17(b) shows the utility tree for architecture design qualities of TSPACE RA (i.e. completeness, feasibility, applicability and modifiability). Feasibility of the reference architecture is evaluated in terms of structure and conceptual integrity as well as by implementing its prototype. Applicability of the reference architecture is analyzed by demonstrating provisioning of different types of tools in TSPACE and by supporting integration among the artifacts at different levels of abstractions. Moreover, TSPACE applicability for different types of tools was also analyzed during the evaluation session. Layered and components based architecture enables addition of new components, enhancements in integration approach and incorporation of different IaaS cloud in TSPACE. TSPACE system qualities shown in the utility tree of Fig. 17(a) 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 are shown in Fig. 17(a).

7. Related work

A number of studies have reported adoption of ontology-based approaches to address software engineering challenges, in particular for software process, knowledge management, software design documentation and software design traces [14]. A handful of studies have also reported the architecture of services and tools in an integrated environment. In this section, we provide an overview of the related work with reference to the following key attributes: problems that are addressed in the reported research, approach to provide solutions to the problems and constraints (if any) of the presented solutions.

7.1. *Ontologies to support process workflows and integration*

Process ontologies focus on capturing the process and identifying relationships between elements of the process [14]. Boškovic et al. [32] have presented an ontology for configurable business processes by identifying the relationship between features of independent software families, and verifying and validating the relationships between business process customers and developers. Feature modeling is performed using semantic annotations but it does not provide integration among the services with respect to business logic and operations. Valiente et al. [33] have proposed an ontology-based approach to integrate software development and information technology service management processes and corresponding support tools. However, the presented approach does not deal with issues specific to tools integration and information consistency, especially when multiple tools generate artifacts that cannot be transmitted among tools and services (opposite to how the artifacts are handled in a workflow-based system).

Artifact ontologies attempt to capture relations between views and corresponding artifacts. Athanasiadis et al. [34] have proposed a technique for object to relational mapping based on semantic web. Ameller and Franch [35] have presented an ontology to describe relationship between architecture views, frameworks, architecture styles, variants of architecture styles and their implementation in context of a web-based application. Antunes et al. [36] 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 specific user is performing. Happel et al. [37] have presented a software reuse methodology based on ontologies to facilitate software libraries reuse by providing background knowledge, and semantic integration of implicit and explicit metadata to derive new facts from the existing knowledge.

Software documentation ontologies focus on formalizing software documentation semantics. Witte et al. [38] 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 for easy traceability recovery between source code and software documentation. Zhang et al. [39] have proposed a traceability recovery approach based on ontologies to establish a relationship between source code and corresponding source code documents at semantic level. The presented approach is limited to simple relationships between design documents describing its different elements and corresponding source code.

A number of studies have also reported use of ontologies for software architecture documentation. Boer et al. [40] have presented ontologies reuse approach named “QuOnt” [41] to visualize architecture design decisions. The presented approach establishes a relationship between quality criterion and corresponding

Table 8
Criteria for comparing work related to ontologies.

Study reference	Focus	Relations type and granularity	Methods or approaches
Boskovic et al. [32]	Configurable business processes	Business process customers and developers	Feature modeling and semantic annotation
Valiente et al. [33]	Management processes	Services and tools	Integration ontologies
Athanasiadis et al. [34]	Artifacts management	Object to relational mapping	Semantic web
Ameller and Franch [35]	Artifacts management	Architecture views, architecture styles and implementation	Relations ontologies
Antunes et al. [36]	Knowledge repositories	Knowledge repositories	Semantic web
Happel et al. [37]	Software reuse	Knowledge artifacts	Semantic integration
Witte et al. [38]	Code analysis and text mining	Source code and source code documentation	Semantic web and traceability
Zhang et al. [39]	Traceability recovery	Software design documents and source code	Semantic integration
Boer et al. [40]	Architecture design decision visualization	Quality criterion and corresponding quality attributes	Ontologies reuse
Tang et al. [42]	Architecture documentation	Requirements, architecture design decisions and components	Relations ontologies
Graaf et al. [43,44]	Architecture documentation	Concepts' structures and relations	Semantic Wiki
Zhou et al. [45]	Systems reengineering	Software components	Reverse engineering and model transformation
Brandt et al. [46] and Rilling et al. [47]	Knowledge management	Descriptive documents and source code	Ontology schemas

quality attributes. Tang et al. [42] 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. [43,44] have presented ontology for software documentation using a semantic Wiki named ArchiMind. The studies on architecture documentation show the significance of using ontologies to structure and relate concepts involved in software architecture documentation activities. The studies on software architecture documentation do not address the root cause of the integration challenge, i.e., how to provide a common integration model for multiple types of artifacts maintained by heterogeneous tools that are used to perform activities associated with software architecting.

Zhou et al. [45] have presented an approach for reengineering software for cloud-based systems using ontologies. Ontologies for enterprise applications are built using reverse engineering and model transformation techniques. Brandt et al. [46] and Rilling et al. [47] have presented a flexible ontology-based schemas for knowledge management, and a meta-model and ontology to link documents with source code. The presented approaches do not provide details of the ontologies 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 the knowledge documents with source code.

We have identified a set of attributes to compare the related work on ontologies for process workflows and integration. The identified attributes include *Focus* of the research approach, *Type and Granularity of the Relations* among the artifacts, and *Method or Approach* that have been proposed in a particular study. *Focus* of the research approach attribute describes key focus of the proposed ontologies for process workflows and integration, e.g., artifacts management. *Type and Granularity* of the relations attribute describes the types of artifacts that are considered by the ontologies relations, e.g., source code documentation and source code. *Method or Approach* attribute describes research approach used to define the ontologies, e.g., feature modeling and semantic annotation. The related work described in this subsection is elaborated with respect to the identified attributes in Table 8.

7.2. Frameworks and architectures for services provisioning and integration

A number of frameworks and architectures for provisioning integrated systems have been proposed. Rezaei et al. [48] have presented a framework to provide semantic interoperability among SaaS on clouds. The authors have argued using a unified interoperability interface and service semantic description editor that can be used to define semantic integration rules. However, manually defining semantic integration rules for every single service in SaaS stack limits applicability of this approach in commercial solutions. Xu et al. [49] have presented a framework to bridge the gap between low-level features and high-level semantics of the video contents on cloud by building the ontology model for basic concepts, events and relations. A semantic intermediate layer is proposed to organize the video data based on their semantics. Though the presented framework is promising, its limitation to work only with video data makes it less useful for heterogeneous artifacts.

Barbosa et al. [50] have presented software-testing ontology for the development of software testing tool. Oliveira and Nakagawa [51] have proposed a Service Oriented Architecture (SOA) for software testing tools. Their work provides the detail on architectural requirements and a layered model to map tools onto the business process. As their work focuses on SOA, it does not cover a complete lifecycle of tools provisioning and operations for cloud-based software systems. Nakagawa et al. [52] have also presented an aspect oriented reference architecture for software engineering environments. The presented architecture do not explicitly focus on integration among the tools and the authors encourage the use of workflows to facilitate information and artifacts exchange. Chapman et al. [53] have presented a software architecture for on-demand cloud provisioning. The presented architecture manages service repository and deploys the services on underlying cloud infrastructure with the help of virtual execution environment manager. Lin et al. [54] have presented a reference architecture for workflow management of scientific applications following SOA principles. The presented reference architecture focuses on workflow and provenance management, however it does not provide any insight to the special needs of the workflow-based tools composition. Moser and Biffl [55] have proposed a semantic integration approach for engineering environments. They have proposed usage of engineering knowledgebase and a common virtual data-model to provide semantic integration between different types of engineering models. Tajalli and Medvidovic [56] have presented a reference architecture for integrated software environments named iDARE, which has an adaptation engine at its core that can transform artifacts to support integration among the applications.

Table 9
Criteria for comparing work related to frameworks and architecture.

Study reference	Focus	Methods or approaches	General applicability
Rezaei et al. [48]	Semantic interoperability	Unified interoperability interfaces and services semantic description	✓
Xu et al. [49]	Features and semantics alignments	Semantic integration layer and ontology model for concepts events and relations	×
Barbosa et al. [50]	Software testing	Testing ontologies	✓
Oliveira and Nakagawa [51]	Software testing	SOA and business process to tools modeling	×
Nakagawa et al. [52]	Software engineering	Aspect oriented architecture and workflows	✓
Chapman et al. [53]	On-demand provisioning	Service repositories and virtual execution manager	✓
Lin et al. [54]	Scientific workflows management	SOA	×
Moser and Biffl [55]	Semantic integration	Engineering knowledgebase and common virtual data-model	✓
Tajalli and Medvidovic [56]	Integrated software environments	Artifacts transformation and adaptation engine	✓
Calvo et al. [57]	Textual information retrieval	Tools for business process modeling	×

Calvo et al. propose an architecture for textual information retrieval from cloud-based collaborative writing tools [57] but their effort is only limited to support automated feedback and process analysis of students' academic assignment write-ups. 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 [58,59]. An extensible architecture description language (xADL) to support integration among architecture centric tools is presented in [60]. 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[21] and a diagramming tool Griffy.[22]

We have also identified a set of attributes to compare the related work on frameworks and architectures for service provisioning and integration. The identified attributes include focus of the research, method and approach that have been used to achieve the research objectives and applicability of the proposed solution in a broader context. *Focus* attribute describes a specific dimension of service provisioning and integration that is focused in a study, e.g., semantic interoperability. *Method or Approach* attribute describes architecture design tactics, decisions and methods that are used to achieve the research objectives, e.g., interoperability interfaces and service semantic description. *General Applicability* attribute indicates if the solution proposed in a particular study can generally be applied for provisioning and integration of cloud-based systems. If a solution proposed in the study is generally applicable then a check mark (✓) is shown corresponding to the study. If the solution is not generally applicable, then a cross mark (×) is shown. The related work described in this subsection is elaborated with respect to the identified attributes in Table 9.

7.3. TSPACE RA with respect to related work

In comparison to the discussed related work, 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. Contrary to the existing ontologies research discussed in the related work section, the TSPACE ontologies focus on providing integration among artifacts at multiple abstractions. The TSPACE ontologies also provide foundation for tools selection, provisioning and awareness of the operations that are performed on the artifacts using the tools. The TSPACE RA uses common metamodel as a control point for design and enhancement of the reference architecture. The proposed TSPACE reference architecture also supports process-centric integration among the tools and semantic integration (using the TSPACE ontologies) among the artifacts that are consumed or produced during different activities that are performed using the tools.

8. Conclusions

We have presented and discussed Tools as a service SPACE (TSPACE) Reference Architecture (RA) in terms of: TSPACE meta-model that identifies the RA elements, an ontology-based semantic integration framework that provides the backbone for semantic integration model of the proposed RA and relations among the elements, and views of the RA at two levels of abstractions. Keeping TSPACE RA meta-model at the core of our adopted RA design strategy provides a centralized point for the identification of the TSPACE RA elements and can facilitate the RA adoption in different domains. The identified ontologies provide a structured approach for not only tools selection, but also for semantically relating and annotating the artifacts produced by the provisioned tools and for providing support for awareness of the operations that are performed on the artifacts using the tools. The presented reference architecture introduces a standardized view of a TSPACE and has the potential of providing a number of benefits to practitioners and researchers. The reference architecture can provide an increased understanding of TSPACE for the 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. Practitioners can use the reference architecture to communicate a TSPACE's requirements and the main architectural principles in software engineering teams. 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 extended by considering security as an explicit non-functional requirement to provide more comprehensive security support in TSPACE reference architecture.

[21] https://c9.io/.
[22] https://www.gliffy.com/.

Acknowledgments

We acknowledge Zhi Wang and Hao Zhang for their contribution for implementing graphical user interface for process centric tools provisioning presented in Fig. 13(a).

References

[1] M.A. Babar, I. Gorton, A tool for managing software architecture knowledge, in: presented at the Proceedings of the Second Workshop on SHAring and Reusing Architectural Knowledge Architecture, Rationale, and Design Intent, 2007.
[2] B. Decker, E. Ras, J. Rech, B. Klein, 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.
[3] P. Kruchten, The Rational Unified Process: An Introduction, Addison-Wesley Professional, 2004.
[4] N. Harrison, P. Avgeriou, U. Zdun, Using patterns to capture architectural decisions, IEEE Softw. 24 (2007) 38–45.
[5] M.A. Chauhan, M.A. Babar, Cloud infrastructure for providing tools as a service: quality attributes and potential solutions, in: presented at the Proceedings of the WICSA/ECSA 2012 Companion Volume, Helsinki, Finland, 2012.
[6] M.A. Chauhan, M.A. Babar, A Systematic Mapping Study of Software Architectures for Cloud Based Systems, Technical Report TR-2014-175, 2014.
[7] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, Addison-Wesley Professional, 2012.
[8] S. Angelov, P. Grefen, An e-contracting reference architecture, J. Syst. Softw. 81 (2008) 1816–1844.
[9] S. Angelov, P. Grefen, D. Greefhorst, A framework for analysis and design of software reference architectures, Inf. Softw. Technol. 54 (2012) 417–431.
[10] P. Avgeriou, Describing, instantiating and evaluating a reference architecture: A case study, Enterp. Archit. J. (2003) 24.
[11] IEEE recommended practice for architectural description of software-intensive systems, in: IEEE Std 1471-2000, 2000, p. i-23.
[12] 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), 2011 pp. 1–46.
[13] F. Arvidsson, A. Flycht-Eriksson, Ontology I, http://www.ida.liu.se/janma/SemWeb/Slides/ontologies1.pdf, (Retrieved 23.06.14).
[14] Y. Zhao, J. Dong, T. Peng, Ontology classification for semantic-web-based software engineering, IEEE Trans. Serv. Comput. 2 (2009) 303–317.
[15] 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 Semant.: Sci. Serv. Agents World Wide Web 4 (2006) 14–28. 1.
[16] J.R. Hilera, x Ferna, L. ndez-Sanz, Developing domain-ontologies to improve sofware engineering knowledge, in: 2010 Fifth International Conference on Software Engineering Advances, ICSEA, 2010, pp. 380–383.
[17] S. Nešić, Semantic document model to enhance data and knowledge interoperability, in: V. Devedžić, D. Gaševic (Eds.), Web 2.0 & Semantic Web, Vol. 6, Springer, US, 2009, pp. 135–160.
[18] A. Segev, Q.Z. Sheng, Bootstrapping ontologies for web services, IEEE Trans. Serv. Comput. 5 (2012) 33–44.
[19] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, Pattern-Oriented Software Architecture: A System of Patterns, John Wiley & Sons, Inc., 1996.
[20] 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, Robot. Comput.-Integr. Manuf. 24 (2008) 585–593. 8.
[21] D. Baxter, J. Gao, K. Case, J. Harding, B. Young, S. Cochrane, et al., An engineering design knowledge reuse methodology using process modelling, Res. Eng. Des. 18 (2007) 37–48. 2007/05/01.
[22] O. Zimmermann, C. Miksovic, J.M. Küster, Reference architecture, metamodel, and modeling principles for architectural knowledge management in information technology services, J. Syst. Softw. 85 (2012) 2014–2033. 9.
[23] F. Zahedi, The analytic hierarchy process-a survey of the method and its applications, Interfaces 16 (1986) 96–108.
[24] P.B. Kruchten, The 4+1 View Model of architecture, IEEE Softw. 12 (1995) 42–50.
[25] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Pearson Education, 1994.
[26] B. Leiba, Oauth web authorization protocol, IEEE Internet Comput. (2012) 74–77.
[27] A.J. Riel, Object-Oriented Design Heuristics, Vol. 338, Addison-Wesley, Reading, 1996.
[28] J.B. Bernabe, J.M.M. Perez, J.M.A. Calero, F.J.G. Clemente, G.M. Perez, A.F.G. Skarmeta, Semantic-aware multi-tenancy authorization system for cloud architectures, Future Gener. Comput. Syst. 32 (2014) 154–167.
[29] 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), IEEE, 2010, pp. 458–465.
[30] P. Clements, L. Bass, Relating Business Goals to Architecturally Significant Requirements for Software Systems, DTIC Document, 2010.
[31] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, J. Carriere, The architecture tradeoff analysis method, in: Fourth IEEE International Conference on Engineering of Complex Computer Systems, 1998. ICECCS'98. Proceedings., 1998, pp. 68–78.
[32] M. Boškovic, E. Bagheri, G. Grossmann, D. Gaševic, M. Stumptner, Towards integration of semantically enabled service families in the cloud, in: WS2, 2011, p. 58.
[33] M.-C. Valiente, E. Garcia-Barriocanal, M.-A. Sicilia, Applying ontology-based models for supporting integrated software development and it service management processes, IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. 42 (2012) 61–74.
[34] I.N. Athanasiadis, F. Villa, 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.
[35] D. Ameller, X. Franch, Ontology-based architectural knowledge representation: Structural elements module, in: C. Salinesi, O. Pastor (Eds.), Advanced Information Systems Engineering Workshops, Vol. 83, Springer Berlin, Heidelberg, 2011, pp. 296–301.
[36] B. Antunes, P. Gomes, N. Seco, SRS: a software reuse system based on the semantic web, in: 3rd International Workshop on Semantic Web Enabled Software Engineering, SWESE, 2007.
[37] H.-J. Happel, A. Korthaus, S. Seedorf, P. Tomczyk, KOntoR: an ontology-enabled approach to software reuse, in: Proc. of The 18Th Int. Conf. On Software Engineering and Knowledge Engineering, 2006.
[38] R. Witte, Y. Zhang, J. Rilling, Empowering software maintainers with semantic web technologies, in: The Semantic Web: Research and Applications, Springer, 2007, pp. 37–52.
[39] Y. Zhang, R. Witte, J. Rilling, 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.
[40] R.C. de Boer, P. Lago, A. Telea, H. Van Vliet, Ontology-driven visualization of architectural design decisions, in: Joint Working IEEE/IFIP Conference on Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009., 2009, pp. 51–60.
[41] R.C. de Boer, H. Van Vliet, QuOnt: an ontology for the reuse of quality criteria, in: ICSE Workshop on Sharing and Reusing Architectural Knowledge, 2009. SHARK'09., 2009, pp. 57–64.
[42] A. Tang, L. Peng, H. van Vliet, Software architecture documentation: The Road Ahead, in: 2011 9th Working IEEE/IFIP Conference on Software Architecture, WICSA, 2011, pp. 252–255.
[43] K.A. de Graaf, P. Liang, A. Tang, W.R. van Hage, H. van Vliet, An exploratory study on ontology engineering for software architecture documentation, Comput. Ind. 65 (7) (2014) 1053–1064.
[44] K.A. de Graaf, A. Tang, L. Peng, H. Van Vliet, Ontology-based software architecture documentation, in: 2012 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture, ECSA, 2012, pp. 121–130.
[45] H. Zhou, H. Yang, 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.
[46] S.C. Brandt, J. Morbach, M. Miatidis, M. Theißen, M. Jarke, W. Marquardt, An ontology-based approach to knowledge management in design processes, Comput. Chem. Eng. 32 (2008) 320–342.
[47] J. Rilling, Y. Zhang, W.J. Meng, R. Witte, V. Haarslev, P. Charland, A unified ontology-based process model for software maintenance and comprehension, in: Models in Software Engineering, Springer, 2007, pp. 56–65.
[48] R. Rezaei, T.K. Chiew, S.P. Lee, Z. Shams Aliee, A semantic interoperability framework for software as a service systems in cloud computing environments, Expert Syst. Appl. 41 (2014) 5751–5770. 10/1/.
[49] Z. Xu, L. Mei, Y. Liu, C. Hu, L. Chen, Semantic enhanced cloud environment for surveillance data management using video structural description, Computing (2014) 1–20. 2014/05/16/.
[50] E.F. Barbosa, E.Y. Nakagawa, A.C. Riekstin, J.C. Maldonado, Ontology-based development of testing related tools, in: SEKE, 2008, pp. 697–702.
[51] L.B.R. Oliveira, E.Y. Nakagawa, A service-oriented reference architecture for software testing tools, in: Software Architecture, Springer, 2011, pp. 405–421.
[52] E.Y. Nakagawa, F.C. Ferrari, M.M. Sasaki, J.C. Maldonado, An aspect-oriented reference architecture for software engineering environments, J. Syst. Softw. 84 (2011) 1670–1684.
[53] C. Chapman, W. Emmerich, F. Márquez, S. Clayman, A. Galis, Software architecture definition for on-demand cloud provisioning, Cluster Comput. (2011) 1–22.
[54] C. Lin, S. Lu, X. Fei, A. Chebotko, D. Pai, Z. Lai, et al., A reference architecture for scientific workflow management systems and the VIEW SOA solution, IEEE Trans. Serv. Comput. 2 (2009) 79–92.
[55] T. Moser, S. Biffl, Semantic integration of software and systems engineering environments, IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. 42 (2012) 38–50.
[56] H. Tajalli, N. Medvidović, iDARE—a reference architecture for integrated software environments, Softw. - Pract. Exp. 44 (2014) 299–316.
[57] R.A. Calvo, S.T. O'Rourke, J. Jones, K. Yacef, P. Reimann, Collaborative writing support tools on the cloud, IEEE Trans. Learn. Technol. 4 (2011) 88–97.

[58] R. Wolvers, T. Seceleanu, embedded systems design flows: integrating requirements authoring and design tools, in: 2013 39th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA, 2013, pp. 244–251.

[59] M. Biehl, J. De Sosa, M. Torngren, 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.

[60] R. Khare, M. Guntersdorfer, P. Oreizy, N. Medvidovic, R.N. Taylor, xADL: enabling architecture-centric tool integration with XML, in: Proceedings of the 34th Annual Hawaii International Conference on System Sciences, 2001. 2001, p. 9.

Muhammad Aufeef Chauhan received the Ph.D. degree in Computer Science from IT University of Copenhagen, Denmark. He is currently working as a Postdoctoral researcher at IT University of Copenhagen (ITU). He holds master degrees in Software Engineering from IT University of Copenhagen, Denmark and Mälardalen University, Sweden; and holds bachelor degree in Computer Science from National University of Computer and Emerging Sciences (FAST-NU), Pakistan. His research interests include software architectures for cloud-enabled and distributed software systems, migration of applications to Software as a Service (SaaS) model, software evolution and Global Software Development (GSD).

Muhammad Ali Babar is a Professor in the School of Computer Science, University of Adelaide. He is an honorary visiting professor at the Software Institute, Nanjing University, China. He also holds an academic position at the IT University of Copenhagen, Denmark. He has established an interdisciplinary research centre called CREST, Centre for Research on Engineering Software Technologies, where he directs the research and education activities in the areas of software systems engineering, security and privacy, and social computing. He obtained a Ph.D. in Computer Science and Engineering from the school of computer science and engineering of University of New South Wales. Further details can be found at http://malibabar.wordpress.com.

Quan Z. Sheng received the Ph.D. degree in Computer Science from the University of New South Wales, Sydney, Australia. He is a professor in the School of Computer Science at the University of Adelaide. His research interests include service-oriented architectures, web of things, distributed computing, and pervasive computing. He was the recipient of the 2011 Chris Wallace Award for Outstanding Research Contribution and the 2003 Microsoft Research Fellowship. He is the author of more than 240 publications. He is a member of the IEEE and ACM.