A Reference Architecture for a Cloud-Based Tools as a Service Workspace

Muhammad Aufeef Chauhan[±1], Muhammad Ali Babar[±1,2], Quan Z. Sheng[2]
[±]CREST-Centre for Research on Engineering Software Technologies[1,2]
[1]IT University of Copenhagen, Denmark
[2]The University of Adelaide, Australia
muac@itu.dk, {ali.babar, michael.sheng}@adelaide.edu.au

***Abstract*—Software Architecture (SA) plays a critical role in developing and evolving cloud-based applications. We present a Reference Architecture (RA) for designing Cloud-based Tools as a service workSPACE (TSPACE) - a platform for provisioning chain of tools following the Software as a Service (SaaS) model. The TSPACE RA has been designed by leveraging well-known design principles and patterns and has been documented using a view-based approach. The RA has been presented in terms of its context, goals and design elements by describing the requirements, design tactics, and components of the RA. We evaluate the RA in terms of completeness and feasibility. Our proposed RA can provide valuable guidance and insights for designing and implementing concrete software architectures of TSPACE.**

Keywords— Cloud Computing, Tools as a Service (TaaS), Software Architecture, Reference Architecture (RA), Ontologies.

I. INTRODUCTION

Software Engineering (SE) in general and software architecting in particular need to be supported by several tools to perform different activities such as software architecture (SA) significant requirements analysis, SA design and evaluation. Traditionally the tools (i.e., individual tools or integrated environments) are offered as desktop or web-based solutions that require frequent configurations, installations and infrastructure management of the tools. With the increasing adoption of Cloud Computing as a flexible and reliable technology for Information Communication Technology (ICT) infrastructure management [1], several commercial and research efforts are focused on provisioning of Cloud-based Tools as a Service (TaaS) (e.g., cloud-based IDE Cloud9[1], online diagramming tool Gliffy[2] and several other efforts reported in [2, 3]).

Whilst Cloud Computing provides a viable and flexible technological infrastructure to provision individual applications as Platform as a Service (PaaS) and Software as a Service (SaaS) models using underlying Infrastructure as a Service (IaaS) cloud [1], TaaS presents several unique challenges [3]. Some of the key challenges include bundling and provisioning a set of tools together in a tool chain as often more than one tool is required to perform the tasks, providing access to the artifacts and data that are managed in different tools, and providing support for the activities that are carried out by the users of different tools [3, 4]. To address these challenges, there is a need of having a cloud-based TSPACE that can bundle and provision a set of tools as part of a tool chain (tool suite). A TSPACE is *an aggregated platform that facilitates activity or task specific tools selection and provisioning, provides integration among heterogeneous types of the artifacts managed by the tools in a TSPACE and raises awareness of the stakeholders' activities (that are performed using the provisioned tools)*. A TSPACE instance is instantiation of TSPACE with a selected set of the tools for a specific project.

Our research effort has been motivated by the need of providing the key specifications and architectural guidelines in terms of a Reference Architecture (RA) based on the Service Oriented Architecture (SOA) principles [5] for designing a TSPACE. A software RA *maps division of functionality together with data flow between the pieces onto software elements and data flow between the elements* [6]. A RA provides a standardization and an abstraction of a concrete software architecture (SA) for a specific domain, facilitates the reuse of design knowledge and reduces the cost of creating new design solutions for respective domains [6]. We foresee that the proposed TSPACE RA will make it easier to design new cloud-based workspaces and will facilitate their software development process. The description of the TSPACE RA details the functionalities to be supported, architecture design decisions made, and different abstractions of the RA. In this paper, we focus on RA for SA domain, although the architectural concepts and design decisions presented in this paper are generic enough to be applied to other domains. Whilst in our previous work we have discussed implications of TaaS in broader context [3, 4], the main contributions of this paper are:

- We present an ontology-model to characterize TSPACE and to design concrete architecture for providing TSPACE. We also briefly discuss how the presented ontology-model can be transformed into a set of concrete ontologies that formalize the tools selection, tools provisioning and semantic integration among the artifacts in TSPACE.
- We provide a detailed description of the TSPACE RA in terms of requirements, development view and logical view as recommended by view based approaches [7]. We

[1] https://c9.io/
[2] http://www.gliffy.com

also provide a brief overview of the solutions that are used to implement the RA prototype.

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

In this paper, we are focusing at the architecture level description of the RA without going into low level algorithmic details. The organization of the paper is as follows. Section II describes our approach for designing and reporting the RA. Section III provides the background and the requirements of the RA. Section IV elaborates the key architecture design principles. Section V describes the RA modules and components at different levels of abstraction. Section VI reports the evaluation, Section VII describes related work and Section VIII concludes the paper.

II. Describing the Reference Architecture

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 way. We describe the proposed RA using a systematic approach that advocates the use of context, goal and design dimensions of a RA documentation as described in [6].Table I lists our TSPACE solution approach corresponding to the different dimensions.

TABLE I. TSPACE Reference Architecture's Dimensions

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 and logical views.
Design	What is described?	The RA is described in terms of high-level modules, connectors, details of the modules in terms of components using logical view 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.
Instantiation	How is it instantiated?	We have instantiated the RA by implementing its prototype.
Evaluation	How is it evaluated?	We have evaluated the RA using scenarios for functional requirements and quality parameters; and assessed its feasibility with a prototype.

The context dimension covers the purpose, the development organization, and maturity stage (e.g., preliminary or classic) of a RA [6]. 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; whether the RA has been described using formal, semiformal or informal approaches. The RA encompasses tools selection, tools provisioning, semantic integration among the artifacts and awareness of the users activities during TSPACE lifecycle. We have extracted and synthesized tools selection and provisioning part of the RA based on a systematic review of cloud provisioning architectures. The complete list of referred architectures can be found in our technical report [8]. The RA part for semantic integration and awareness is a new proposition by us.

III. Requirements

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. TSPACE requirements have evolved based on our previous work on considerations for TaaS infrastructure [3] and a review of the literature on important quality characteristics of cloud-based systems [8]. We have identified the functional requirements 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. The quality (i.e., non-functional) requirements are classified into two categories: i) quality requirements for cloud based solutions (QR1, QR2 and QR3) and ii) quality requirements of the RA (QR4, QR5, QR6) [9]. Table II lists TSPACE requirements.

IV. Architecture Design Strategies and Elements

We have designed the TSPACE RA for tools that are used to support software architecting activities such as architecture requirements description, architecture modeling and architecture evaluation. We have developed the RA experimentally and iteratively. For designing the RA, we followed the part-whole (functional decomposition) principle and several architecture styles [9]. Functional decomposition and part-whole principles help achieve a number of quality characteristics such as modifiability and integratability. Functional decomposition also makes it easy for practitioners and researchers to understand different components of the RA and to tailor it for their specific needs. We have used an ontology-based semantic integration approach to support flexibility (QR4) and interoperability (QR5). Ontology-based semantic integration enables the RA to accommodate different types of artifacts produced or consumed by different tools using standardized or proprietary formats. In the following subsections, we describe the design strategies to achieve the requirements that are listed in Table II.

TABLE II. TSPACE REQUIREMENTS

		ID	Requirement
Functional Requirements		FR1	*Provisioning:* The RA should support provisioning of a TSPACE and associated tools according to the requirements of different activities to be carried out using the tools and constraints on tools enactment location.
		FR2	*Semantic Integration:* The RA should support semantic integration among the artifacts of different types as a TSPACE instance consists of multiple tools that can use different formats to store the artifacts.
		FR3	*Awareness Support:* Multiple artifacts are produced or consumed during the lifecycle of a specific project for which a TSPACE is instantiated. Hence, the RA should provide support for awareness of the users' activities.
Quality (Non-functional) Requirements	***TSPACE Quality***	QR1	*Automated Provisioning:* A RA shall support automated provisioning of a TSPACE and encompassing tools.
		QR2	*Multi-tenancy:* Being a Cloud-based infrastructure, the TSPACE RA needs to be a multi-tenant [10] platform. Each TSPACE instance shall have its own set of tools and rules for awareness. A particular tenant shall be able to access all its specified features and configurations.
		QR3	*Scalability:* The RA shall scale as the number of activities that are performed using the tools increase.
	Reference Architecture Quality	QR4	*Flexibility:* As the tools in a specific instance of a TSPACE depend upon the activities to be performed within a project, the RA shall be flexible enough to provide semantic integration and awareness support for different types of tools.
		QR5	*Interoperability:* A RA shall provide semantic integration and awareness support for different types of artifacts (e.g. textual documentation and UML models).
		QR6	*Completeness and Applicability:* Completeness of the RA is important so that it can serve as a guiding model for designing a specific instance of a TSPACE. The applicability quality characteristic is important so that the RA can be used to design and evaluate a concrete architecture.

A. *Use of Ontologies to Formalize TSPACE*

We have used the ontologies to formalize a TSPACE because ontologies provide shared conceptualization and vocabulary that can be used to model a specific domain [12]. There is an ontology-model at the core of the proposed RA that characterizes the elements of a TSPACE and establishes the relations among the elements as shown in Figure 1. As we intended to concretize the TSPACE RA for the software architecture domain, the proposed ontology-model is based on ISO/IEC/IEEE 42010:2011 conceptual meta-model of architecture description [11]. The ontology-model shows the abstract elements of the TSPACE RA. A project's stakeholders usually work with multiple tools provided by commercial vendors or Open Source community. These tools need architectural level support for interoperability so that the artifacts produced in different formats (texts, diagrams, standardized formats and proprietary formats) can be integrated with each other. *Tool* element (at top of the figure) in ontology-model represents the tools that can be provisioned in a TSPACE instance. The hosted tools can provide support for different types of activities and sub-tasks of those activities, which are represented by *Activity* element. The hosted tools can provide different types of features and can support different types of quality attributes (e.g., multi-tenancy). In the ontology-model, the features and quality attributes of the tools are represented as tools' capability.

We have leveraged semantic integration technologies to support interoperability, however the RA needs to be complemented by appropriate algorithmic solutions for information discovery from the tools. The RA needs to have a set of rules to support collaboration, awareness and information discovery of the related (traceable) artifacts as a project's stakeholders usually perform the activities using different tools. The artifacts are part of high-level representation class (e.g., architecture view) that is shown by *Artifact Representation* element in the model. An instantiation of the TSPACE RA for a specific domain may require additional specializations of *Artifact Representation* concepts such as in the case of software architecting TSPACE, viewpoints and architecture views can be specializations of *Artifact Representation*. The proposed ontology-model provides flexibility to incorporate additional concepts by supporting dynamic composition and aggregation of different concepts in a TSPACE instance.

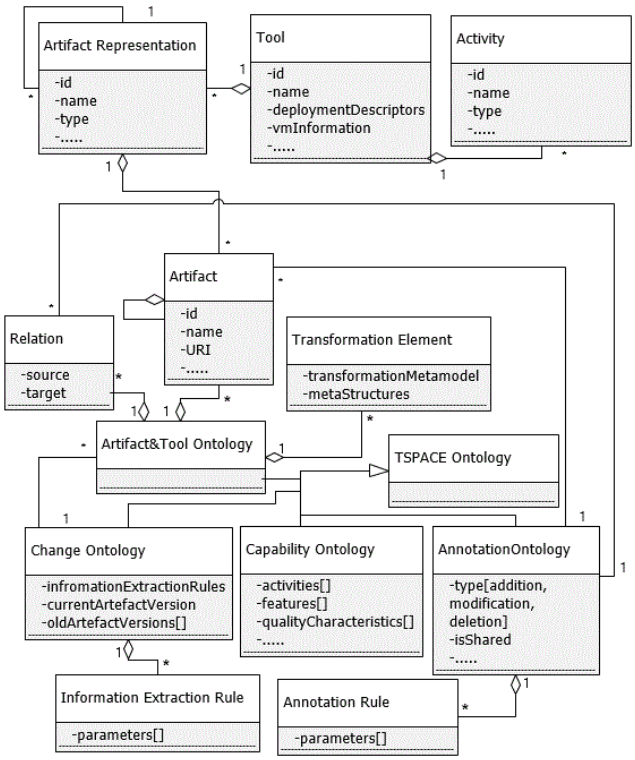

Fig. 1. TSPACE Ontology-Model

TSPACE Ontology element has four specializations as represented by ontology-model that is shown in Figure 1 and captures its elements and relations among the elements. The ontology-model consists of two main ontologies: *Capability*

Ontology and *Artifact Ontology*. *Annotation Ontology* and *Change Ontology* complement the *Artifact Ontology*. The ontologies provide the basis for formalizing the tools selection process, establish the relationship among the artifacts that are produced or consumed in a TSPACE instance, and capture the activities (operations) that are performed on artifacts. Different components of the RA (to be discussed in Section V) use one or more of the TSPACE ontologies. *Capability Ontology* captures features that are supported by the tools registered with the TSPACE and users' tools needs in a TSPACE instance. *Artifact Ontology* captures the relationship among the artifacts in a TSPACE instance. *Annotation Ontology* provides support to annotate the artifacts that are produced or consumed by the tools constituting a TSPACE and provides foundation to manually define or automatically recover trace links between artifacts based on the annotation rules. *Change Ontology* keeps track of the old and new versions of the artifacts in a TSPACE and raises awareness among users with notification according to *Notification Rules*. The ontologies are populated as different activities are performed using the tools in a TSPACE instance and are maintained in Resource Description Framework (RDF)[3] data structure. Figure 1 shows abstract TSPACE ontologies and their relations with each other.

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

Challenges: Building ontologies for complex domains is a non-trivial undertaking. The process of building such ontologies requires expertise in domain knowledge for defining the high-level concepts and relationships between different artifacts of a TSPACE. The ontology-model presented in Figure 1 shows the high-level relationships between different concepts and artifacts of a Software Architecting TSPACE, which can be tailored and extended for other domains.

B. *Using SOA for TSPACE Façade*

For designing the façade of the RA, we used Service Oriented Architecture (SOA) [5] and REST architecture styles [13]. The tools associated with a TSPACE interact with the RA via its façade.

Benefits: The use of SOA and REST makes it easy to modify the RA's components and supports seamless integration of heterogeneous tools to be provisioned.

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

C. *Use of Centralized Repository Pattern to Share Abstract Ontology Templates*

We have used shared repository pattern [14] to provide a common Global Ontology Knowledgebase for TSPACE instances for multiple domains. A centralized ontology repository hosts standard abstract Artifacts Ontology, Annotation Ontology, Change Ontology and Capability Ontology for different domains.

Benefits: A centralized global ontology repository provides a single point of access to different ontologies of a TSPACE. It also positively addresses the flexibility characteristic (QR4) of the TSPACE RA.

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

D. *Use of Pipes and Filter Pattern*

There can be a number of tools in a specific instance of TSPACE and the RA needs to support multiple TSPACE instances. The architectural support is needed to handle an increasing amount of data generated by multiple tools associated with each instance of a TSPACE. That is why we have used two-staged *pipes and filters pattern* [14] in the RA to meet the performance requirements of the platform.

Benefits: The adoption of the two-staged pipes and filter architecture style provides a queuing mechanism to provide support for scalability and multi-tenancy. Pipeline based approach provides support to handle large volume of input data in processing queues. In the first stage, there is a common queue pipeline at which data from all the tools belonging to different instances of a TaaS are received. In the second stage, there are multiple queue pipelines corresponding to an instance of a TSPACE. The input data are sent to the queue of the corresponding tenant with the help of a monitoring filter.

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

E. *Loosely Coupled Layers*

The layered architecture style [14] is widely used to provide loose coupling and separation of concerns in a system. We used the layered architecture at multiple levels of abstraction in the TSPACE RA.

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

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

F. *Tenant Specific Semantic Integration, Information Discovery and Awareness*

Multi-tenancy is an important characteristic of *aaS model [10]. The proposed RA fulfills the multi-tenancy characteristic to provide proper isolation of tools and data of one tenant of a TSPACE instance from other tenants (QR2). The isolation between the architectural elements has been provided at two different levels of abstractions: i) between the ontology instances and corresponding RDF data stores (persistence) of each TSPACE instance of a specific tenant by having logical isolation among the components and ii)

[3] http://www.w3.org/RDF/

between the tools provisioned in a TSPACE instance using virtual machine (VMs) of underlying IaaS cloud.

Benefits: Logical isolation between TSPACE elements that maintain tenant specific information allows customization of TSPACE with respect to specific needs of each tenant.

Challenges: Some domains with stringent multi-tenancy requirements may need adoption for more formal approaches. The WSO2 carbon platform [8] can be used to provide isolation between components of a TSPACE instance. The information flow authentication model based on security policy and role based authorization mechanism can be incorporated to implement security in multi-tenant access points [15]. The multi-tenant access and indexing techniques [8] can be used for multi-tenant persistence of ontologies and corresponding RDF data stores.

V. TSPACE RA Design, Implementation and Overview of Provisioned Tools

We present the TSPACE RA at three 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 RA. The legend presented in Figure 2 shows the notations that are used in the diagrams of the RA.

A. First Level Decomposition

According to the functional requirements, three lifecycle phases of tools (enactment and provisioning, semantic integration and awareness of activities) constituting TSPACE are supported by the TSPACE RA. Figure 2 provides an overall representation of the RA (development view). The modules at first level of decomposition are organized following the layered architecture style [9]. The TSPACE RA conceptually consists of three modules: i) Tools Selection and Provisioning Manager, ii) Semantic Integration Manager, and iii) Awareness and Information Discovery Manager.

The *Tools Selection and Provisioning Manager* enables users to select the tools that are suitable for the activities to be performed and provision the tools using preconfigured Amazon EC2[4] VMs. TSPACE is implemented using JavaEE, SOAP and REST services technologies (JAX-RS[5], JAX-WS[6]). The *Semantic Integration Manager* supports semantic integration among artifacts that are maintained by the provisioned tools. The *Collaboration, 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 RA is used. At the core of the RA, there is an ontology-based semantic integration model (Section IV.A). 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 [14] to support integration among components and modifiability. The decomposition at the first level fulfills the functional requirements discussed in Section III.

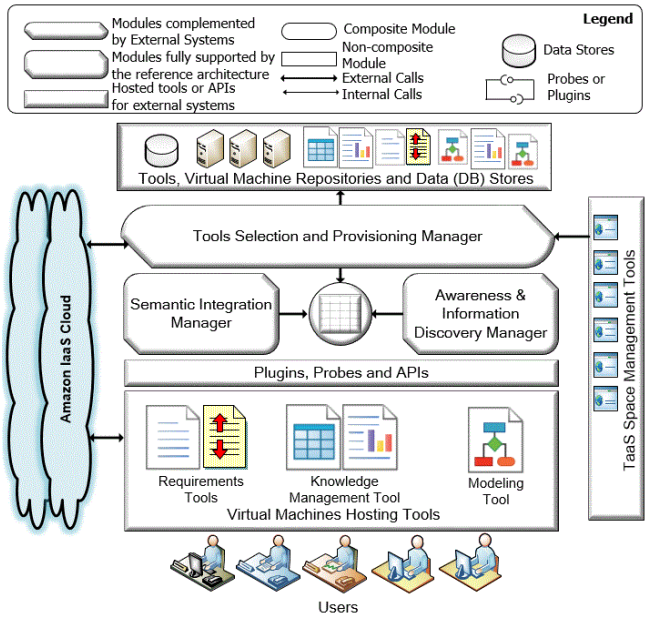

Fig. 2. Overall Architecture of the TSPACE – Development View

B. Second and Third Level Decomposition

The decomposition of the *Tools Selection and Provisioning Manager* is based on requirements FR1 and QR1. FR1 deals with enactment of TSPACE based on the tools' needs for the activities of a specific project and with respect to the location and resource constraints. QR1 deals with automation of the provisioning process. Decomposition of the *Semantic Integration Manager* is based on providing support for semantic integration among heterogeneous artifacts (FR2) and interoperability (QR5). Decomposition of the *Awareness and Information Discovery Manager* is designed to provide awareness to users about different actions that are performed on the artifacts using different tools constituting a TSPACE (FR3). We have also considered the interaction among different components to describe the behavioral model of the RA and have described it in terms of information exchange among the components.

1) Tools Selection and Provisioning Manager: The components constituting this module provide support for tools selection and provisioning. The high-level views of the provisioning architectures synthesized in [8] inspire the RA and have been extended for TSPACE by incorporating the tools selection ontology (Section IV.A) that formalizes tools' capability and users' requirements for tools.

Figure 3 shows decomposition of *Tools Selection and Provisioning Manager*. The *Graphical User Interfaces (GUIs)* provides an interface that supports users interaction and allows administrators to register tools with an instance of the RA, allows stakeholders to specify their tools' requirements and supports administration activities. The *Tools Repository Manager* component maintains a repository

[4] http://aws.amazon.com/ec2/
[5] http://jax-rs-spec.java.net/
[6] https://jax-ws.java.net/

of tools that are registered with the system, the Capability Ontology model of each tool and the VMs that are to be used to host the tools. *Tool Selector* transforms a user's tools' requirements into a relevant ontology and compares it with the Capability Ontology of all the tools registered by the *Tools Repository Manager* by looking if the capabilities (e.g. features) required by the users are supported by the registered tools. *Tools Enactment Preference Manager* takes care of the constraints associated with the enactment of the tools. For example, location constraints require that all the tools for a specific instance of TSPACE shall be provisioned from a public or private IaaS clouds hosted in European Union territory. *Cloud Enactment Engine* enacts tools on an underlying IaaS cloud using *IaaS Cloud Management APIs*. In prototype of TSPACE, we are using Amazon EC2 IaaS cloud to host and provision individual tools using Amazon EC2 APIs[7]. If a private or hybrid IaaS is to be deployed, then a cloud management framework such as IBM Altocumulus Framework [8] can be used.

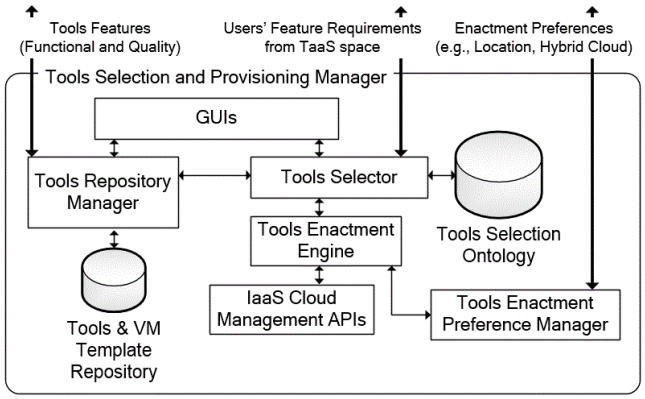

Fig. 3. Tools Selection and Provisioning – Logical View

2) Semantic Integration Manager: The components that are included in this module support semantic integration among the artifacts produced or consumed by the tools that constitute a TSPACE (FR2). There is an ontology based semantic integration model at the core of this module as described in section IV.A. We have implemented the ontology mechanisms using Apache Jena Framework[8].

Figure 4 shows the *Semantic Integration Manager's* decomposition. *Plug-ins (and GUIs)* that are installed on the provisioned tools, link the tools with TSPACE APIs and provide *Semantic Integration Manager* a point of access to the tools. The RA supports the implementation of multiple instances of the TSPACE. The data sent from Plug-ins or GUIs are received at a *Tenant Independent Data Collection Queue*. A *Data Monitor* component monitors all received data elements and filters for forwarding to a *Tenant Specific Data Collection Queue*. There is at least one dedicated data collection (DC) queue for each tenant. If input data streams exceed beyond the acceptable threshold, data collection queues are replicated along with *Data Monitor* component. The monitoring and filtering rules are used by *Data Monitor* to identify tenants from the incoming data stream according to tenant identification specifications.

We have designed a dedicated *Transformation Module* for each instance of the TSPACE. This module handles the data sent by *Tenant Specific Data Collection Queues*. The *Transformation Module* is further subdivided into multiple components. There are two types of ontology knowledge base in the RA: the *Global Ontology Knowledgebase* maintains the tool's *Capability Ontology* and *Artifact Ontology* templates that establishes the relationships among all the possible concepts (the artifacts and their types) that can exist in a specific domain. The *Local Ontology Knowledgebase* maintains the relation between the concepts for a specific instance of a TSPACE corresponding to the tools included in the instance. *Annotation Ontology* and *Change Ontology* only provide annotation templates and change monitoring rules, and these do not need to have tenant specific instantiations. *Ontology Builder and RDF Generator* populates the root Artifacts Ontology based on the data inputs from *Tenant Specific Data Collection Queue*. In TSPACE prototype, *Ontology Builder* maintains the ontology in a proprietary tree like data structure in first stage, which is then transformed into RDF using Apache Jena Framework.

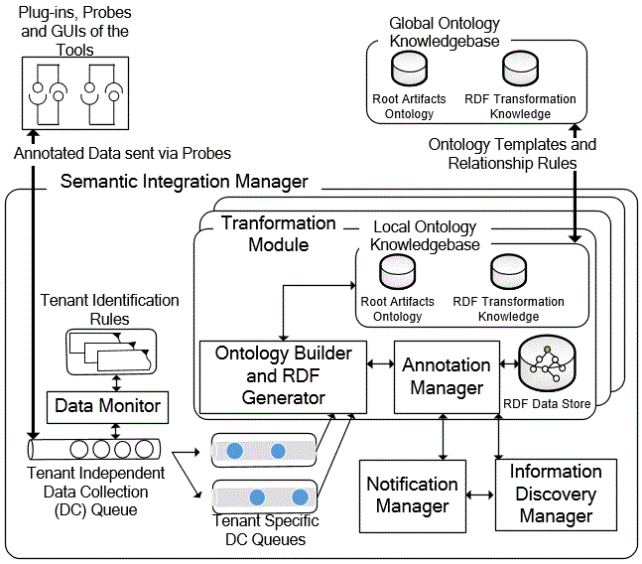

Fig. 4. Semantic Integration – Logical View

3) Notification and Information Discovery Managers: This module provides support to raise awareness about users' activities (FR3) and provides support to trace the changes and the sources of the changes to the artifacts during the lifecycle of a TSPACE. These components leverage the RDF data store that is populated by *Semantic Integration Manager* and use information discovery rules for different types of change and trace notifications.

Figure 4 shows interaction of *Notification Manager* and *Information Discovery Manager* components with *Semantic Integration Manager*. *RDF Data Store* is the core of these components. The Annotation Manager acts as a data input source for *Information Discovery Manager* and *Notification*

[7] http://aws.amazon.com/sdk-for-java/
[8] https://jena.apache.org/

Manager. *Information Discovery Manager* uses predefined information discovery rules that are stored in the information discovery data store. In the prototype implementation of the RA, we are using SPARQL[9] queries for information extraction from RDF data stores. SPARQL provides a configurable and dynamic mechanism to query RDF data structures. *Notification Manager* generates the change and trace notification for the users using the tools according to the notification rules. The notification rules primarily guide for what information needs to be sent to users for trace and change notification, whether the users have subscribed for pull or push notification and what is the criteria and frequency for push notifications.

C. A Case Study

As a case study for proof of concept of the prototype implementation of TSPACE RA, we have integrated PakMe [16], a customized version of the decision support tool ArchDesigner and a modeling tool Microsoft Visio with the platform and provisioned them using Amazon EC2 Virtual Machine templates. PakMe and ArchDesigners are Web-based tools. We have modified their GUIs to integrate those with the platform. Visio is a desktop-based tool that provides support for add-ins. We have implemented an add-in for Visio to integrate it with prototype implementation of the RA for SA TSPACE. A combination of tools that are maintaining the artifacts in their proprietary data structures within the tools (PakMe and ArchDesigner) and as a standalone standardized artifacts (Visio) is selected to demonstrate applicability of TSPACE semantic integration with tools of heterogeneous nature. The screenshot of the PakMe GUI and Visio add-in is shown in Figure 5.

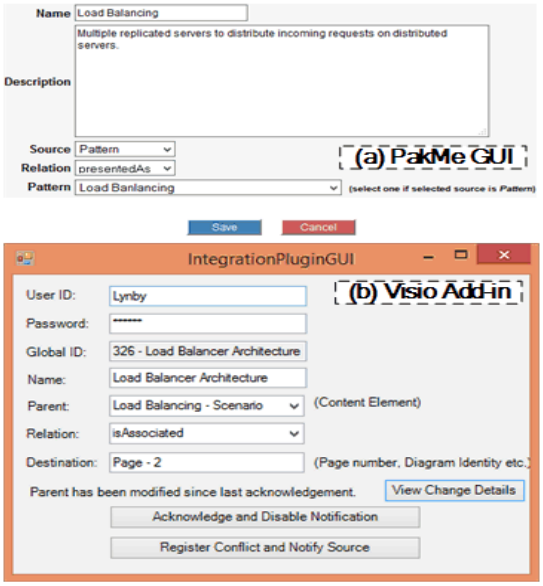

Fig. 5. Prototype GUI and Add-in

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

VI. Evaluation

Software architecture community has developed several methods for architecture evaluation such as Architecture Tradeoff Analysis Method (ATAM) and Software Architecture Analysis Method (SAAM) [17]. We have evaluated the completeness of the proposed RA for functional requirements (FR1, FR2 and FR3) and have used scenarios based evaluation for non-functional requirements (QR1, QR2, QR3, QR4, QR5 and QR6). Because of space limitation, we do not provide the complete details of the architecture evaluation using a scenario based method; rather we report the key reasoning points and outcomes of the evaluation decisions. Table III shows the mapping between the lifecycle phases, functional requirements and corresponding components from the high-level and decomposed architectural representations. It is clear that different parts of the RA provide support for all the phases and corresponding requirements (Req. ID).

TABLE III. Requirements and Components Mapping

Life Cycle Phase	Req. ID	RA Components
Tools Registration	FR1	Capability Ontology and Tools Repository Manager
Tools Selection	FR1	Capability Ontology and Tools Selector
Enactment and Provisioning	FR1	Capability Ontology, Tools Enactment Preference Manager, Tools Enactment Engine and IaaS Cloud Management APIs
Semantic Integration and Interoperability	FR2, QR5	Artifact Ontology, Tenant Independent and Tenant Dependent Data Collection Queues, Data Monitor, RDF Data Store, Ontology Builder and RDF Generator, Annotation Manager, Global Ontology Knowledgebase and Local Ontology Knowledgebase
Awareness on the Activities	FR3	Annotation Ontology, Change Ontology, Information Discovery Manager and Notification Manager

We have presented the RA in terms of its goals that are transformed into functional and non-functional requirements, the TSPACE ontology-model, different modules and components of the RA at three levels of abstraction, and have explained interaction among the components of the RA. It covers all the important dimensions for reporting an RA as per [6] and the views of Rational Unified Process [7]. It also positively addresses the completeness of the RA (QR6). Our decision of using a layered approach supports separation of concerns among the components and high degree of modifiability. The Global Ontology Knowledgebase provides an abstract representation of the TSPACE ontologies and it is materialized using abstract data repository architecture style. It does not only achieve indirection in ontologies but also positively addresses flexibility and Integration (FR2). Façade pattern is used at the interface layer to provide interoperability (QR5) between the tools and the TSPACE RA. A pipes and filter pattern is used to support scalability for handling ontology construction for multiple instances of a TSPACE and to support multi-tenancy (QR2) in the ontology-based semantic integration. The adoption of ontology-based approach for tools selection, provisioning, integration, collaboration and awareness enables the RA applicability (QR6) for supporting

heterogeneous tools and activities in a TSPACE instance. Applicability of the reference architecture is further evaluated by implementing its prototype.

VII. Related Work

Some efforts have been made to report the architecture of cloud-based tools but none of them provides a coherent solution covering all the required dimensions. Calvo et al. propose an architecture for textual information retrieval from cloud-based collaborative writing tools [18] but their effort is only limited to support automated feedback and process analysis of students' academic assignment write-ups. Oliveira and Nakagawa propose a Service-Oriented Architecture for software testing tools [19]. 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 [20, 21]. An extensible architecture description language (xADL) to support integration among architecture centric tools is presented in [22]. Zhao et al. provide a survey of ontologies that have been proposed for software engineering [23]. We have proposed specialized ontologies for the TSPACE RA because the reported software engineering ontologies do not satisfy the specific needs of the RA. In comparison to the discussed existing work in the area, the TSPACE RA 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 about the stakeholders' activities using the bundled suite of tools. The TSPACE RA also supports semantic integration (using TSPACE ontologies) among artifacts consumed or produced during different activities that are performed using different tools that are likely to have proprietary data structures and process models.

VIII. Conclusions

We have presented and discussed the key motivators that have stimulated the requirements for the Tools as a Service Space (TSPACE) Reference Architecture (RA), an ontology-based semantic integration approach that provides the backbone of the proposed RA and different views of the RA at multiple levels of abstractions. The presented RA introduces a standardized view of a TSPACE and has the potential of providing a number of benefits to practitioners and researchers. The RA can provide an increased understanding of the TSPACE for software architecting domain in particular and other engineering domains in general. The main aim of the RA is to facilitate the design of concrete TSPACE systems in various domains. The practitioners can use the RA to communicate a TSPACE instance's requirements and the main architectural principles among software engineering teams. The researchers can use the RA for the identification of potential research areas. Investigation of the application of the existing traceability and information retrieval mechanisms in the context of the TSPACE to provide automated traceability and semantic integration among different types of artifacts is one direction for future research. We also intend to extend the RA ontology-model for other domains and analyze the RA components for the extended model. In the proposed RA, we have discussed security implicitly as part of the multi-tenancy. In the future, we tend to enhance the RA by considering security as an explicit non-functional requirement.

References

[1] P. Louridas, "Up in the Air: Moving Your Applications to the Cloud," *Software, IEEE,* vol. 27, pp. 6-11, 2010.

[2] P. Yara et al.," in *Software Engineering Approaches for Offshore and Outsourced Development*. vol. 35, Springer, 2009, pp. 81-95.

[3] M. A. Chauhan and M. A. Babar, "Cloud infrastructure for providing tools as a service: quality attributes and potential solutions," WICSA/ECSA Companion Volume, Helsinki, Finland, 2012.

[4] M. A. Chauhan and M. A. Babar, "Towards a Reference Architecture to Provision Tools as a Service for Global Software Development," WICSA, 2014, pp. 167-170.

[5] M. N. Huhns and M. P. Singh, "Service-oriented computing: Key concepts and principles," *Internet Computing, IEEE,* vol. 9, pp. 75-81, 2005.

[6] S. Angelov, P. Grefen, and D. Greefhorst, "A framework for analysis and design of software reference architectures," *Information and Software Technology,* vol. 54, pp. 417-431, 2012.

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

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

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

[10] C. P. Bezemer et al., "Enabling multi-tenancy: An industrial experience report," in *Software Maintenance (ICSM), 2010 IEEE International Conference on*, 2010, pp. 1-8.

[11] "ISO/IEC/IEEE Systems and software engineering -- Architecture description," *ISO/IEC/IEEE 42010:2011(E),* pp. 1-46, 2011.

[12] F. Arvidsson and A. Flycht-Eriksson, ""Ontology I". Retrieved 23 June 2014."

[13] R. T. Fielding, "Architectural styles and the design of network-based software architectures," University of California, Irvine, 2000.

[14] F. Buschmann et al., *Pattern-oriented software architecture: a system of patterns*: John Wiley & Sons, Inc., 1996.

[15] J. Bernal Bernabe et al., "Semantic-aware multi-tenancy authorization system for cloud architectures," *Future Generation Computer Systems,* vol. 32, pp. 154-167, 2014.

[16] M. A. Babar and I. Gorton, "A Tool for Managing Software Architecture Knowledge," SHARK, 2007.

[17] M. A. Babar, L. Zhu, and R. Jeffery, "A framework for classifying and comparing software architecture evaluation methods," ASWEC, 2004, pp. 309-318.

[18] R. A. Calvo et al., "Collaborative Writing Support Tools on the Cloud," *Learning Technologies, IEEE Transactions on,* vol. 4, pp. 88-97, 2011.

[19] L. B. R. Oliveira and E. Y. Nakagawa, "A service-oriented reference architecture for software testing tools," in *Software Architecture*, ed: Springer, 2011, pp. 405-421.

[20] R. Wolvers and T. Seceleanu, "Embedded Systems Design Flows: Integrating Requirements Authoring and Design Tools," in *Software Engineering and Advanced Applications (SEAA), 2013 39th EUROMICRO Conference on*, 2013, pp. 244-251.

[21] M. Biehl, J. De Sosa, M. Torngren, and O. Diaz, "Efficient Construction of Presentation Integration for Web-Based and Desktop Development Tools," COMPSACW, 2013, pp. 697-702.

[22] R. Khare et al., "xADL: enabling architecture-centric tool integration with XML," *Hawaii International Conference*, 2001, p. 9 pp.

[23] Y. Zhao, J. Dong, and T. Peng, "Ontology Classification for Semantic-Web-Based Software Engineering," *Services Computing, IEEE Transactions on,* vol. 2, pp. 303-317, 2009.