

PTaaS: Platform for Providing Software Developing Applications and Tools as a Service

Muhammad Afeef Chauhan
PhD Student
Software and Systems Section
muac@itu.dk

Muhammad Ali Babar
Professor
Software and Systems Section
maba@itu.dk

Copyright © [2014], Muhammad Afeef Chauhan
Muhammad Ali Babar

IT University of Copenhagen
All rights reserved.

Reproduction of all or part of this work
is permitted for educational or research use
on condition that this copyright notice is
included in any copy.

ISSN 1600-6100

ISBN 78-87-79-49-314-8

Copies may be obtained by contacting:

IT University of Copenhagen
Rued Langgaards Vej 7
DK – 2300 Copenhagen S
Denmark

Telephone: +45 72 18 50 00
Telefax: +45 72 18 50 01
Web: www.itu.dk

IT University of Copenhagen

Software and Systems Group

**PTaaS: Platform for Providing
Software Development
Applications and Tools as a Service**

Supervisor
M. Ali Babar

Student
Muhammad Afeef Chauhan

Acknowledgement

I would like to acknowledge valuable contributions from my supervisor, M. Ali Babar for mentoring me to carry out the research presented in the thesis. I would also like to thank him for providing me opportunity to carry out research under his supervision, his guidance in designing and refining the research objectives, reviewing my work, answering my questions and providing valuable feedback on intermediate deliverables. It would not have been possible to complete this work without his active involvement and thorough guidance.

Muhammad Afeef Chauhan
May 21, 2013

Abstract

Cloud computing has become an established paradigm for enabling organizations to build scalable software systems and to meet challenges of rapid demand of computing and storage resources. There has been a significant success in building cloud-enabled applications for many disciplines ranging from web based and mobile application to intensive video and data processing systems. This initial success of cloud has opened new horizons for more complex domains. Global Software Development (GSD) is one of such domains. GSD is different than traditional applications domains because of involvement of large number of complex activities that does not only include technological aspects but also social aspects. A large number of applications and tools have been devised for providing solutions to the challenges of the GSD that emerge as a result of distributed development teams. However, the technological solutions that have been proposed so far are limited in their ability to meet specific GSD challenged and emerging trends of GSD in which software development is not only global but it also involve multiple organizations. Involvement of the multiple organizations in GSD increase the complexity of aligning their processes and establishing technology support needed to facilitate working according to new processes. Although the benefits of using cloud computing to solve GSD issues have been discussed in the literature but there has not been a significant attempt to provide fully functional technological support for it that is not limited to one specific tools and a particular phase of software development life cycle. In this thesis, we have explored the possibility of offering software development applications and tools as services that can be acquired on demand according to the software development process in globally distributed environment. We have performed the structured review of the literature on GSD tools to identify attributes of the software development tools that have been introduced for addressing GSD challenges and we have discussed significance of technology alignment with process. Information gained from the review of literature on GSD tools and processes is used to extract functional requirements for the middleware platform for provisioning of software development applications and tools as services. Finding from the review of literature on architecture solutions for cloud-enabled systems has been used to seek inspirations for providing architecture of the platform. In order to demonstrate feasibility of our proposed architecture, we have implemented a prototype of the middleware using Amazon as underlying IaaS Cloud and have demonstrated its functionality with the help of simple client application that act as a bridge between software development tools and middleware platform.

Table of Contents

1. Introduction	13
2. Related Work.....	17
2.1. Question 1: Types and characteristics of applications and tools used for supporting GSD activities	18
2.2. Question 2: Role of Process in GSD	19
2.3. Question 3: Limitations of Existing Solutions	20
2.4. Question 4: Key Characteristics of Cloud Enables Systems.....	21
2.4.1. Multi-tenancy.....	21
2.4.2. Supporting Multiple Types of Client Devices.....	21
2.4.3. Middleware and Platforms.....	21
2.4.4. Workflow Support	22
2.4.5. Consideration for Building Cloud Applications.....	23
2.4.6. Data Management	23
2.4.7. Managing Distributed Resources	23
2.4.8. Risks in Cloud-Enabled Systems.....	23
2.4.9. Resource Scalability	24
2.4.10. Reliability.....	24
2.4.11. Service Adaptability	25
2.5. Discussion	25
3. PTaaS Requirements	27
3.1. Requirements for Platform Features.....	28
3.2. Quality Requirements.....	30
3.3. Scenario Analysis.....	30
4. Architecture of PTaaS	33
4.1. Interface Layer	35
4.2. Accounting Services.....	36
4.3. Applications, Tools and Services Management Layer	36
4.3.1. Repository Manager.....	37
4.3.2. Multi-tenancy and Location Manager	37
4.3.3. Life Cycle and Provisioning Manager	37
4.3.4. Service Composer and Orchestrator.....	38
4.3.5. Authentication and Process Manager	40
5. Proof of Concept.....	45
5.1. Architecture Description Methodology	45
5.2. Use Cases:	46
5.2.1. Authentication.....	46
5.2.2. Development Process Handling.....	47
5.2.3. User Access to Resources	47
5.2.4. Support for Collaborative Work	47
5.2.5. Support for Traceability	48
5.2.6. Accounting Services	48
5.2.7. Collaboration Activities.....	48
5.2.8. Notifications for Traceability	48
5.2.9. Posting and Retrieving Generated Artifacts at Collaboration Points	49
5.3. Deployment View	49
5.4. Logical View.....	50
5.5. Process View	54
5.6. Data Model.....	56

5.7. Implementation Overview	57
5.7.1. Middleware Platform Services	57
5.7.2. Client Application for Defining Software Development Process in the Middleware and Invoking Application and Tools	58
5.7.3. Desktop Client	61
6. Conclusion and Directions for Future Work.....	65
7. Appendix.....	77
A.1. PTaaS APIs.....	79
A.1.1. Overview of OCCI Specification.....	79
A.1.2. APIs to Access Platform Features.....	82
A.1.2.1. Authentication Mechanism.....	82
A.1.3. Platform APIs	83
A.1.3.1. APIs for Tenant Management.....	83
A.1.3.1.1. Get Information of an Existing Tenant.....	83
A.1.3.1.2. Register a New Tenant.....	83
A.1.3.1.3. Update Information of an Existing Tenant.....	84
A.1.3.1.4. Delete a Tenant.....	85
A.1.3.2. User Management.....	85
A.1.3.2.1. Get Information of an Existing User.....	85
A.1.3.2.2. Register a New User.....	85
A.1.3.2.3. Update Information of an Existing User.....	86
A.1.3.2.4. Delete a User.....	86
A.1.3.3. Registration of Applications, Tools and Services.....	86
A.1.3.3.1. Get Information of a Registered Application, Tool or Service.....	86
A.1.3.3.2. Registered a new Application, Tool or Service.....	87
A.1.3.3.3. Update a Registered Application, Tool or Service.....	88
A.1.3.3.4. Delete an Application, Tool or Service.....	89
A.1.3.4. Collaboration APIs.....	90
A.1.3.4.1. Initiate Collaboration Activity.....	90
A.1.3.4.2. Register Application, Tool or Service in a Collaboration Activity.....	90
A.1.3.4.3. Unregister Application, Tool or Service in a Collaboration Activity.....	91
A.1.3.4.4. Post Data in a Collaboration Activity.....	91
A.1.3.4.5. Get Data from a Collaboration Activity.....	91
A.1.3.4.6. Terminate a Collaboration Activity.....	92
A.1.3.5. Workflow Management.....	92
A.1.3.5.1. Process Workflow Management.....	92
A.1.3.5.1.1. Create Process Workflow.....	92
A.1.3.5.1.2. Get Process Workflow.....	92
A.1.3.5.1.3. Update Process Workflow.....	93
A.1.3.5.1.4. Delete Process Workflow.....	93
A.1.3.5.2. Node Managements on a Process Workflow.....	93
A.1.3.5.2.1. Add Node in a Process Workflow.....	93
A.1.3.5.2.2. Get Node in a Process Workflow.....	94
A.1.3.5.2.3. Update Node in a Process Workflow.....	95
A.1.3.5.2.4. Delete Node in a Process Workflow.....	95
A.1.3.5.3. Managing Tools assigned to the Workflow Node.....	96
A.1.3.5.3.1. Assign Application, Tools or Services to a Node.....	96
A.1.3.5.3.2. Get Application, Tools or Services to a Node.....	96
A.1.3.5.3.3. Update Application, Tools or Services to a Node.....	97
A.1.3.5.3.4. Update Application, Tools or Services to a Node.....	97
A.1.3.5.4. Managing Tenants assignment to the Nodes in the Workflow.....	98
A.1.3.5.4.1. Assign Tenants to a Node.....	98

A.1.3.5.4.2.	Get Tenants Assigned to a Node.....	98
A.1.3.5.4.3.	Update Tenants Assigned to a Node.....	99
A.1.3.5.4.4.	Delete Tenants Assigned to a Node	99
A.1.3.6.	APIs for Supporting Traceability.....	100
A.1.3.6.1.	Register Artifact for Traceability and Get Identifier	100
A.1.3.6.2.	Get Identifier of an already Registered Artifact	100
A.1.3.6.3.	Update an already Registered Artifact.....	101
A.1.3.6.4.	Delete a Registered Artifact.....	101
A.1.3.6.5.	Register Callback Method for Traceability	101

Table of Figures

Figure 1: Context of Middleware Platform.....	33
Figure 2: Subsystems making up Middleware Platform	34
Figure 3: Interface Layer.....	35
Figure 4: Provisioning Management.....	38
Figure 5: Composition Scheme.....	39
Figure 6: Composition Handling.....	39
Figure 7: Authentication and Process Management.....	41
Figure 8: Collaboration Handling.....	42
Figure 9: Collaboration Flow	43
Figure 10: Use Cases	46
Figure 11: Deployment View	50
Figure 12: Sequence Diagram - Define and Enact Development Process – (i)	51
Figure 13: Sequence Diagram - Define and Enact Development Process – (ii).....	51
Figure 14: Authentication Service	52
Figure 15: Using XML annotations for converting java objects into XML documents.....	52
Figure 16: Code for enacting Amazon EC2 Instances hosting Applications and Tools.....	53
Figure 17: Node Entity.....	53
Figure 18: Activity diagram for enacting software development process	54
Figure 19: Activity diagram for collaboration.....	55
Figure 20: Activity diagram for traceability.....	56
Figure 21: Data Persistence Model.....	57
Figure 22: Authentication	59
Figure 23: Creating Process (i).....	59
Figure 24: Creating Process (ii)	59
Figure 25: Adding Nodes/Sites to the Process.....	60
Figure 26: Defining Data Movement Sequence Between Nodes/Sites	60
Figure 27: Displaying Access Information	61
Figure 28: Desktop Client Supporting Collaboration and Traceability – (i).....	61
Figure 29: Desktop Client Supporting Collaboration and Traceability - (ii)	62
Figure 30: Desktop Client Supporting Collaboration and Traceability - (iii).....	63

1. Introduction

On demand resource provisioning model of cloud computing has opened new horizons for organization to meet increasing demand of computing and storage resources without huge upfront investments [1-3]. It does not only provide scalability but also provides freedom from low-level configuration and maintenance tasks associated with infrastructure as well as services hosted on infrastructure [1, 2]. There are a large number of studies reporting application development using cloud computing targeting various domains including information systems [4, 5], video processing [6-9], e-government [10-12] and health care [13-15]. Initial success of cloud computing has opened new horizons for applications domains, which are of high complexity and are diversified in nature.

Cloud computing offerings are broadly classified into three services and five deployment models [1, 2, 16-18]. Three categories of service models are: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Five deployment models include: public, private, hybrid, community and virtual private clouds. **IaaS** cloud provides abstraction to underlying computing, storage and network resources using virtualization technologies. It also provides basic software resources such as operating systems to utilize the virtualized hardware resources. IaaS Clouds poses additional overhead to applications and technical staff for monitoring and optimizing resources to meet Quality of Service (QoS) requirements according to Service Level Agreements (SLAs). However, this infrastructure has advantage of support for customization. Additional tools and software can be installed as per requirements of the applications and end users. Amazon elastic cloud [19], Amazon storage services [19], Eucalyptus¹ and OpenNebula² are example of IaaS cloud. **PaaS** cloud provides application programmable interfaces (APIs) to develop applications. Application build using PaaS APIs do not need to handle resource provisioning from underlying infrastructure. Google app engine³, Microsoft azure platform⁴ and Salesforce⁵ are examples of the PaaS cloud. **SaaS** represents applications that are build on top of either IaaS or PaaS clouds and offer business solutions to end users. One of the key features of these applications is multi-tenancy. It enables single instance of the application to service large number of organizations and end users. SaaS provide limited support for application customization.

Public clouds represent cloud infrastructure and software resources maintained by an organization and is offered to end users for lease on basis of some pricing model. End users can access infrastructure through Internet. Amazon elastic computing (EC2) and storage (S3), Google app engine and Microsoft azure are examples of public clouds. **Private clouds** represent infrastructure and software resources maintained by organization for its internal use. In some cases, organizations adopt a hybrid strategy

¹ <http://www.eucalyptus.com/>

² <http://opennebula.org/>

³ <http://code.google.com/appengine/>

⁴ <http://www.windowsazure.com/>

⁵ <http://www.salesforce.com/>

and combine private infrastructure with public clouds. It is called *hybrid cloud*. *Virtual private cloud* (VPC) [18] and *community cloud* [16] are build on top of the public and private clouds [20]. A VPC utilize resources of public cloud with additional features of virtual private network. It provides support for customizable network topology an security settings [20]. In some cases, organization with shared business objectives decide to collaborate with each other and form a common cloud by combining their private cloud. It is referred as *community cloud*.

Global Software Development, which is characterized as software development from geographically distributed locations, has specific challenges that are not easy to address using conventional desktop and client/server technologies and infrastructure. The challenges occur as a result of:

- Involvement of stakeholders in software development process from geographically distributed locations.
- A large number of software applications and tools used for development as well as management activities.
- Tractability between artefacts that are produced and consumed by multiple tools.
- Specific requirements of data privacy and constraints on their storage locations because of sensitive nature of data and legal constraints imposed by organizations involved in GSD project.
- In many cases, involvement of more than one organization in a GSD project that maintains their own isolated set of applications and tools.
- Software development activities are carried out by following some specific software development process and organizations have to tailor their tools suites according to the processes.

Cloud computing has potential to address challenges associated with software development in general and GSD in particular. Potential benefits of using cloud computing in Global Software Development have been discussed in literature [21, 22]. Many of the benefits that can be achieved by incorporating cloud-computing paradigm are targeted to problems that emerge as a result of distance, differences in time and culture, and lack of face-to-face communication support. Some of the potential benefits that can be achieved by incorporating cloud-computing in GSD are possibility for increased coordination, easy support for technical development, minimize impact of geographic distances, enhanced communication and collaboration, and rapid solutions to technical issues [21, 22]. Moreover, applications and tools provisioning as services can reduce tools licensing costs, facilitate easy alignment of tools with processes, support enhanced traceability between artefacts managed by geographically distributed teams, enable manipulation of huge data volumes of sensitive nature, provide easy access to expensive technology and sensitive hardware, and enhance awareness and collaboration [22].

Few studies have reported potential benefits that can be achieved by introducing cloud computing paradigm in GSD [21, 22] and have elaborated high-level requirements as well potential solutions [22]. However, to the best of our knowledge there are no comprehensive solutions available that use cloud-computing paradigm to support software development activities throughout GSD lifecycle. In order to support the provisioning of GSD applications and tools, we do not only need to provision them rapidly but also be able to combine individual services inside applications and tools to match end user requirements. It is important to bundle the applications and

tools together so that they can be offered as a suite of tools that can seamlessly be adopted by a organizations involved in GSD project. When we consider bundling heterogeneous tools and applications together, there is also a need for providing support for traceability among artefacts being maintained by tools and supporting collaboration activities among applications/tools. The applications and tools may compose multiple services inside and the services may have multiple versions, hence we also need to incorporate composition of services according to the requirements.

Software development processes are main drivers of development activities. When bundled suite of applications and services are provisioned, they need to be aligned on the software development process and data transformation between them should take place according to the defined process. Last but not the least, all the provisioning should take place according to the according to the location constrains of the data. To meet the challenged associated with a GSD we have provided a middleware **Platform for providing software development applications and Tools as a Service** named **PTaaS**. The platform presented in this thesis is a first step to move forward in addressing GSD challenges. Our aim is to complement existing pool of available applications and tools by providing a platform that supports on demand provisioning and seamless integration among them. In the thesis; we have discussed the possibility for provisioning software development applications and tools as services, the important features and requirements for the platform to support offering of applications and tools as services in GSD context, and software architecture of the platform that satisfy the requirements. Although the platform may be used to support collocated teams as well but our focus is on GSD; hence, the discussion carried out in the thesis is focusing on GSD. Specifically, we address following research questions.

1. What are important characteristics of the GSD tools, which should be considered while providing a platform for their provisioning as services?
2. What are important attributes of software architectures for cloud-enabled systems, what are different approached for achieving those attributes?
3. How the knowledge gain from research question 1 and 2 can be combined and can be extended to provide architecture for the middleware platform that can support provisioning of software development applications and tools as services?

In order to find answers of research question 1 and 2, we have discussed literature relevant to GSD and software architecting techniques for cloud-enabled systems in considerable depth in Chapter 2. We have used this knowledge to extract requirements for the middleware platform and have presented the requirements in Chapter 3. In Chapter 4 we have provided reference architecture of the middleware platform for satisfying the requirements described in Chapter 3. In Chapter 5, we provide detail of the prototype implementation of the platform to demonstrate proof of concept. In Chapter 6 we have provided summary of the topics to conclude the thesis and have provided direction for future work. Detail of APIs to access important features of the platform is provided in Appendix.

2. Related Work

In this chapter, we present an overview of the related work to gain through understanding of the literature that can be used to provide software development applications and tools as services using scalability and resource provisioning model of cloud computing to meet Global Software Development (GSD) challenges. The literature discussed in this chapter is aimed at not only providing a useful insight into the state of the art practices for designing and implementing systems using cloud computing resource provisioning model but we also aim at discussing specific GSD challenges that arise the need for having software development applications and tools provisioning as services. We discuss the current web enabled applications and integrated development environments (IDEs) being offered by different vendors and discuss why the currently approaches are not sufficient to provide solutions to GSD challenges. We also elaborate specific GSD dimensions to find out specific features that our envisioned reference architecture should support. The related work discussed in this section aims at addressing the following questions.

Question 1: *What type of application and tool support is available for GSD projects and what are primary problems that the applications and tools are addressing?* The software applications in general and GSD applications in particular can be provided in many different forms and at many different layers. For example, applications can be provided as independent tools, in form of integrated development environments and or in terms of middleware platforms. First purpose of this question is to identify the different types of applications and tools. Second purpose is to identify what are key features of the applications and tools used in GSD. It is important to find out key features and characteristics so that the envisioned reference architecture of the platform for hosting GSD applications and tools can be designed accordingly.

Question 2: *Does shared software development process among multiple sites involved in GSD play a role in success of GSD project?* This question is aimed at finding evidence regarding the importance of having a shared development process and a common set of applications and tools aligned with the process.

Question 3: *Do existing cloud-enabled applications and tools are able to provide an appropriate solution to meet GSD challenges? Are there any shortcomings that hinder the adoption of existing cloud-enabled solutions?* These research question aims at briefly analyzing existing solutions and find a gap between offerings and needs.

Question 4: *What are key characteristics of the cloud-enabled systems that should be considered while designing and constructing them?* As stated in introduction that the main objective of this thesis is to propose reference architecture for facilitating software development applications and tools as services. By exploring different dimensions of the cloud-enabled systems, we get an insight to the related aspects of proposing reference architecture for the platform.

In the following subsections, we explore related literature to find answers of the research questions. We use important findings from the literature to extract

requirements of the platform for providing software development applications and tools as services (described in chapter 3 of the thesis).

2.1. Question 1: Types and characteristics of applications and tools used for supporting GSD activities

It is stated earlier that GSD is characterized by involvement of distributed development teams from geographically distributed locations and it has emerged as one of the most popular model for developing software [23, 24]. In order to successfully leverage the advantages of the GSD, appropriate tooling support is required for all the activities encompass in GSD [25]. There are a large number of open source and closed source tools available to support GSD activities [26]. Tell and Ali Babar has also provided a comprehensive overview of the available tools for supporting GSD in their systematic mapping study on Tools for Distributed Software Development Teams [27]. They have classified studies into categories including communication, collaboration and coordination. Steinmacher et al. have also described communication collaboration and coordination as primary means to raise awareness among distributed development teams [28]. Tell and Ali Babar [27] has argued that applications and tools for supporting GSD are offered in following forms or types.

- ***Stand along applications and tools:*** These are independent applications and tools developed by vendors to address a specific activity of GSD.
- ***Frameworks:*** Applications structures to provide a foundation for building domain specific tools.
- ***Middleware and platforms:*** These are middleware software implemented to facilitate integration between compatible applications and supporting intra operability.
- ***Integrated Development Environments:*** These tools consist of development environments for programming languages with the ability to integrate with related applications, e.g. databases, applications servers and version control repositories.
- ***Plug-in:*** There are extensions developed on top of existing applications or integrated development environments by using the APIs provided by them.

Questions 1 also aim at finding problems, which software development applications and tools are trying to address so that we can identify support required by the platform for provisioning them as services. For that purpose we have reviewed the problems that applications and tools used in GSD are trying to solve. We have gathered the list of tools and corresponding literature reported in mapping studies on software development tools used in GSD [27, 29]. Our aim is only to find out the key characteristics that are to be supported by the envisioned platform for hosting GSD tools as services; hence we are providing main classes of the problems reported in the [27, 29] below without going into details of studies.

Awareness, Collaboration and Coordination:

It is one of the primary features being focused by tools for supporting GSD. Awareness is defined as “an understanding of the activities of others, which provides context of one’s own activities” [30]. Awareness is improved by allowing people to coordinate and collaborate with each other. The objective of collaboration is to allow people to collaboratively work with each other in relevant group activities [28], Whereas coordination focuses on making people aware of their activities and its

effects on others [28]. Collaboration and coordination are considered essential to raise awareness among team members. The large number of studies reporting the need for awareness [30-33], collaboration [34-37] and coordination [23, 38-41]; and proposing tool support emphasize their critical role in GSD and the need to incorporate them in new paradigms for supporting GSD activities.

Support for Integration:

Software development tools need to be properly integrated with each other in order to support end-to-end software development life cycle to maximize the productivity of the stakeholders involved in software development activities [42]. Maalej has also reported challenges that traditional software integration mechanisms fail to cater [42]. These challenges are associated with retrieving artifacts and making establishing context of the change on local environment, synchronization of the development artifacts among development teams, manual synchronization of artifacts and corresponding working plans and sharing context of the changes.

Virtual Communities/Teams:

The stakeholders involved in GSD are often referred as virtual teams because they do not share physical locations but still work as teams. Although software development activities on distributed locations are controlled through processes; but the unavailability of the thorough integration between application, tools and technologies used make it hard to keep the stakeholders aware of the each other activities when working in a collaborative fashion [43]. Choosing appropriate tools for virtual teams involved in distributed development is also not straight forward, specially when participating sites of GSD are not part of same organization [44]. A large number of studies have been reported to address challenges associated with virtual teams [44-47] highlighting the importance of explicitly taking care of technology need for virtual teams for supporting tool integration, collaboration, coordination and support for adoption of desired applications and tools.

Traceability and Automated Updates:

Providing traceability between artifacts being maintained at distributed sites is of high importance in order main integrity of products under development. Souza et al. have highlighted the need for having explicit support for traceability through visualization and emphasized that tools should not only offer support for traceability but should also keep record of producers and consumers of the artifacts [48]. Although traceability between a unified set of tools offered by some vendors specific tools (e.g. IBM Rational Suite), but providing traceability between across heterogeneous tools is a challenging task because of unavailability of the information about the users and context of the artifacts.

2.2. Question 2: Role of Process in GSD

The role of software development process in success of GSD projects for successful coordination while performing team activities has been discussed in many studies [32, 49-51]. Dustdar and Gall [32] has argued the need for process awareness of virtual teams involved in GSD and described information sharing, process sharing and process composition as key requirements to be met by the automation support. They have also emphasized that the need for synchronous collaboration of artifacts. Wiredu [49] has marked process as an important dimension in his framework for analyzing coordination in GSD and has phrased is as “mode of operation”. Bhat et al. has [51]

marked process as a strategic success factor for requirements engineering and have argues that lack of shared process can result in conflicting requirements. These studies shows that the absent of a shared process across sites involved in GSD can result in improper artifacts. All of these studies emphasize that any technology support for providing for GSD should keep software development process at it core.

2.3. Question 3: Limitations of Existing Solutions

There are many web based and cloud based applications available to support GSD. These tools include project management applications, tools to facilitate distributed teams engage in collaborative activities like planning, software design tools, development IDEs, and collaborative text writing tools. A list of some poplar cloud enables applications and tools is shown in Table 1 along with their main features. All the tools provide comprehensive solutions to address GSD problem. However, their inability to integrate with each other make them less effective to provide end-to-end solutions. Many of them (for example, LucidChart, Cloud9 and eXo) does not provide control over acquiring them according to location and security constraint (for example secure multi-tenant support) make them not usable in projects that involve working on security sensitive artifacts. Moreover, few tools including development IDEs need to integrate with a large number of other applications and tools (for example, version control systems, applications servers and databases) before they can be efficiently used in the software development environment. There are also integration frameworks like IBM Jazz Platform [52] those provide comprehensive support for collaborative software development and has been discussed in literature. However, the infrastructure and cost required to set up Jazz Platform make it less effective to be used in small and medium sized organization. These limitations raise the need to have a new paradigm that can incorporate integrate platforms like IBM Jazz and provide on demand provisioning of applications and tools to support their integration based on the data artifacts they produce or consume and allow their provision according to location and security (secure multi-tenancy) constraints.

Table 1: Available Cloud-enabled Solutions

Tool Name	Features
AgileZen [53]	Collaborative project management.
LucidChart [54]	Tool to create flowcharts, organizational charts, website wireframes, UML designs, mindmaps, software prototypes.
MeetingSphere [55]	Group meeting and decision support system.
Microsoft Live Meeting [56]	Web conference service.
Microsoft Project [57]	Project management solution.
Microsoft Team Foundation Server [58]	Source control, data collection/reporting and project tracking.
Pidoco [59]	Software to design GUIs for web and mobile apps and make it live to share with other users.
IBM Rational Software Solution for Cloud [60]	Suite of tools consisting Rational Team Concert, Rational Quality Manager, Rational Requirements Composer, Rational Asset Manager, Rational Build Forge, Rational Solution for collaborative lifecycle management, and IBM Smart Cloud.
Cloud9 IDE [61]	Cloud-anabled Online IDE.
Eclipse Orion [62]	Cloud-enabled IDE that can be hosted on private/public clouds.
eXo Platform [63]	Collaboration Platform and IDE.

2.4. Question 4: Key Characteristics of Cloud Enabled Systems

There are a large number of studies focusing on solving problems for cloud-enabled system at different levels of abstractions. We have picked up selected number of paper from the available literature to find out answer of question 4. These papers are grouped into different categories of themes on the basis of the problems they are addressing to identify important characteristics of the cloud-enabled system. These characteristics are used to identify requirements of the platform as described in Chapter 3 and reference architecture as described in Chapter 4.

2.4.1. Multi-tenancy

Multi-tenancy is a characteristic of cloud-enabled applications to serve multiple clients through same instance of application. Multi-tenancy in cloud enabled system can be achieved at different levels of abstractions, i.e. at service level as well as at data level.

Multi-tenancy at middleware is supported by assigning separate service instances from repositories to each tenant [64, 65]. Information of services corresponding to tenants is maintained in service registry and a message handler at server routes client requests to services with the help of message dispatcher. A security component is responsible for handling user management by considering roles assigned to each user belonging to a specific tenant along with resources assigned to tenants and actions that users can perform on the specific resource. Services of a specific tenant have their own infrastructure, persistence units and management console for managing infrastructure.

Multi-tenancy at database layer can be achieved by implementing a database driver capable of maintaining database index that can provide a mapping between tenants and their databases [66]. Moreover, inside each database, every table has an additional column that specify owner of the data. Having this type of multitenant database index allows applications to query databases in normal way without explicitly selecting separate databases for each tenant.

2.4.2. Supporting Multiple Types of Client Devices

The ability of being part of network enables different types of client devices to utilize features of cloud computing for accessing smart and reliable applications [13, 67]. Limitation of processing and storage resources on mobile devices can be overcome by taking advantage of automated, dynamic and reliable runtime configuration of components on mobile devices and on cloud [13, 68]. Spatial and context awareness features allow building location aware services on mobile devices [67-69].

The studies [70, 71] discuss distribution schemes of components on mobile devices to perform complex tasks. The technique described in [71] divides components into two categories; i) components used for interaction with end users and ii) components used to perform business logic. Components used for business logic can be deployed on available mobile devices even though the devices are not being used by end users for interacting with system. The study [70] presents algorithms for selecting optimal configuration of components on mobile and on cloud to take maximum advantage of mobile platforms and network bandwidth while keeping cloud resources and cost and bare minimum level.

2.4.3. Middleware and Platforms

The papers grouped into this category described middleware software platforms that are supporting services and applications hosting on cloud infrastructure and their

interaction with each other. The studies [13, 72, 73] describe middleware support for reliable automated deployment of services and components on cloud as well as on front-end machines according to Quality of Service (QoS) requirements. The communication between services satisfying QoS agreements is also the focus of these studies. Representational State Transfer (REST) based approach for managing infrastructure and REST based services on cloud is described in [74, 75]. The techniques separately maintain configuration of distributed services and services associated with domain aware processes. This configuration is managed at three different levels of isolation: information common for all services containing contract information and device configuration, standard management information of services defining standard specifications, and private information of services like QoS requirements and Service Level Agreements (SLAs). Altocumulus framework by IBM is described in [76] which address intra operability issues between clouds. It provides Dashboard supporting interaction point between end users, a core encapsulating rules and cloud adapters for supporting execution on different cloud platforms. An extended YML framework for scientific computing is presented in [77] explaining different components of the framework used in scientific computing.

Cloud-enabled applications can server a large number of tenants that can use different versions of the same software. Architecture for maintaining multiple versions is presented in [78] and suggested need of version management system module capable of maintaining partitioning between different versions of the system and capable to bundle specific version of the system with respect to a particular tenant's subscription. The platform presented in [79] allows home services to be selectively opened to remote users and semi-trusted external services. The motivations to host data and services at private self managed location is availability of large and cheap storage availability as compared to the online services, data privacy with full administrative and legal rights, real-time data access if the data is available directly from private locations and common repository with all contents at the single location. Two steps request filtration mechanism is suggested by using with trusted proxy server on public Internet as first level of filtration and tunnel server between Internet and home network as second level of filtration.

2.4.4. Workflow Support

Papers included in this category provide architectures for trust worthy and continuous data processing in open distributed systems, workflow management for processing huge volumes of data and distributed processing on sensitive data [80-84]. This section focuses on architecture constructs for taking advantage of combining private cloud with public infrastructure in workflow bases system.

The data processing approach presented in [80] propose that every component should know about minimum part of the workflow rather than knowing all of it. Components keep track of each data unit they receive and produce and know only about its upstream and downstream components. A randomized consistency check is performed for verification and integration of scalable dataflow processing. For distributed processing of data, use of cloud side engine and client side engine is presented in [82]. These engines are responsible for managing components' configurations according to data sensitivity requirements as well as with respect to processing capabilities of infrastructure at client and cloud side. The studies [81, 83, 84] describe components of workflow systems and multiple stages involved in data

processing. The components are distributed among user layer, infrastructure layer and middleware. Cloud side engine and user end engine is responsible for components distribution on user end and cloud end. Data collection, re-projection, analysis, task scheduling and processing correspond to multiple data processing stages.

2.4.5. Consideration for Building Cloud Applications

Studies presented in [85, 86] layouts factors to consider for considerations while building applications for cloud. The main considerations are integration with existing ecosystem of application, exposing reusable business logic as services, subscription based resource provisioning of cloud resources through application and service offerings according to QoS requirements. The cloud aware approach is presented in [86] that suggests building applications after considering underlying cloud virtual infrastructure of the target platform in order to achieve application's non functional quality attribute requirements. The cost of CPU, storage and networking resource should be carefully analyzed for estimating overall application cost.

2.4.6. Data Management

Multiple alternatives for implementing databases in cloud applications are presented in [87-90]. Data partitioning and replication is commonly used technique for handling data in distributed environment while exposing database services as REST interfaces is convenient approach for using a shared database by multiple services. Memory-based column database is presented in [89] to make aggregations and joins at run time without having materialized views. It consists of three types of processing components: cluster leader, router and instance managers. Cluster leader assigns the data to the instance manager, which takes care of multiple tenants. Assignment information is maintained in a cluster map and managed by cluster leader. Cluster map is propagated to the routers and instance managers. Cluster leader also track changes to the cluster state. As a response to the valid request, a change is made in the node for which request is received and later it is asynchronously replicated to other nodes.

A model for defining boundaries of sensitive data and achieving data privacy in Service Oriented Architectures (SOA) is discussed in [91]. The model suggests that services for managing data privacy and protection at run-time should take care of boundaries, ownership and legal requirements regarding the sensitivity of data. The use of separate management services on cloud and data hosting platform is also suggested.

2.4.7. Managing Distributed Resources

The studies [90, 92, 93] describe techniques for utilizing distributed cloud resources. Google has developed a Map Reduced programming framework for supporting parallel computing and access to the database [92]. Client side execution, caching, pre-fetching of data, paging scheme for faster data retrieval, environmental aware evolution and adaptation, autonomous handling of scheduling, prioritization, safety and security issues, design of product services, design of business cases and considerations for service delivery platform are key features of architecture for real time and enterprise service oriented cloud computing [90].

2.4.8. Risks in Cloud-Enabled Systems

Risks associated with cloud computing are grouped into four categories: operational risks, contingent risks, security risks and business risks [94]. Operational risks deal

with availability, reliability, integrity of services and data, service provision according to users' requirements and maintainability. Contingent risks address service and data survival, impact of major service interruption, frequency of interruption and resilience, as well as compatibility and flexibility. Security risks include service and data security, authentication and authorization, and susceptibility to denial of service attacks. Business risks involve cost, unsatisfied customer service by cloud providers, privacy breach, and legal conflict of application requirements with how infrastructure is maintained by cloud providers.

2.4.9. Resource Scalability

An approach to achieve pre-specified scalability of cloud-enabled application and mash-up services using resources pooling from multiple clouds is presented in [95]. System level schedulers and load dispatchers are used for sharing load between services deployed on cluster of clouds. Some other system services are suggested for providing enhanced availability of the system. These services include fault identification services to determine services failures and possible cause of failure, actuator services for finding remedies of the failures, proxies for providing encapsulation and orchestration, and modules taking care of adaptability of partially mismatching services.

Scalability and resource provisioning can also be achieved by defining elasticity rules and load balancing algorithms [96, 97]. It requires explicit declaration of rules for resource allocation, replication, migration and decomposition of composite services as well as each individual service inside composite ones. The language for describing constrains on architecture for cloud computing infrastructure that can deal with resource allocation and implementation of service life cycle management process is also discussed in [96]. A dynamic scaling algorithm monitors the performance of application instances and launches new instances on pre-configured virtual machine images. It also handles shutting down of running instances if load is reduced. It considers number of concurrent users, number of active connections, number of requests per second and average response time per second are important scalability indicators for monitoring the scalability requirements of the services and application on cloud [97]. However, these studies do not describe applicability of this technique on PaaS environment where users don't have specify scalability rules for aggregated or composed services.

2.4.10. Reliability

Reliability deals with system functioning according to the desired behavior. It is measured in terms of systems availability and correctness of the results produced. It is more complicated to ensure reliability on cloud enables systems. These systems are often built on top of infrastructure and services provided by external providers. If any of the services fails to meet reliability standards, it will impact whole system. In order to cope with this challenge, the system should be able to dynamically find replacement services in case any of existing services fails to meet reliability conditions. In order to have risk free collaboration, services build using cloud infrastructure also requires assurance for reliability. An insurance based reliability scheme is introduced in [98] for achieving this quality attribute. Insurance agent supports matching between service provider and service consumer. It also guarantees that service providers offer services according to QoS requirements by charging insurance fee from each of the service provider in case they fail to meet SLA. A mathematical model to predict bankruptcy risk is also presented. According to this

scheme, aggregated service provider is responsible for reliability of all the composed services and it does not take into account reliability of composed services.

2.4.11. Service Adaptability

It allows services to be deployed on heterogeneous cloud environments and with self-organizing deployment mechanisms services can manage resource provisioning without human administration. Services build using REST architecture style is easily adoptable because of their stateless nature. The term liquid services is used in [99] for adaptive and transparently deployable services. In the proposed model, every resource (reactor) can be a service provider as well as service consumer. Web service application has one entry reactor to accept requests from the external clients and to provide the final results. It further categorizes reactors into two types, managed reactors (whose business logic is maintained within the application) and unmanaged reactors (proxies to external third party RESTful web services used by managed reactors to provide final services). The approach presented in [100] uses service mediation and negotiation techniques to handle environmental changes and system failures. However, both of these approaches neither discuss cross platform adaptability nor describe how services can be ported from one environment to other when they are live.

2.5. Discussion

It is described in the beginning of this chapter that the aim of literature review is to find out answers of the research questions 1, 2, 3 and 4. After reviewing the relevant literature, we are able to find the answers in sufficient detail that can help to derive inspiration for forming requirements and designing architecture of the middleware platform.

Regarding questions 1, the review of the literature has identified that applications and tools to support GSD are implemented in different forms ranging from stand-alone applications to plug-in for integrated environments. This highlighted that middleware should support its interface in a platform neutral fashion that are accessible by all types of application. The important characteristics that middle should incorporate are support for collaboration, traceability, explicit support for integration between different types of information, and capability to address needs of virtual teams. The literature reviewed to find answer of question 2 suggests the importance of explicitly taking care of software development process by the middleware.

The tools discussed in regard to find answer of question 3 demonstrate that there are selected tools available in form of online IDEs, design tools, text processing software, and platforms to support for software development activities like Jazz Platform and IBM rational suite. The inability of the cloud based tools to integrate with other tools in order to support full life cycle of software development life cycle make them less effective to meet needs of complex GSD activities. The investment and effort required adapting to platform like Jazz Platform and IBM rational suit make them less feasible for small and medium sized organization. Hence, there is a need for a new paradigm that is only easy to adopt but also cost effective for small and medium sized organization with limited resource. After reviewing the literature to get in depth knowledge and answer of question 4, we have come up with large number characteristics for cloud-enabled systems. More important of these characteristics are support for multi-tenancy, ability of the applications to server different types of client

devices, workflow support, managing distributed resources, scalability and reliability of the resource.

The knowledge gained from the review of literature is used to come with the requirements of the platform elaborated in Chapter 3. The review of the literature on cloud computing has provided us detailed knowledge about important characteristics and architecture solutions to incorporate these characteristics in cloud-enabled systems. We have utilized this knowledge in designing architecture of the PTaaS described in Chapter 4.

3. PTaaS Requirements

In this chapter, we give a brief description of the requirements that we intend to incorporate while defining the reference architecture of the platform. Although, we already have discussed the relevant literature in detail in Chapter 2; we will discuss some important aspects in a bit more detail to lay foundation for the requirements.

There are few studies attempting to describe the role of cloud computing and virtualization technologies in the Global Software Development [21, 22, 101, 102]. Use of virtualization technologies to bundle set of tools in a virtualized environment is demonstrated in [101]. Hashmi et al. [21] have described how GSD can take advantage of the cloud computing offerings to address geographical and temporal issues. These issues emerge as a result of distance between distributed teams, lack of knowledge transfer, limited access to unified set of tools, less visibility of the activities across other sites and a risk of losing project artifacts. Introducing cloud and service computing in the context of GSD can make it possible to achieve dynamic and runtime binding of the services and their availability to the end users on demand [21, 102]. It also reduces overhead of manual configuration of development environments on each site involved in GSD. By utilizing scalable cloud storage and computing resources, organizations can host their data and services on centralized locations and make them available to the end users [21].

In our previous work [22], we have investigated concept of offering software engineering applications and tools as services on cloud. We have provided a high level overview of the advantages that can be achieved by the offering. The advantages include: no obligations to fixed license fees, possibility of aligning tools with software development processes, tractability between artifacts maintained on different distributed sites, ability to work with huge volumes of sensitive data, and support for awareness & collaboration. Some of the quality attributes that are important to be achieved in such an infrastructure include: support for large number of organizations following different processes, maintaining different version of the applications & services, composing them into a unified solution, inter tool compatibility, providing secure access to sensitive data, ability to consider need of multiple types of end user devices and compliance with end user service level agreements.

Martignoni [102] has argued that lack of transparency in traditional software development and management tools may arise problems when used for GSD. He has emphasized the need for tools and services that can align business and technical issues. He has proposed a model to combine functional and strategic dimensions together to support and achieve optimal technical development strategy and minimizing project risks. Using SaaS model in GSD can enable tools offerings from a centralized location and help external collaborators to have access to same development environments as used by teams who are working on development of projects. Apart from providing support to meet technical challenges, centralized offerings of GSD tools and services can also enable organizations to develop best practices by having a complete insight into software development and management activities.

Richardson et al. [103] formalize the concept of *Virtual Teams* in GSD as described in chapter 2. Although virtual teams work to achieve shared objectives in the same way as traditional teams, but their placement on different geographic locations and working from different time zones make them different because communication takes place through electronic communication channels rather than face-to-face meetings. They have emphasized importance of collaboration between teams to maintain work product ownership boundaries among interfacing locations, define and maintain processes to exchange data, and exchange of actual artifacts that are either consumed by sites or produced by sites. Awareness and visibility of activities among sites is also an important factor that plays a role in more fruitful GSD environment.

Based on the finding from question 1, 2 and 3 explored in Chapter 2 and points discussed above; we provide brief description of the platform requirements in following subsections. The requirements are derived by taking inspirations from the discussed literature and our vision of the platform for providing GSD applications, tools and services. We have divide requirements into two main parts: features of the platform and quality requirements the middleware platform needs to satisfy.

3.1. Requirements for Platform Features

R1: Providing Support for Applications of Different Types

We have discussed in chapter 2 that applications for supporting software development activities can exist in many forms including stand alone application, integrated development environments, middleware platforms and plug-in. For the envisioned platform to be able to support this heterogeneous types of applications, it should expose its interfaces in form of platform neutral APIs so that the tools can access platform's features seamlessly.

R2: Authentication and Authorization

One of the primary requirements of platform to provide end-to-end solution for GSD applications, tools and services is to manage users and tenants on the platform. In the context of our middleware platform, we consider tenants as a group of users with a specific set of preferences. A tenant may represent all stakeholders involved in the GSD project, a project site, or an individual team on a distributed site. Management of tenants and users by the platform is also needed to provide single authentication mechanisms when application, tools and services developed by multiple vendors integrate with each other. If we do not provide a uniform authentication scheme through a single point, it is not possible to combine different set of applications, tools and services according to their features and provide them to end users as a unified solution.

R3: Repository Management

For materializing concept of offering software development application and tools as services, first and one of the foremost requirements is to manage the repository of applications, tools and their corresponding services so that they can be bundles and provisioned according to the need.

R4: Integration

Our goal to provide the platform for offering software development tools and applications as services in context of GSD is to provide an end-to-end solution to the

organizations that are participating in a globally distributed project. Hence, the platforms should be able to capture tenant's requirements, and use the requirements to bundle tools and applications in a way that the bundled package supports all the necessary requirements.

When application, tools and individual components are to be bundled as a unified solution, the platform should support a mechanism to resolve interoperability issues and facilitate data exchange between individual elements of bundled package. The inter operability solution should not only provide support to exchange data in different formats but also should support different communications protocols.

R5: Composition

Software development activities are specific because they often involve more than one tool to carry out development task. The required tools and applications are dependent upon each other and cannot be used in isolation. For example, during development of web applications, IDEs often need to be configured with application server so that unit testing can be performed. The platform should provide features and interfaces so that tools dependencies can be specified.

Compatibility between the applications, tools and individual services should also be considered when they are composed as a unified solution. To cater this requirement, the platform should have feature matching and compatibility support so that the desired configuration can be composed by combining compatible individual elements and satisfying users' requirements.

R6: Alignment of Tools with Processes

To materialize the concept of offering software development applications and tools as services, it is important that the platform is able to capture the process that organizations involved in the GSD activity are going to follow and map tools on process accordingly. To accomplish this feature, the platform needs to consider different stages of the process, tenant participating in each stage, which tools are going to be associated with each stage, how tenants from one stage of the process are going to have access to produced artifacts, how tenants from different stages of the process will have access to produced artifacts and how the artifacts can be managed on cloud so that they satisfy data privacy requirements of the tenants. When large numbers of software are working in a workflow like approach, the platform also needs to ensure that data is exchanged between software in proper order.

R7: Support for Collaboration and Traceability

Collaboration among teams is a primary concern in GSD. The published literature refers collaboration as interaction between teams members to facilitate them in carrying out their daily tasks as discussed in chapter 2. This interaction takes place through the use of electronic medium. In context of software development application and tools as services, collaboration takes an additional dimension. The set of tools provisioned for a particular project also need to collaborate with each other in order to exchange data. Similarly the platform should also support traceability among artifacts that are maintained inside heterogeneous tools so that other tools can be notified whenever the artifacts are modified.

R8: Supporting Virtual Teams

The platform should provide appropriate support for virtual teams and keep virtual team management aligned with processes. It is necessary to capture concept of virtual

teams at the platform so that appropriate persons can be assigned to different parts of the software development process and only assigned persons can have access to the provisioned tools and engage in collaboration activities.

3.2. Quality Requirements

Other than incorporating functional requirements, the cloud platform also needs to incorporate some non-functional requirements. This section provide a description of the quality concerns that important from the point of view of GSD application and tools as services.

QR1: Compliance with location and multi-tenancy constraints

Location of data and services, and sharing of services among multiple tenants are primary concerns in multi-tenant cloud environments. This concern becomes prevalent in GSD context because of involvement of multiple stakeholders. The stakeholders may have different privacy concern. Therefore the platform need not only capture these concerns but also provision application, tools and individual components/services according to preferences.

The provisioning of applications and tools should be done according to multi-tenancy and location constrains specified by the users. Multi-tenancy constrains can be specified as shared or exclusive. If tenants have opted for exclusive usage, separate instances of applications, tools and services are to be enacted for the tenant. If tenants have opted for shared usage, they can be granted access to applications instances that are shared among other users as well. For shared usage, application users have to rely on multi-tenancy support provide by applications and tools. Tenants can also specify location constraints on application and tools, and these are provisioned on the underlying IaaS cloud on the specified locations (e.g. US Region or EU region) provided that IaaS provides hosting support on the regions.

QR2: Awareness of Context of the Client Devices

The clients of the applications and tools may access the hosted applications from different types of devices. Also the tools and services may have different flavors of the services for different types of client devices. The platform should be able to manage different flavors of applications and tools as well as should be able to identify context of the client devices and manage requests accordingly.

QR3: Lifecycle Management

The applications and tools are hosted on the underlying IaaS cloud when they are enacted and provisioned. The platform must have capabilities to manage resource provisioning from the underlying IaaS cloud and impose constrains on the IaaS that ensure smooth functioning of the applications, tools and associated components/services.

3.3. Scenario Analysis

To better explains the potential benefits that can be achieved by having a platform that can support software development applications and tools as services, and to clarify the requirements elaborated in section 3.1; we are providing a concrete scenario. The scenario covers potential benefits that can be achieved by having the platform and why it is important to incorporate the specified requirements.

The hypothetical scenario describes enhancement in the software named ShipDesigner dealing with core business of the company named XBuilder. XBuilder is a Scandinavian company having expertise in design ships. The company has its offices located in two Scandinavian countries Denmark and Sweden. The software development company named DesignerSoft had been developing software in the past. Before they enhancement project was initiated, DesignSoft had reduced their workforce in Denmark because of some organizational restructuring. As a result, when they have received enhancement request they do not have required workforce to allocate on the project. To meet this challenge they have decided to introduce global software development in their company.

After initial analysis of the potential companies around the world, they have identified that Vietnam and China are potential countries where they can find partners organizations with required expertise and workers with proper domain knowledge. However, DesignSoft has to consider a number of factors before they can engage in partnership with Chinese and Vietnamese companies. The first and foremost challenge that they encounter is constraints on project source code and test bed of data that XBuilder has provided to DesignSoft. XBuilder does not want its complete source code to be provided to development firms in China and Vietnam because they are afraid that their competing organizations can have access to it and they may lose their competitive advantage over them. Second concern they have is about movement of test data drawings. Some of the test data drawings are of sensitive nature and contain designs of some Danish and Swedish war ships. They are under obligations by both Danish and Swedish government to not let the data go out of European Union.

DesignSoft has decided to introduce cloud in their company for the first time to address this situation. They have set up a small private cloud to store private and sensitive ship drawings on their premises to avoid a risk of their theft or loss. To store the non sensitive data and they have acquired resources of Amazon's European Cloud offerings. This setup helps them to address the issues of data privacy and risk of theft. The other part of the problem remains unsolved, i.e. how to enable offshore software development firms from China and Vietnam to have access to data and do development on the project without actually handing over project's code to them.

Software architects and engineering at DesignSoft have decided to exploit few other features of cloud to solve their problem. They have decided to configure virtual machines on Amazon European cloud that developers from China and Vietnam can access to perform their development activities. They have configured the virtual machines in a way they developers from remote sites cannot transfer code and related artifacts to other sources. However in order to get the code compiled and integrate with main code base, they need to transfer individually developed code artifacts from virtual machines and place them on a place where they can be effectively integrated and easily compiled. The developers may need to compile code several time during development to check errors, perform unit testing and see the results of their code. Hence software engineering at DesignSoft has written glue code components that can fetch code from the virtual machines and deploy that on a cluster of scalable Amazon European cloud so that large number of requests can be catered without introducing any unnecessary delay. They have also written applications to display compilation results to the developers. As virtual machines also hosted on Amazon European cloud

where source code is being compiled, hence the data propagation delays are not significant high. Facilitate testing on using private data; DesignSoft has deployed their testing environments and virtual machines on private cloud enabling testers from remote site to use real data and enabling testing on all possible scenarios.

Data transmission schemes are managed by implementing a process bus on Amazon and DesignSoft's private cloud that ensures artifacts are posted on the corresponding services (for example for compilation of code artifacts) and results are sent to intended destination (compilation result of the code is returned to the virtual machines of the developer who placed a compilation request). DesignSoft is able to meet to the requirements of their current development phase by exploiting cloud-computing features. This setup has enabled DesignSoft to take benefits of the GSD without imposing any risk on their business and assets of their clients. This scenario depicts how we can cope with complex challenged of GSD by utilizing cloud computing which otherwise would have been difficult to address.

4. Architecture of PTaaS

This chapter describes the reference architecture of the PTaaS (middleware Platform for providing software development applications and Tools As A Service). The reference architecture is designed following Service Oriented Architecture and REST principles [104]. SOA approach is adopted to make it easy to replace application services with vendor specific implementations. In order to provide the reference architecture for providing software development applications as services we have taken inspiration from available literature on software architecting approaches for cloud enabled systems as discussed in section 2.4 and have tailored the approaches according to requirements for our platform.

Figure 1 shows an abstract view of middleware platform and elaborates how it fits into concept of providing application as services in GSD context. It is elaborated earlier that GSD is characterized by involvement of stakeholders in application development from geographically distributed locations. Figure 1 shows hypothetical scenario representing stakeholders from four different locations involved in a software development process. There are two sites hosting development teams and one site hosting testing team. Figure 1 also shows a fourth site representing main office responsible for designing application architecture, managing development operations and interaction with clients.

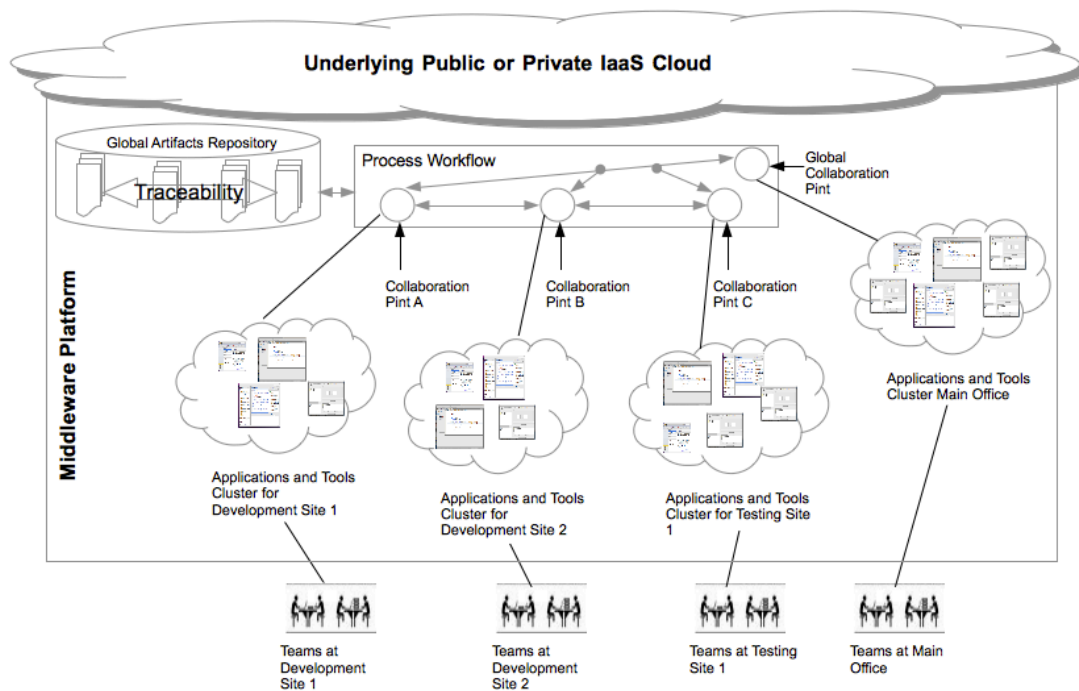


Figure 1: Context of Middleware Platform

When a new project is kicked off, management team and architects defines development process in the middleware defining nodes to represent each development site. The management team also specifies that how the artifacts would be shared among different sites or how ownership of the artifacts would be changes from one site to another. The stakeholders from the development sites request middleware with their requirement for applications needed at each site. The middleware platform

bundles applications as required by tenants at each site and provision them on cloud according to their requirements. Middleware also attach bundled applications with the nodes in process that represent each unit involved in the development process. The unit can be an organization involved in GSD, a development site of a GSD project or a department at a development site. Users at each unit can request applications suite in multiple forms. They can request for applications that would be shared among all the users in a development site (e.g. web based applications or specific services), they can request individual tools (e.g. development IDEs and applications servers) hosted inside virtual machines for a specific user or they can request an application that will be used by stakeholders from all sites (e.g. bug tracking system or requirements management application). The traceability between artifacts is provided through platform (applications needs to implement compatible interfaces to support traceability) among application provisioned within a site and across multiple sites. Changes in data ownerships and data transmissions (in case data need to be transmitted from application to another) are also managed by platform with the help of collaboration points.

Figure 2 shows block diagram of the subsystem making middleware platform for supporting applications and tools provisioning as services. The middleware is designed as a layered architecture in which each subsystem provides different functionalities. Each layer of the middleware is explained in subsequent subsections. First, the interface layer is described which provide support for having access to system resources and applications from multiple types of devices. Second, role of accounting services is described. Third, management layer is explained. This layer is responsible for management of applications, tools and their corresponding services on cloud. This layer also takes care of provisioning of the applications and associated resources on cloud according to the tenants' location and multi-tenancy specifications. After that fourth layer is described that is responsible for management of data during collaboration activities and providing authentication support for users when multiple applications and services are bundled together. Fifth and sixth layers of the

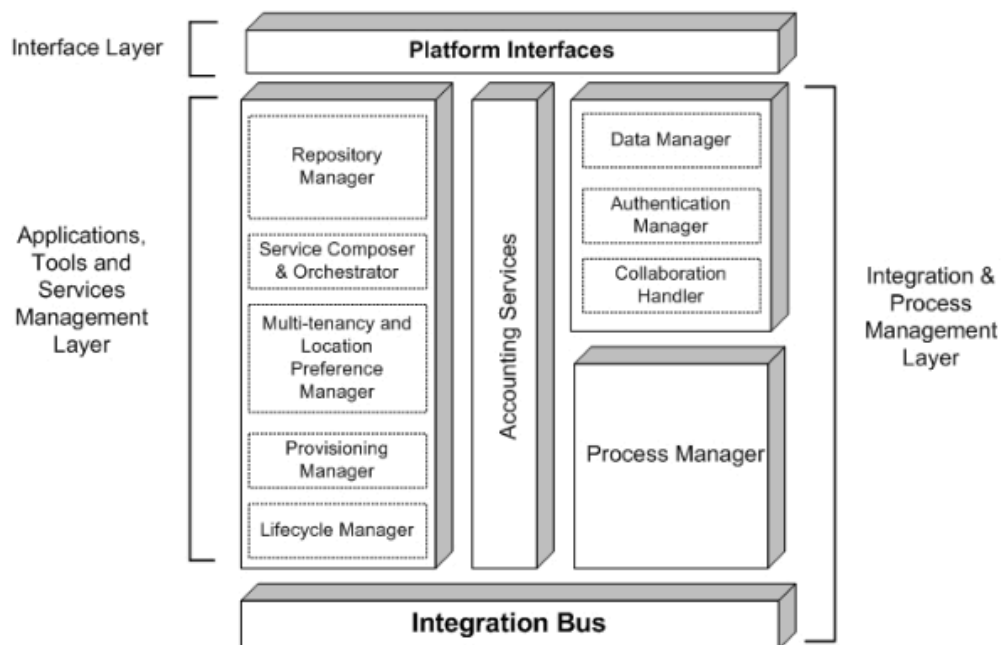


Figure 2: Subsystems making up Middleware Platform

middleware platform deals with management of development process and integration of application and services when bundled together to work as a unified application suit. We are treating different components inside each layer of the reference architecture as services so that they can easily be deployed on heterogeneous cloud environments without imposing any additional constraints on their configurations in context of a particular cloud environment.

4.1. Interface Layer

The Interface Layer is the top layer of the platform. It provides support for applications to handle request from multiple types of devices, routing of requests to the appropriate clients and adjust cloud resources according to the specific needs of clients. The ability of the cloud environment to support adjustment of application resources on cloud makes it easy for applications to support multiple types of client devices [13, 67]. Especially for mobile devices, limited processing and storage power can be balanced by automated, dynamic and reliable runtime configuration of components on cloud to perform resource hungry operations on clouds and sending them results to client devices in compatible formats [13, 68]. Spatial and context awareness features of mobile devices can also be helpful for many GSD applications to support awareness inside application [67-69]. Applications specific components on the clouds need to be adjusted according to perform complex tasks when different types of devices are accessing same application through unified interfaces [70, 71]. To provide these features on the cloud platform, we are presenting architecture style described in Figure 3. The presented architecture style is only focusing on platform specific features for supporting multiple types of devices (mobiles, tablets as well as PC variants). Applications can implement their internal processing components according to their specific requirements and can keep this information abstract from the platform.

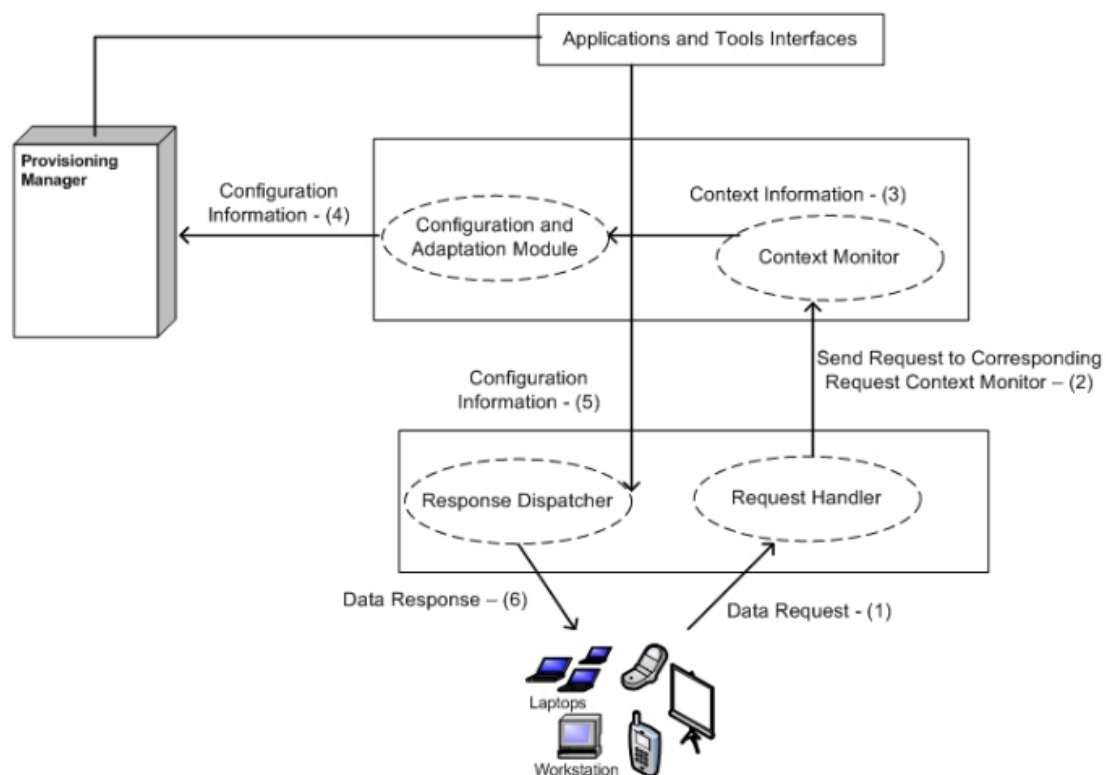


Figure 3: Interface Layer

We have taken inspirations from the studies [13, 67-71] and concepts presented in them, and have applied the concepts to propose this reference architecture style. The description of the different services of the reference architecture style is as follows.

The request from client devices is received by *Request Handler* service. This service monitors the request and forwards it to the corresponding *Context Monitor* that is capable to parse the request and fetch context information out of it. This context information is then passed to the *Configuration and Adaptation* service that determines the optimal service configuration and sends this information to the *Provisioning Manager* inside Management Layer. Internal functioning of the Provisioning Manager is provided in section 4.3. For first request from a specific kind of devices *Provisioning Manager* configures application services according to the configuration information. To serve future requests from the same types of devices, service requests are directly transferred to the corresponding services. *Response Dispatcher* use context information of the device to transforms the response recognizable for the client device and dispatches it.

4.2. Accounting Services

These services are responsible to record utilizations of the resources and act as a logger of the resource utilizations by tenants and individual users. This information is used by the platform to compute user and tenants specific billing. Applications and tools hosted on cloud that want smart resource accounting needs to call these services in order to have billing details of each user and tenant. Accounting services are needed because platform enacts applications and tools at run time and information of how much actual cost is incurred when resources are utilized cannot be captures outside the platform.

4.3. Applications, Tools and Services Management Layer

Management layer encapsulates intermediate services responsible for composition of applications into a suit, their provision on the underlying cloud resources and monitoring of services once they are deployed and running on the cloud.

The platform support for reliable and autonomous composition and deployment of services is a key feature for providing applications suits as services for GSD teams. The importance of having middleware support for autonomous service acquisition is reported in numerous studies [13, 72, 73, 76, 77]. Autonomous management as a part of cloud middleware platform to support provisioning of GSD tools as services is important from two perspectives. First, as applications hosted on cloud can serve a large number of tenants each of them may have specific requirements for a particular version of the software [78]. In GSD context, there may be large number of tenants requesting for multiple tools with their specific version requirements. It is impossible to manage all the requests through manual administration; hence the platform needs to provide automated support for selecting applications and services according to their specific versions. The complexity increase multiple times when multiple applications and tools are bundles together to be provisioned as a suit. Moreover, when it is required to host data on secure private cloud, external application and service should have only selected access to the data [79]. In short, the platform needs to support following features for managing applications and associated services.

- Handle autonomous composition of application and services according to tenants' specification [13, 72, 73].

- Separate application and service management and distribution strategy from business logic of the applications and services [74, 75].
- Version management of applications and services to maintain separation between different components of the applications and their version specific composition according to the tenant's specification.
- Trusted services that act as proxies of the secure data services hosted on private cloud [79].

In the following subsection, we provide description of the main components of the applications, tools and services *Management Layer* shown in Figure 2. The reference architecture styles and their services are also described.

4.3.1. Repository Manager

This service takes care of registration of applications and tools associated with GSD, and their services with the middleware platform. The applications, tools and services can be registered in two forms.

- By registering their executables, version, list of features, deployment scripts and information about hosting platforms (application servers, operations system, network requirements).
- By registering the unique identity of the preconfigured virtual machine template that can be provisioned to make an application or tool available for use.

The applications and tools are classified into different categories. E.g. integrated development environments (IDEs), design tools, requirements engineering tools etc. Whenever applications and tools are registered in the platform, their category is also specified. Inside each category, applications have sub categories. For example, sub categories for IDE are Eclipse, NetBeans etc. Sub categories of design tools are ArgoUML, Visio etc.

4.3.2. Multi-tenancy and Location Manager

This service takes care of tenants' locations and multi-tenancy preferences and facilitates service provisioning according to their specification by collaborating with *Initialization and Realization* module of provisioning management sub-system. Multi-tenancy specifications have two levels in the platform, shared or exclusive. If tenants have specific shared level, then if applications and tools are already enacted, tenants are granted access to them and these applications and tools as well as their compose services are shared among multiple tenants. If exclusive option is selected, then new instances of applications and tools are enacted with a specific tenant as an exclusive user of the application and tool instance.

Tenants can specify location constrains so that applications and tools are hosted on underlying IaaS resources on the specified location. If location constraints are defined, then applications and tools along with their composed services are enacted on the specified locations provided that underlying IaaS cloud supports location specific enactment.

4.3.3. Life Cycle and Provisioning Manager

This set of services takes care of deployment of application, services and tools on underlying cloud infrastructure, enact them and make them available for end users. This service is composed of multiple sub service. Tenant specific location constraints and elasticity rules are stored in *Life Cycle & Scaling Rules Repository*. Whenever a new request is received for provision, its life cycle and scalability rules are retrieved

from the *Life Cycle & Scaling Rules Repository*. These rules are parsed by *Parser* and are structured so that *Interpreter* can convert them into action scripts and pass it on to *Initialization & Realization Module* so that applications and services can be provisioned according to the location and privacy constraints. Once applications and services are deployed on the underlying IaaS cloud, these are continuously monitored by *Monitor* to see if services performing according to performance and privacy constraints. Monitor takes input from *Performance Parameter Manager* for required performance requirements. Monitoring report is send to *Analyzer*. *Analyzer* analyzes the monitoring report and metrics for inconsistencies. If any inconsistency is detected, analysis report is passed on to the *Planner* service, which makes a new execution plan and pass it on to the *Adaptation Module*. The adaptation module then adjusts the resource provisioning according to the new execution plan.

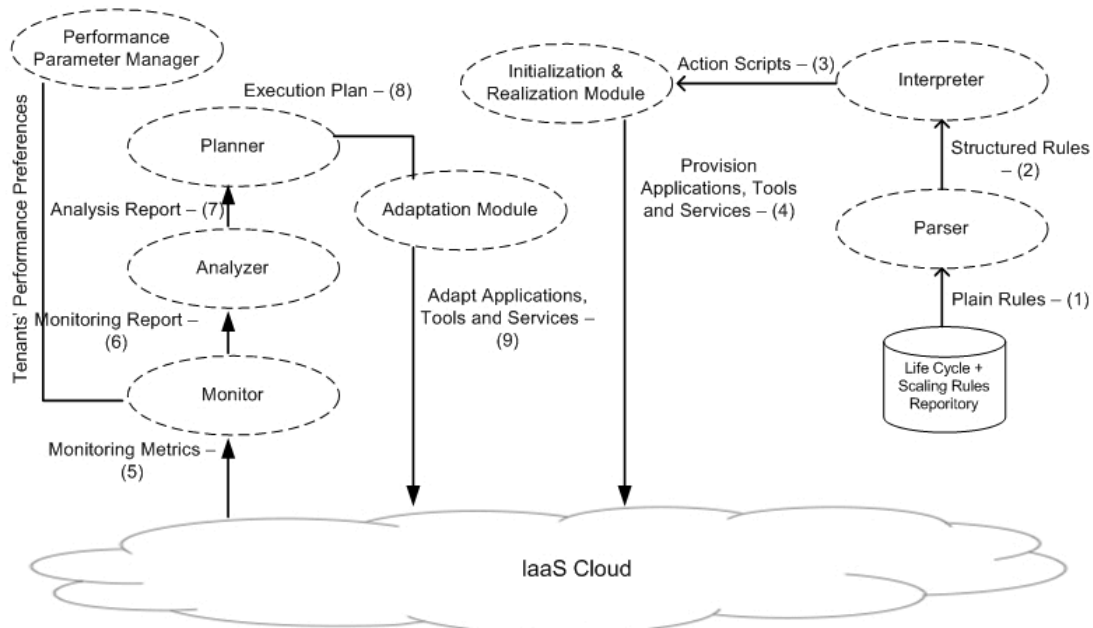


Figure 4: Provisioning Management

4.3.4. Service Composer and Orchestrator

The set of services included in this module performs composition of applications, tools and their respective services. The module also takes care of applications and services placement on the cloud and provides integration support. Graphical representation of the composition mechanism is shown in Figure 5. For each request of applications and tools provisioning, they are bundled together in form of a tool suite that is capable of handling end-to-end operations against a particular task. Each **tool suite** has a **distribution scheme** that determines that how individual tools inside suit should be deployed on cloud and their mapping on the software development process. A **tool suite** can consist of one or more **applications and/or tools**. **Applications and tools** are connected to each other through connectors. Each **tool** may has one or more **services** inside it. A **service** can either be **scalar** or can be **composed** of multiple **sub-services**. **Applications, tools and services** have their deployment information associated with them. This information is used by the platform to deploy for their provisioning on underlying cloud environment. Applications and tools are connected to each other through **connectors**. Services inside applications and tools have **properties** containing any additional information needed for service enactment and management.

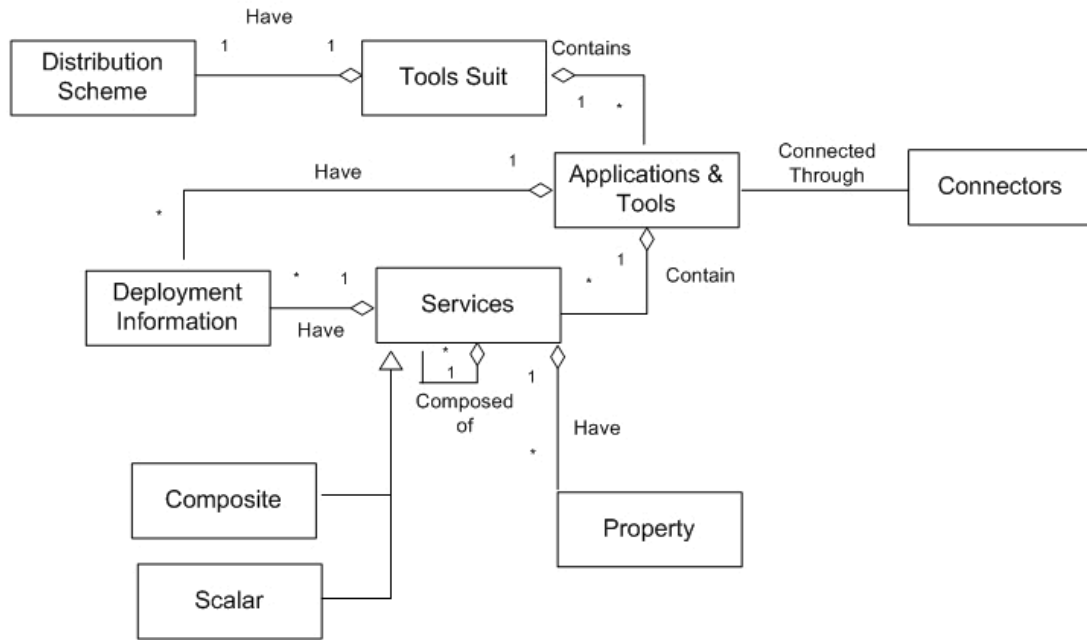


Figure 5: Composition Scheme

The overview of the how the composition works and services involved in the process are shown in Figure 6. The information of the applications, tools and services available for provisioning is stored in “**Application, Tools and Services Resource Pool**” the as described in section 4.3.1. Services **composer** compose them according to requirements of tools as specified by tenant. **Distribution Manager** interacts with **Provisioning Manager** (section 4.3.3) to deploy applications (or tools) and associated services on underlying cloud infrastructure. Data transfer manager established interaction points on the **Integration Bus** and through which bundled applications, tools and services interacts with each other. Details on how the interaction points are defined and how the communication between services takes place are described in section 4.3.5.

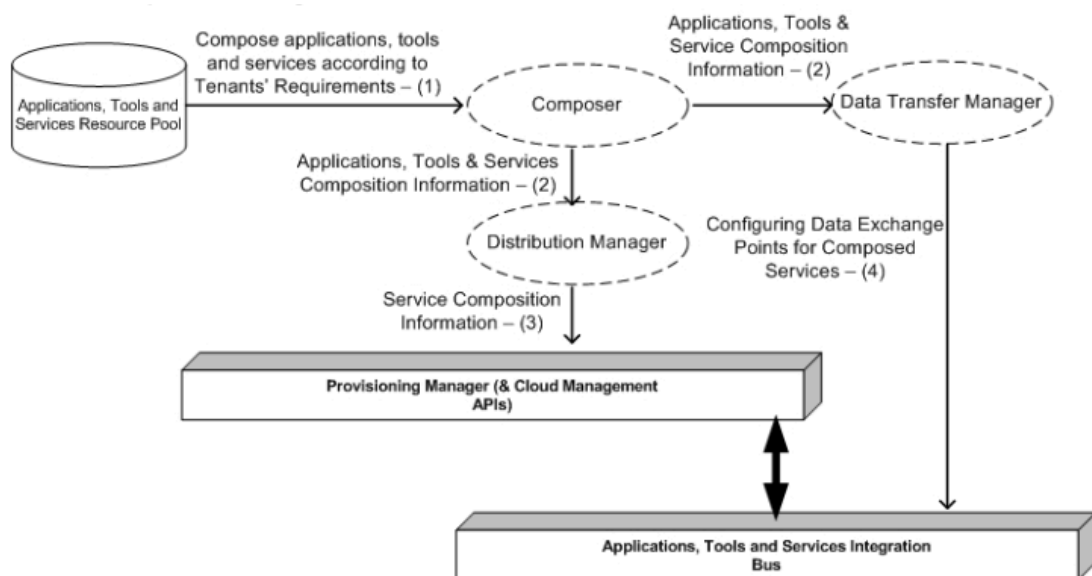


Figure 6: Composition Handling

4.3.5. Authentication and Process Manager

When applications, tools and services are provisioned at run time: there is a need to provide a unified mechanism for authentication and authorization of services. **Authentication Manager** takes care of it. All the application, tools and services that need to authentication users should interact with this service in order to verify end users credentials.

It is discuss in introductory part of the thesis and in chapter 2 that software development incorporates multiple activities. These activities are streamlined by defining a software development process. Hence, a cloud platform for supporting software development activities in general and GSD activities in particular, should be able to define and manage a software development process. Moreover, it should also take care of applications, tools and services provisioning according to the process defined for development activities.

There are few studies that are explicitly discussing strategies for pipeline based data processing on cloud [81, 83, 84]. The studies focus on three aspects:

- Processing of the data on independent nodes.
- Re-collection or re-projection of data to produce results.
- Scheduling of tasks to be distributed on different nodes.

Apart from this, when data pipelines are distributed across a hybrid cloud the pipelines also needs to consider:

- The track of data that is produced by different processing nodes and how it is propagated across nodes [80].
- In case of secure pipeline, the processing nodes need to know only about their immediate down stream node (the node from which it receive data) and upstream node (to node to which it sends data) to avoid exposing whole pipeline [80].
- When processing pipeline is distributed on a federated cloud, the processing nodes should be distributed according to the data sensitivity requirements [82].

Figure 7 describes how authentication and process management works. The processing pipeline of the platform described in figure takes care of all the features described above for pipeline styled process. The process pipeline also encapsulates **Integration Buss**. **Integration Bus** supports integration among different types of applications and tools attach with each node of the process pipeline with the help of collaboration and traceability points.

Development Process Handler takes care of initializes the process on pipeline. The process consists of nodes on which applications/tools can be attached. A node can represent a developer machine, a team, a development site involved in GSD project or a whole organization participating in GSD project. It also consists of workflow specifying the sequence of nodes in the development process (e.g. which node is associated with design tools, which node is associated with development tool, which node is producer of artifacts, which node is consumer of artifacts etc.). Realization of the nodes on cloud infrastructure is done with the help of Provisioning Manager (Section 4.3.3) according to the requirements of the respective tenants. Each node of the workflow is associated with one or more collaboration & traceability points represented as CP in Figure 7. These points facilitates inter application/tool collaboration and traceability. **Collaboration and Traceability Handler** manages collaboration and traceability points. It also takes care of how data will be transmitted from one point to next one depending upon how the workflow is created. Each application/tool can have one or more **Re-projection Components/Services**. These

components/services takes care of merging data artifacts together when artifacts are processed at more than one node. Other than Re-projection Services, additional services can also be attached with traceability and collaboration points that are invoked whenever data is posted on or retrieved from them. **Workspace** represents a virtual abstraction of the location involved in a GSD process. Each workspace has applications and services associated with it, users of the workspace, their roles and activities that users will perform on artifacts using applications/tools. Workspace is composed of four services. **Applications and Tools Association** for managing association of applications and tools with nodes of process workflow. **User Manager** for managing users of applications and tools. **Role Manager** for taking care of users' role while they are using applications and tools to work on software artifacts. **Activity Manager** to manage activities that users perform on artifacts.

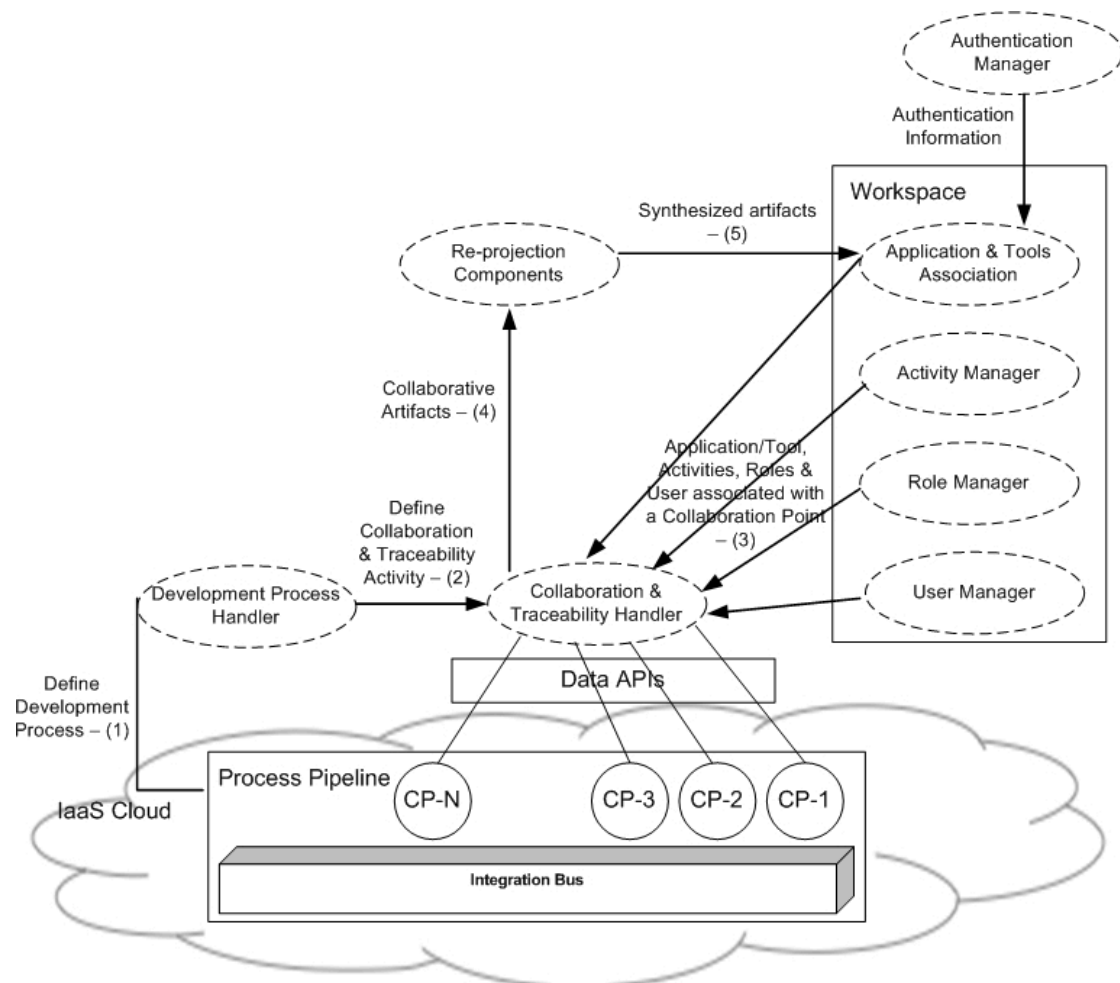


Figure 7: Authentication and Process Management

Collaboration and Traceability Handler supports two types of inter application activities: collaborative work on artifacts from different types of applications and traceability support among artifacts being maintained by different applications. It is discussed in section 4.3.1 that applications and tools are classified into different categories. Each category of application has a set of predefined collaboration types corresponding to features that are supported by the applications. For example, software architecture design tools have a collaboration type associated with the design activity that is supported by the tool; i.e. activity diagram collaboration, use case diagram collaboration, class diagram collaboration, deployment diagram collaboration

etc. Similarly, each category of application also has a set of predefined traceability types corresponding to features that are supported by the applications. Traceability types associated with software design tools are activity diagram traceability, use case diagram traceability, class diagram traceability, deployment diagram traceability etc.

Figure 8 shows collaboration supports with the help of a hypothetical scenario in which two software applications intend to engage in a collaborative task. The collaboration activity can be invoked by any of the participating applications. In the scenario depicted in Figure 8; Application A, as first participant of the collaborative activity invokes collaboration by sending information of the participant including tenants' and users' identities. Each application that wants to utilize collaboration features of the platform can register a call back interface with the platform for each type of collaboration activity. In case call back method cannot be registered (for example in case of desktop based applications and tools), information on collaboration can be found by periodically calling method to retrieve collaboration information. In case when call back methods are registered, the platform notifies the application through registered call back methods whenever there is an update on artifact for an application participating in the collaboration activity. In the scenario depicted in Figure 8, Application B will be notified about the update collaboration activity. In case if there are more than more one applications being involved for a collaboration activity, all of them are notified about the update or alternatively they can ask information about updated by calling collaboration APIs. A collaboration points is established between every two interacting nodes of the process workflow by the middleware platform. The platform maintains intermediate copies of the artifacts inside each collaboration point. An instance of the **Time Stamping Component** is associated with each collaboration point that is used to generate vector time stamps [105] in which each entry of the vector time stamp represents logical time of the user participating in collaborative work. For N applications associated with a collaboration point, vector time stamp takes form $[t_1, t_2, t_3 \dots t_N]$ corresponding to applications $[A_1, A_2, A_3 \dots A_N]$. Whenever application A_i makes an update, t_i is incremented by one. Whenever there is an update on the intermediate copy of the shared artifacts, all the applications participating in the collaboration activity are notified. The notification includes the information of who made the last update (including application id, tenant id and user id), the identity of updated the artifacts, and the time

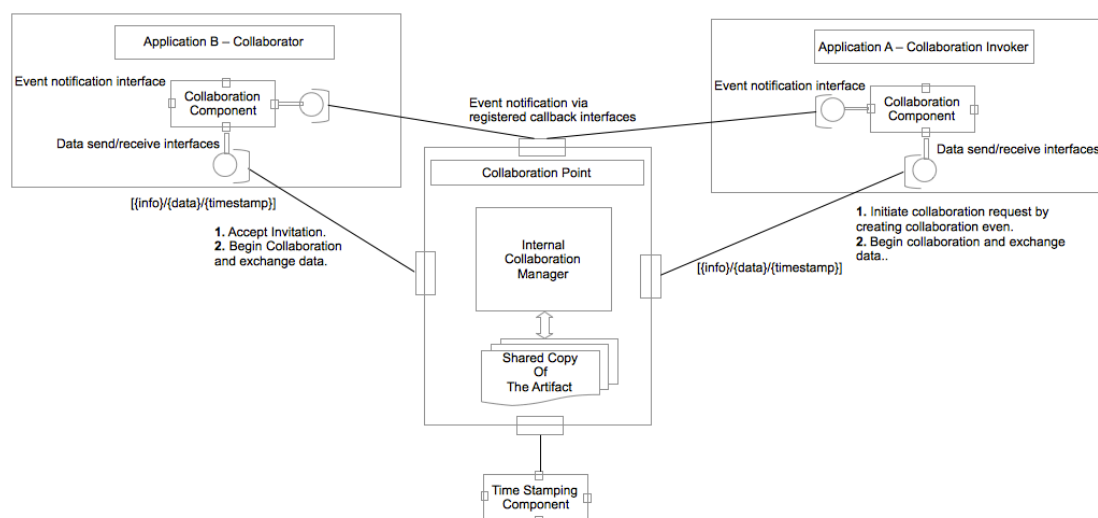


Figure 8: Collaboration Handling

stamp associated with the updated version of the artifact. The data can be fetched from the collaboration point by passing the artifact identify and time stamp by the applications who are set as consumers of the artifacts in the process workflow. If the time stamp provided by application is older than time stamp of the collaboration point, updated artifact is returned. Otherwise an error code is returned. Applications and tools that cannot register callback methods need to call collaboration interfaces periodically in order to find about updates on collaboration activity.

A simple collaboration flow is shown in Figure 9 in which three nodes in the workflow are connected to each other according to waterfall proves model. There is a collaboration point between every two connected nodes. Tools are posting and retrieving information from collaboration points with the help of collaboration handler. Whenever workflow is enacted by the platform, collaboration and traceability points between every two interacting nodes of the process workflow are automatically enacted.

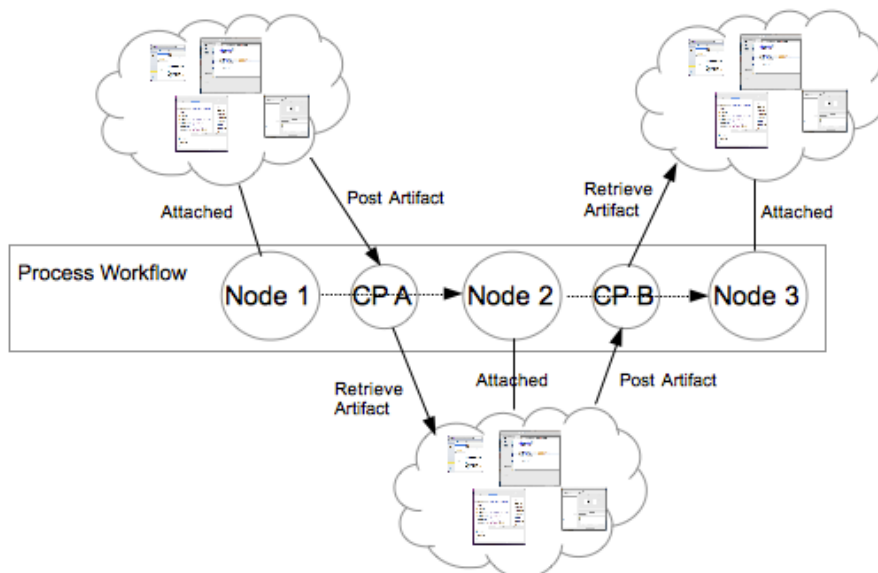


Figure 9: Collaboration Flow

The platform is aimed to provide end-to-end solution for GSD project by bundling applications and tools into a suite. The tools suite may contain multiple tools that are associated with more than one sites involved in a GSD project. When artifacts are being developed across multiple sites using heterogeneous tools, the platform needs to support traceability mechanism so that if there is a change made on an artifact by an application or tool, the information about the change can be propagated to other tools (tools that are working on the artifacts that are linked to or dependent upon the changed artifact). **Collaboration and Traceability Handler** provides following features to support traceability.

- i) Features to acquire unique identity for each artifact that needs to be traceable with other artifacts.
- ii) Features to register call back methods of tool that contain artifacts in form of a tuple [T, A, F] where T is the application/tools that contains artifacts corresponding to identity A and F is the callback method of application/tool T. This step is optional and alternatively, applications and tool can periodically call interfaces for traceability to get updated information.

- iii) Features to register a traceability request. Traceability request can be registered with the system in form of a tuple $[A, (B, C, D)]$. Where A, B, C and D are identities of the artifacts such that if artifact corresponding to identity A is changed; artifacts corresponding to identities B, C and D also need to be updated.
- iv) Interface that applications/tools can call to provide information to the platform whenever the artifact is updated. The information is provided to applications and tools in form of tuple $[A, D]$ where A is the artifact id and D is the data describing information of the updates.

When the platform is informed about the updates in any artifact, it calls the callback methods of the dependent tools if these are registered. Time stamp associated with artifact is incremented with every update. The platform keeps record the artifacts that are linked with each other traceability along with tools that are maintaining the artifacts as described in (ii) and (iii). By using this information, the platform send information to the tools that need to be informed about the update in the artifact by using their registered call back methods. The data containing information about the update is passed as it is by using the callback methods. If callback methods are not registered, client applications and tools can get this timestamp to compare with their locally stored time stamp to check if any update is made and get the traceability information. Processing of the data is left on the applications/tools because each application may need to process it according to their specific requirements and it is not feasible to capture all processing scenarios in the platform. However, Intermediate services can be attached with traceability points and these services can perform desired computation on the artifacts.

5. Proof of Concept

In this chapter, we provide overview of the prototype implementation of the reference architecture elaborated in chapter 4, analysis of the main features of the platform and design decisions that are made to support them. In the prototype implementation of the reference architecture elaborated in chapter 4, we are focusing on the core functionality of the middleware platform; i.e. defining the development process, specifying different development sites in the development process, assigning applications and tools with each development site, assigning tenants to different development sites (nodes of the development process), defining artifacts flow sequence between nodes, assigning additional services to be called when artifacts are exchanged between nodes, establishing collaboration and traceability points, and enactment of the all these involved application, tools services on the underlying cloud infrastructure. We are elaborating implementation details using 4+1 view model of software architecture description as explained in [106].

In first subsection we architecture description methodology followed by system use case. Then we describe different view of the system according to 4+1 view model [106]. Finally, we explain implementation details.

5.1. Architecture Description Methodology

Kruchten [106] have presented an architecture representation model called 4+1 view model for describing architecture of software intensive system. This model has four view (logical, development, process and deployment) plus an additional dimension described as either by using use cases or architecture scenarios description. The four views are briefly described as follows.

Logical View:

This view describes the functionality of the system provided to end-users. Sequence diagrams, class diagrams and collaboration diagrams are used to present this view.

Development View:

This view presents the internal details of sub-systems and sub-services. We can also say that this view illustrates system from development points of view explaining system's internals. Components diagrams, sub-services and package diagrams are used to support this view.

Process View:

This view represents the dynamic (runtime) behavior of the system by explaining system processes using UML activity diagrams.

Deployment View:

This view represents topology of the system on physical layers by elaborating how system components or services are deployed on physical runtime environment and how physical interaction between these components is being taking place. This view is represented by using UML deployment diagrams.

Use Cases:

This view is expressed using UML user case diagrams and represent sequence of interaction between system's features and their actors.

We have chosen specific parts of the middleware platform to provide implementation detailed of different services of the middleware and interaction between applications and services that are provisioned using the middleware.

5.2. Use Cases:

Selective use cases of the middleware platform are shown in Figure 10. The figure incorporated most important use cases of the prototype implementation. Our prototype implementation covers a middleware platform as well as a client application that facilitates existing desktop-based tools to interact with the middleware. On the left side of the diagram, the features of the middleware platform are shown that are used by applications and tools provisioned by the middleware. These features can be accessed by interacting with corresponding APIs as explained in Appendix A.1. The right side of the diagram shows use cases corresponding to the client components/plug-in for accessing middleware features from a desktop based environment. Corresponding to the middleware and the client, actors are classified into two categories, external systems/client applications and users that interact with middleware platform using client applications. A quick description of each use case is provided in the following subsections.

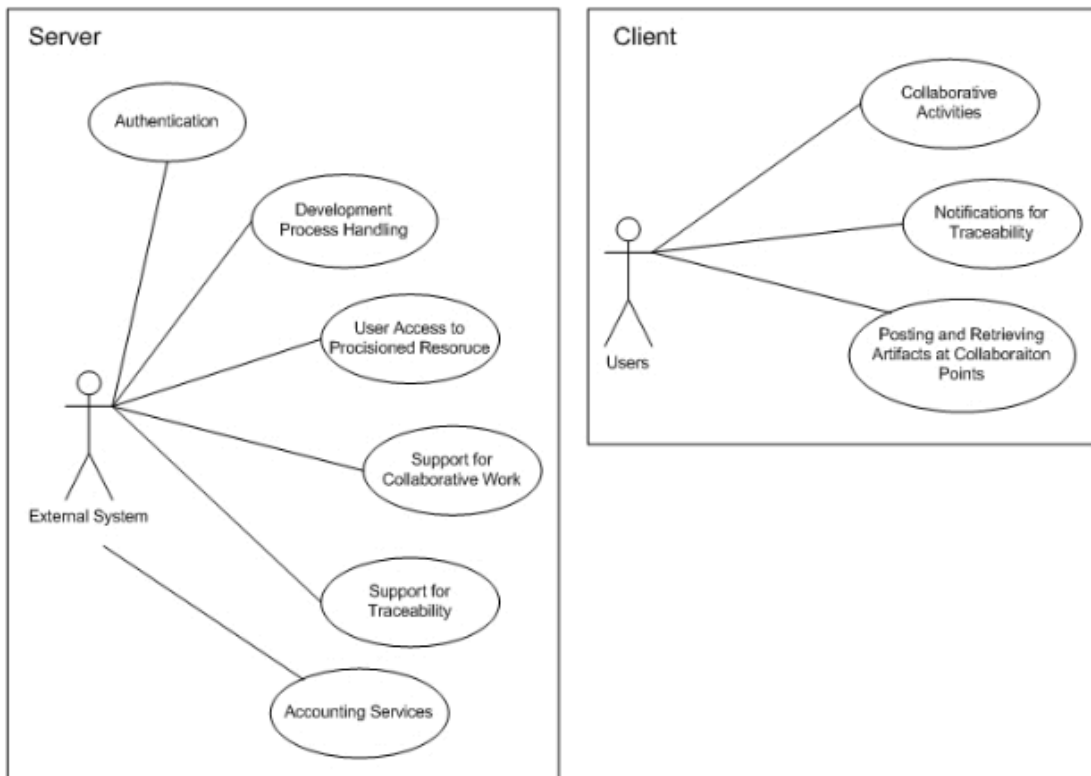


Figure 10: Use Cases

5.2.1. Authentication

Use Case Description:

Before applications, tools and their associated services interact with the middleware platform to utilize its features, these need to be authenticated by the platform. Upon

requesting the platform by providing a valid user id and password, an access key is returned. This access key is in subsequent request to the platform for verification of the credentials.

Pre Condition:

N/A

Post Condition:

Verification key is generated and returned to the client application/tool.

5.2.2. Development Process Handling

Use Case Description:

This use case represents platform features to define a development process, assign different nodes to the development process, define how artifacts are going to be transmitted between the nodes (sequence of data movements between nodes), associate applications and tools with each development node, and assign tenants to the nodes. Additional services can also be attached with the development process flow that can perform some actions on the data when it is transmitted between nodes. Encryption/decryption services and services for verification of artifacts are examples of the intermediate services that can be associated with nodes. Once development process is successfully defined, it is enacted and associated applications and tools are provisioned on the cloud.

Pre Condition:

Application/tools, tenants, and intermediate services should be registered with the middleware.

Post Condition:

Development process is enacted and applications and services are provisioned on cloud infrastructure.

5.2.3. User Access to Resources

Use Case Description:

This use case facilitates users to access provisioned applications and tools, and to utilize features of the platforms; platform provides authentication services that need to be accessed by applications and tools registered on the platform. These services verify if the user is one of the tenants that are assigned to a development site (node of the development process), and if check gives positive results, users are granted access to corresponding resources.

Pre Condition:

Application and tools provisioned by the platform must have implemented authentication services to validate users.

Post Condition:

After successful verification, access to the corresponding cloud resources is granted.

5.2.4. Support for Collaborative Work

Use Case Description:

The platform provides collaboration services for supporting collaborative work on the artifacts using applications and tools provisioned by the middleware platform. A collaboration point is established between every two nodes of the development process that are linked with each other. To utilize the collaboration features, applications and tools need to interact with platform APIs. The artifacts for collaborative work are exchanged between nodes and associated tools according to the defined sequence as described in use case 5.2.2.

Pre Condition:

Application and tools provisioned by the platform must implement features to utilize collaboration services using corresponding APIs.

Post Condition:

N/A.

5.2.5. Support for Traceability**Use Case Description:**

These features represent traceability support by the platform for heterogeneous applications and tools. The platform APIs corresponding to the traceability services facilitated traceability between artifacts within the applications/tools attached to the node in a development process. A traceability points is established between every two nodes of the development process that are linked with each other. To achieve this functionality, the applications and tools need to call platform features through specified APIs.

Pre Condition:

Application and tools provisioned by the platform must have implemented features to utilize traceability services using corresponding APIs.

Post Condition:

N/A.

5.2.6. Accounting Services**Use Case Description:**

To keep record of the cloud resource utilization, platform needs to offers APIs that can be called by the applications and services to register functionalities that are accessed by end users. The provisioning of the resources is transparent to the applications and tools. The applications and tools call corresponding APIs of the billing services to compute resource utilization metrics against each individual users and tenants.

Pre Condition:

Application and tools provisioned by the platform must have implemented features to utilize billing services using corresponding APIs.

Post Condition:

N/A.

5.2.7. Collaboration Activities**Use Case Description:**

The client applications need to provide support for engaging into collaboration task activities as defined in the development process. The end users not necessarily to be aware of the overall collaboration flow and from which collaboration points data need to be fetched or posted, so the whole working should be transparent to end users. The client applications should be able to fetch all the corresponding information against a collaboration activity and perform accordingly.

Pre Condition:

Development process is enacted and collaboration points are established on the platform.

Post Condition:

N/A.

5.2.8. Notifications for Traceability**Use Case Description:**

The client applications and tools need to provide support for receiving traceability notifications from the platform and send notification back using the traceability points defined against the development process and its nodes. Like collaboration activities, the traceability handling should also be transparent to the end users and should be done according to the defined development process.

Pre Condition:

Development process is enacted and traceability points are established on the platform.

Post Condition:

N/A.

5.2.9. Posting and Retrieving Generated Artifacts at Collaboration Points

User Case Description:

The client applications and tools associated with a site (node of a development process) should be able to post generated artifacts to applications and tools associated with next nodes in the process workflow. Similarly when artifacts are generated from preceding nodes in the process workflow, client applications should be able to pull the artifacts generated by applications and tools in the preceding node of the process flow.

Pre Condition:

Development process is enacted and artifacts flow sequence between nodes is defined.

Post Condition:

Artifacts are pulled/pushed as they are produced.

5.3. Deployment View

For pilot implementation of the reference architecture for PTaaS, we are using Amazon IaaS Cloud [19] as underlying infrastructure for hosting the middle platform, its services, and tools and application when these are enacted. Services of the platform are hosted on windows server 2008 with cloud watch and elastic load balancer associated with it. Underlying persistence units of the middleware are hosted on Amazon MySQL Relational Ratabase Service (RDS). Database is wrapper by a RESTful web service wrapper to facilitate data base porting. Middleware and its different services are implemented using RESTful [104] approach, hence its services are stateless and can be scaled by specifying elasticity rules in Amazon's Cloud Watch and Elastic Load Balancer. All instances of the PTaaS access a common RESTful web service wrapper on MySQL database instance (when PTaaS instances are scaled up according to specified parameters). States of the services (e.g. active workflows and associated collaboration and traceability points) is maintained in the database, hence when middleware platform and its composed services are replicated, they can access the state information from a common persistence unit. Server instances are hosted virtual machine on EU-Ireland. Applications and Services that are enacted by the middleware are hosted on Amazon according the location constrains of the tenants associated with applications and services. The collaboration and traceability points can be enacted inside PTaaS or according to the location constraints of tenants that are associated with nodes producing artifacts. High-level view of the deployment view is presented in Figure 11. Detailed architecture of the middleware PTaaS can be found in section 4 and is not described here again to avoid redundancy.

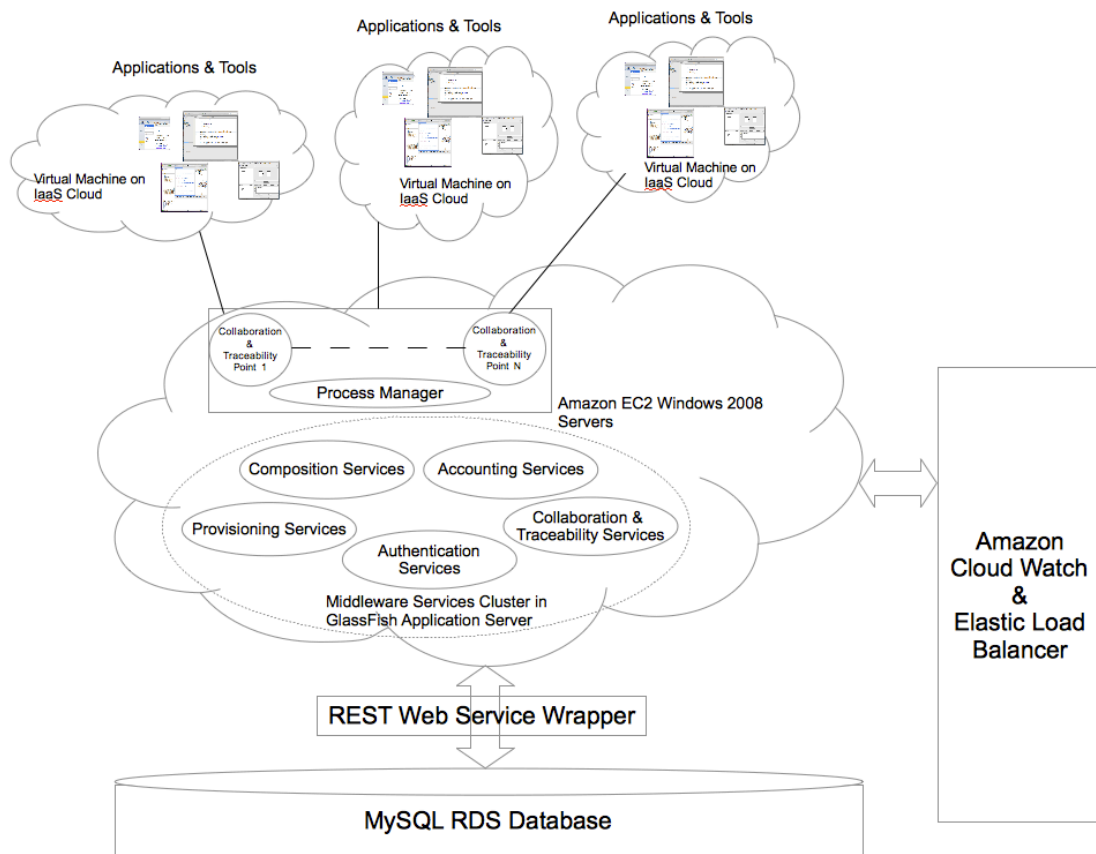


Figure 11: Deployment View

5.4. Logical View

The logical view of the middleware platform is explained by using a sequence diagram, showing the interaction of middleware platform services involved in the defining the development process, adding nodes to it, assigning applications/tools with a development site, defining artifacts flow between applications/tools and enacting them on the underlying cloud infrastructure along with collaboration and traceability points corresponding to each interaction point between process nodes. There can be multiple development processes inside main development process but to keep the sequence diagram simple, we are considering only the root level development process. We also provide a quick overview of the interaction happening between the technologies of adopted along with code snippets of the selected parts of middleware services.

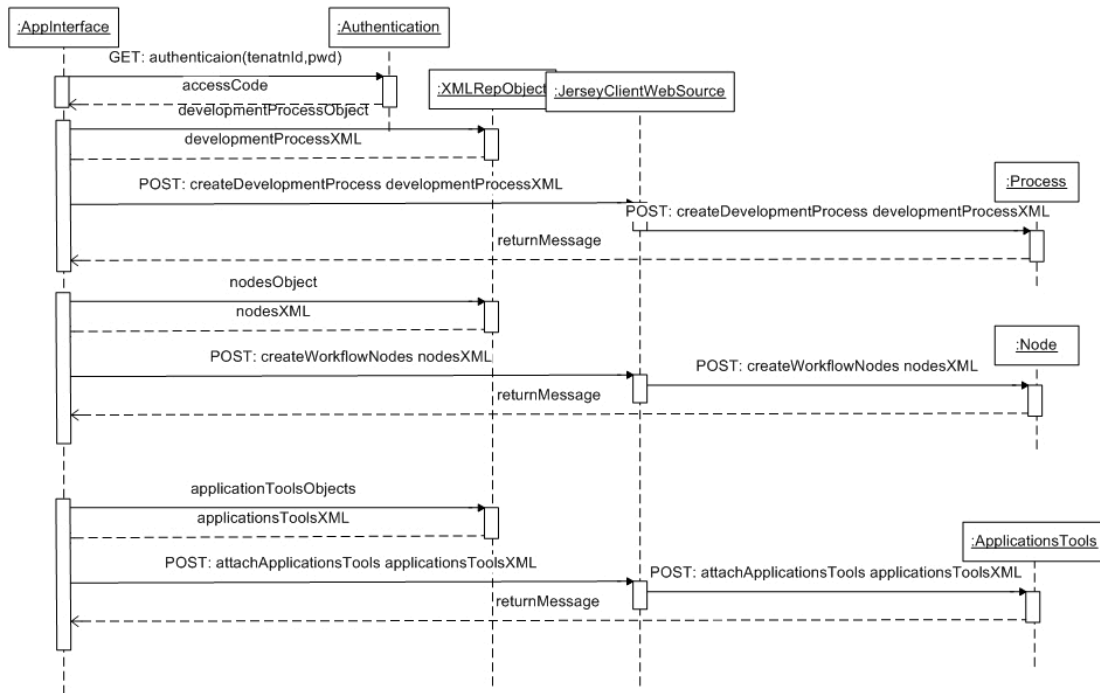


Figure 12: Sequence Diagram - Define and Enact Development Process - (i)

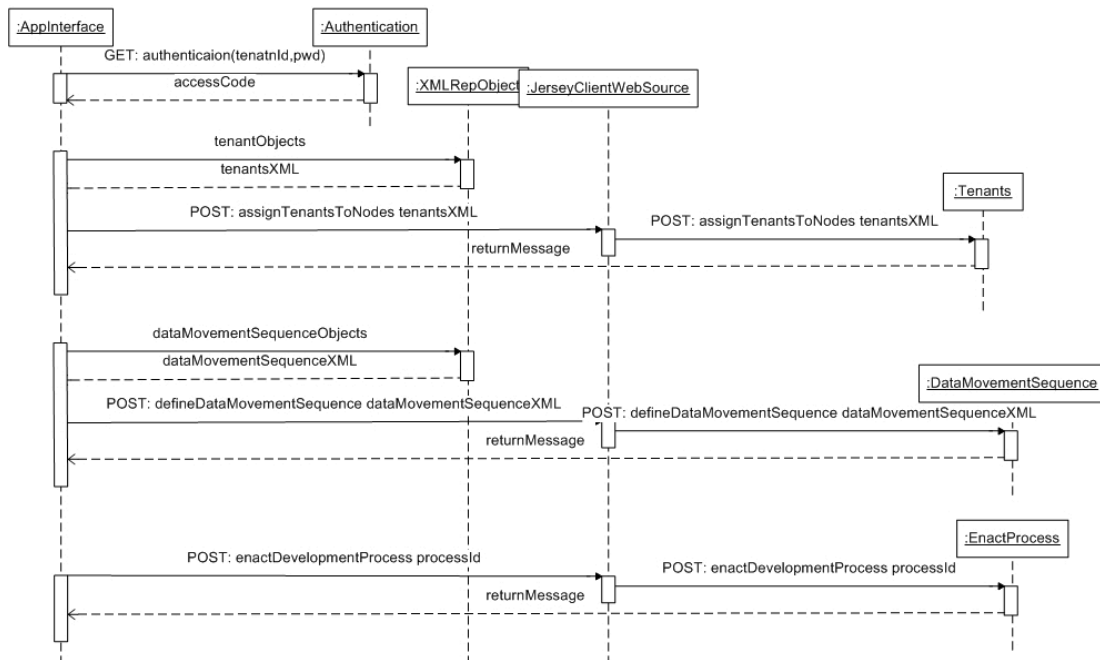


Figure 13: Sequence Diagram - Define and Enact Development Process - (ii)

Sequence diagram shown in Figure 12 and Figure 13 presents interaction of the services involved in defining and enacting software development process. The interfaces of all the services are exposed in a RESTful manner [104, 107] and are implemented using Jersey framework (JSR 252 specifications) [108]. Before external applications can access system services, they need to acquire an access code from **Authentication** service. The authentication can be acquired by providing id and password of user and corresponding tenant in http header. The platform features can also be accessed by providing user and tenant credentials with every request. However, the authentication mechanism is provided so that users and tenants

credentials do not need to be exposed with every single request. Once access code is obtained, it is passed in http header for subsequent requests to the platform features. Figure 14 shows web service method that generates and returns authentication code.

```

@GET
@Path("/{applicationId}")
@Produces({"plain/text"})
public String getAuthenticaiionCode(@PathParam("applicationId") String applicationId,
    @HeaderParam("userId") String userId, @HeaderParam("userPwd") String userPwd,
    @HeaderParam("tenantId") String tenantId, @HeaderParam("tenantPwd") String tenantPwd)
{
    String authenticationCode = "";
    try
    {
        if( verifyUser(userId, userPwd) || verifyTenant(tenantId, tenantPwd) )
        {
            authenticationCode = generateAccessCode(applicationId);
        }
    }
    catch(Exception ex)
    {
        System.out.println(ex.toString());
    }
    return authenticationCode;
}

```

Figure 14: Authentication Service

The platform APIs receive/send request in from of XML documents. Java objects inside applications are converted into XML documents using javax XmlElement and XmlRootElement annotation. Figure 15 shows code snippet of node object with XML annotations. Details of the services APIs used to create software development process and assign applications tools and tenants to it are provided in Appendix A.1.3.5.

```

package beans;

import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

/**
 *
 * @author muac
 */
@XmlRootElement
public class Node{
    private String id;
    private String name;
    private String procnodeid;
    private String precnodeid;
    private String afl;
    private String af2;
    private Workflow workflowId;

    public String getId() {...}

    @XmlElement
    public void setId(String id) {...}

    public String getName() {...}

    @XmlElement
    public void setName(String name) {...}
}

```

Figure 15: Using XML annotations for converting java objects into XML documents

Figure 16 shows code that is used to launch Amazon EC2 [19] instances hosting desktop based applications and tools. The RESTful web service wrapper is using Java Persistence APIs (JPA) [109] to interact with database. Figure 17 shows node entity class that is used to persist node information in the database.


```

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

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

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

```

Figure 16: Code for enacting Amazon EC2 Instances hosting Applications and Tools

```

public class Node implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 100)
    @Column(name = "id")
    private String id;
    @Size(max = 1000)
    @Column(name = "name")
    private String name;
    @Size(max = 1000)
    @Column(name = "procnodeid")
    private String procnodeid;
    @Size(max = 1000)
    @Column(name = "precnodeid")
    private String precnodeid;
    @Size(max = 1000)
    @Column(name = "af1")
    private String af1;
    @Size(max = 1000)
    @Column(name = "af2")
    private String af2;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "nodeId")
    private Collection<ToolSuit> toolSuitCollection;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "nodeId")
    private Collection<Collaboration> collaborationCollection;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "nodeId")
    private Collection<NodeTenant> nodeTenantCollection;
    @JoinColumn(name = "workflow_id", referencedColumnName = "id")
    @ManyToOne(optional = false)
    private Workflow workflowId;
}

```

Figure 17: Node Entity

5.5. Process View

The process view is illustrated through activity diagrams corresponding to creating and enactment of the software development process, collaboration and traceability. Figure 18 shows the overall activities to create a software development process in the middleware for provisioning of applications and tools. The activity begins by selecting/specifying software development process name through front-end application of the middleware platform. If the process against the selected/specified name is already defined then an exception is raised. If the process is not already defined, user can create different nodes of the development process. Each node at the root process represents the development site involved in GSD. Once nodes are defined, the applications/tools and tenants are attached to the nodes. All the users belonging to the attached tenants have access to the applications/tools attached to the node. The sequence of the nodes in the process is defined. This sequence is used to allow applications/tools in the proceeding nodes have access to artifacts generated by the applications/tools of the preceding nodes. The applications/tools attached to a node

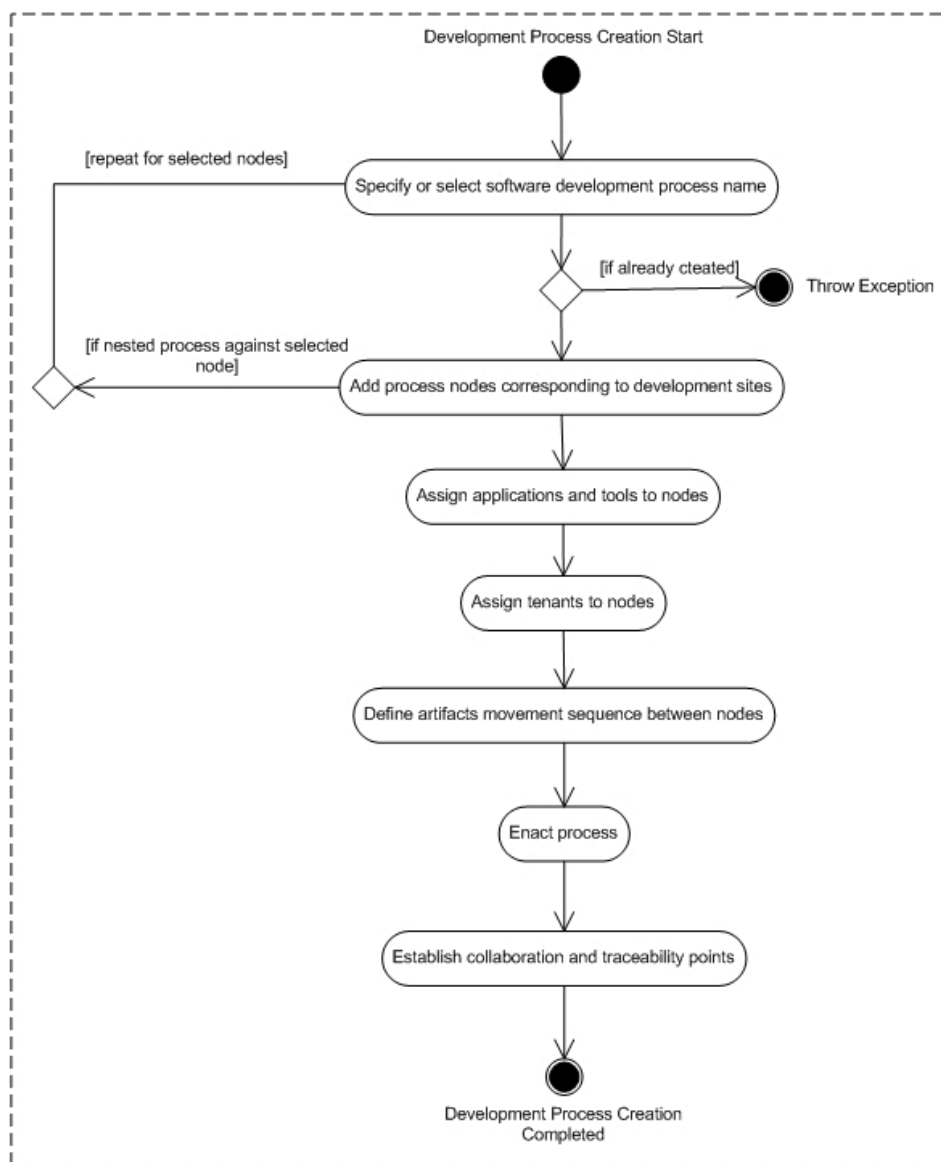


Figure 18: Activity diagram for enacting software development process

can engage in collaborative work with application/tools of the proceeding and preceding nodes and can receive traceability notifications if the artifacts associated with traceability points are updated. Intermediate applications can be attached to the process to perform specific types of operations on the artifacts before they are passed on to the tools in proceeding nodes.

Activity diagrams for collaborative work on artifacts are shown Figure 19. The platform established a collaboration point for each pair of interacting nodes. For example, if three nodes N1, N2 and N3 are defined in the software development process such that N2 is proceeding node of N1 and preceding node of N3. The platform established two collaboration points one between N1 and N2, and one between N2 and N3. Two types of collaboration activities are supported by the platform: push in which platform application/tools register their RESTful callback methods with the collaboration points so that updated versions of the shared artifacts can be automatically pushed to the applications/tools, and pull in which application/tools check the collaboration points for updated artifacts. Details of the activities for posting and retrieving artifacts for push and pull types are shown in Part A and B of Figure 19.

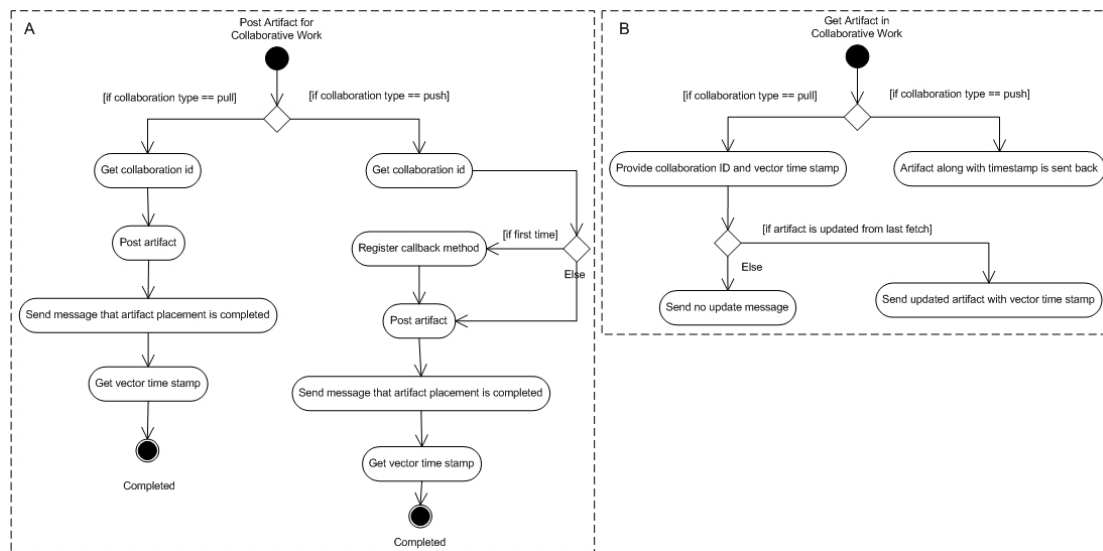


Figure 19: Activity diagram for collaboration

The platform established the traceability point between two nodes in same way as it defines collaboration points. Vector time stamp is used for maintaining artifacts' information on traceability and collaboration points. The vector time stamp consists of form $[t_1, t_2]$ where t_1 and t_2 are logical time for application A at node 1 and application B at node 2 respectively, exchanging information through traceability point between node 1 and node 2. If application A make update in the artifacts and post this information of update on traceability point t_1 is incremented by one. If application B make update in the artifact and post the information on traceability point t_2 is incremented by one. For N applications associated with a traceability point, vector time stamp takes for $[t_1, t_2, t_3, \dots, t_N]$ corresponding to applications $[A_1, A_2, A_3, \dots, A_N]$. Detail of how the information is posted and retrieved from traceability points is elaborated in part A and B of Figure 20. Detail on working of collaboration and traceability is provided in section 4.3.5.

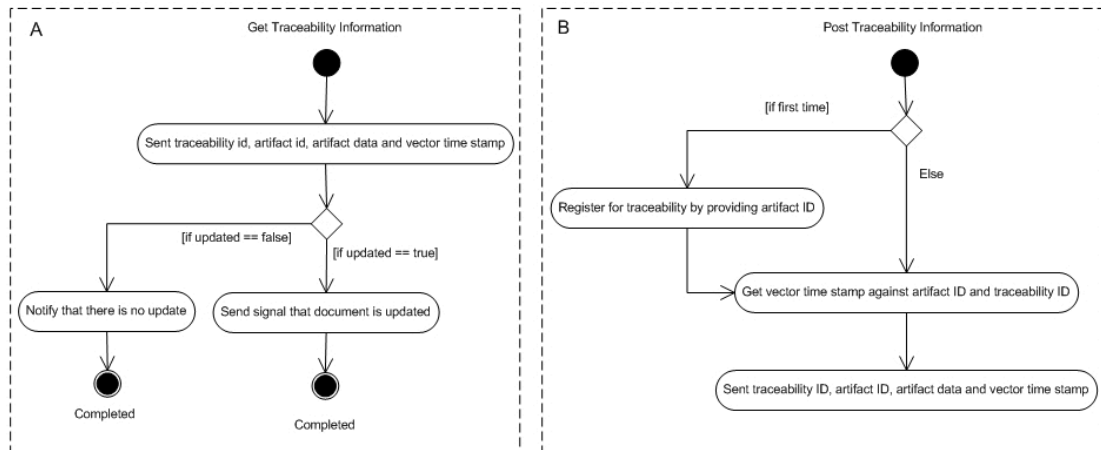


Figure 20: Activity diagram for traceability

5.6. Data Model

Data model used for persistence is shown in Figure 21. The boxes in the figure represents group of entities responsible to persist data from respective services. In order to avoid unnecessary complexities, we are not providing details of the entities inside each entity group. The relationship between entity groups in maintained at services level, i.e. relation between entities inside entity group is maintained at database level in form of referential integrity constrains whereas inter entity group relationships are logical relations maintained in form of aggregation and composition of services corresponding to entity groups. In this section, we are only explaining relations of the entity groups and how they satisfy the requirements of the platform for provisioning of applications and tools as services.

Workflow is central entity group of the data model that is used to store information of process workflows. Process workflow stores information of the development process being following by organizations involved in GSD project. More details are described in sections 4.3.5 and 5.7.2. Workflow is composed of multiple Nodes. Nodes information is stored in *Node* entity group. Each node corresponds to one element of the development process. At first level of abstraction, a node is a development site involved in a GSD project. At second level of abstraction, a node can be a department inside a development site. Nodes are associated with tool suits and tenants, and this information is maintained in *Tool Suit* and *Tenants* entity groups. A tool consists of combination of applications associated with a Node. Tenants are assigned to nodes. All the users belonging to assign tenants have access to tools associated with nodes. Deployment information of the applications is stored in *Deployment Information* entity group. Deployment information consists of deployment scripts that are needed to deploy applications and tools on cloud when are enacted. In case of desktop based tools, information of virtual machine images are maintained in *Virtual Machine Images* entity group. This group maintains information of cloud environments, virtual machine image identities, their access information, and scripts to launch virtual machine images. Applications and tools may be composed of multiple sub-services. The information of sub-services is maintained in *Services* entity group and their deployment information is maintained in associated *Deployment Information* entity group. For all interaction nodes of the process workflow, collaboration and traceability services are enacted as described in section 4.3.5. The information of where collaboration and deployment services are deployed and enacted is stored in *Collaboration Service* and *Traceability Service* entity groups.

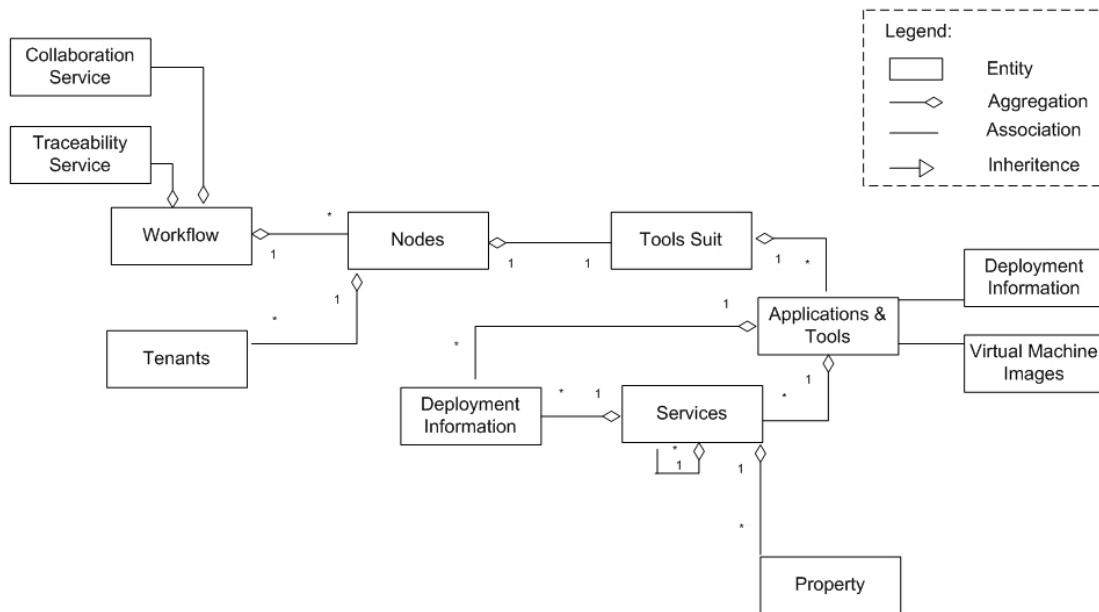


Figure 21: Data Persistence Model

5.7. Implementation Overview

The prototype implementation of the proposed reference architecture consists of RESTful middleware and three applications. The middleware implements core features of the reference architecture presented in Chapter 4. Front end of the middleware is web-based application that is used to facilitate applications and tool provisioning as services. Second client application facilitates administration of the middleware and facilitates registration of applications/tools with the middleware along with related information. Third application is a client implemented to facilitate interaction of desktop-based applications and tools with the middleware. The desktop is implemented to provide an easy access for the desktop-based applications and tools to interact with middleware. The tools and applications can also directly access the middleware without relying on client application. In the following sub sections, we provide overview of the middleware implementation, web based client applications facilitating defining process and provision tools according to the requirements, and desktop client.

5.7.1. Middleware Platform Services

The middleware consists of following RESTful services [104] implemented using Jersey Framework [110]; an implementation of JSR 252 specification [108]. The middleware platform is using Amazon EC2 [19] as underlying infrastructure as a service cloud. In our pilot implementation we have only implemented following services dealing with core part of the platform. For managing life cycle of the middleware platform resources, we are relying on Amazon Cloud Watch [111] and Load Balancer [112].

User, Tools and Applications Registration Services:

These services facilitate registration of the users and application/tools with the middleware platform. Each user in the system is associated with a tenant. A tenant can be a team working on a particular activity of a GSD project, all the stakeholders from a location involved in a GSD project, or the whole staff or the organization involved in GSD project. Access to application/tools and corresponding resources is

granted to tenants. All the users belonging to that tenant automatically get access to the resources. Applications and tools are provisioned by the platform dynamically and the platform control access to the provisioned resources. The applications and tools need to call platform for the verification of the users.

Tools and applications registration maintains repositories of the applications and services, their deployment information, information of the cloud images stored on underlying IaaS cloud and deployment scripts need to deploy the services and associated database scripts in case of web based applications.

Authentication Service:

This service facilitates authorization of the users to access an application or tool hosted on cloud. The applications that need to verify the users need to call authentication services for verification of the users. The platform checks if the corresponding tenant of the users is associated to the nodes on which applications are attached, and grant or reject access accordingly.

Process Management and Enactment Services:

This set of services facilitates defining software development process in the middleware, defining nodes in the process, specifying sequence of information flow between nodes of software development process, assigning applications and tools with the nodes and enacting them on underlying IaaS cloud. If users select desktop-based applications and clients, preconfigured cloud images are invoked. If based applications and tools are selected, they are deployed on the cloud resources according to specification of end users. The platform supports defining nested processes to accommodate individual processes inside

Two types of constraints can be imposed on applications and service provisioning. Location constraints specifying if applications and tools are to be deployed and enacted on certain regions of the cloud. E.g. European, American or Asian regions. The location preferences are associated with tenants. Services are invoked according to the location constraints of the tenant assigned to a particular node. For web-based tools there is an additional constraint regarding sharing of services. If tenants do not need exclusive access to the applications and tools, already invoked instances of the services are shared among new tenants who want to avail the services. If exclusive access is required then new instances of the services are invoked.

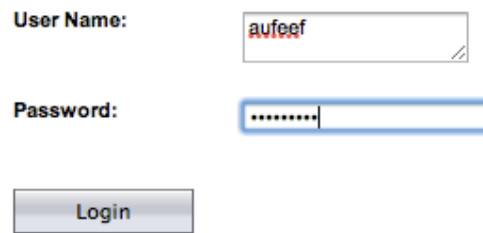
Collaboration and Traceability Services:

For every two interacting nodes inside the process, collaboration and traceability services are invoked. The collaboration services provide a bridge for exchanging information on collaborative work. The traceability service serve a notification point to raise alarms if the documents defined as a part of traceability and maintained inside an applications or tool are changed. The detail on working of collaboration and traceability services is provided in section 4.3.5.

5.7.2. Client Application for Defining Software Development Process in the Middleware and Invoking Application and Tools

Front end of the middleware provides graphical user interface to the platform APIs for defining software development process, adding nodes in the development process and assigning applications and tools with development nodes. The frond end also facilitated assigning tenants to the nodes and defining sequence of collaboration and

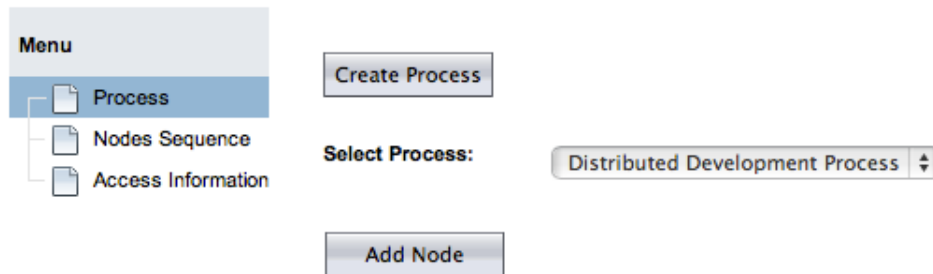
data movement between nodes. Figure 22 shows authentication screen. Users with administrative privileges can have access to the platform.



The authentication screen consists of three main components: a 'User Name:' label followed by a text input field containing the text 'aufeeef'; a 'Password:' label followed by a password input field with masked characters '.....'; and a 'Login' button centered below the input fields.

Figure 22: Authentication

Users can define a new process by pressing create process button as and then specifying process name as shown in Figure 23 and Figure 24. Once the process name is entered, nodes can be added to the process by invoking corresponding interface by clicking Add Node button.



The 'Creating Process (i)' screen features a 'Menu' on the left with three items: 'Process' (highlighted), 'Nodes Sequence', and 'Access Information'. To the right, there is a 'Create Process' button, a 'Select Process:' dropdown menu currently showing 'Distributed Development Process', and an 'Add Node' button.

Figure 23: Creating Process (i)



The 'Creating Process (ii)' screen is a simple form with a 'Name:' label and an empty text input field. Below the input field is a 'Save' button.

Figure 24: Creating Process (ii)

Once process is defines, nodes can be added to the process. With each node, a set of tools and tents can be attached. Nested nodes can be defined inside a node representing nested processes. For example, if first level nodes represent sites involved in GSD development, nested nodes may be a representation of teams involved in a project. GUI for node management is shown in Figure 25. In the scenario presented in the diagrams a simple process is defined involving two development nodes and one testing node. First development node is associated with an Eclipse IDE. Second development node is associated with NetBeans IDE and testing node represents a virtual machine where code generated by node 1 and 2 can be deployed there.

Figure 25: Adding Nodes/Sites to the Process

Once all nodes in the process are defined, nodes sequence and connection between nodes can be defined by using GUI shown in Figure 26. Nodes listed under source are predecessors of the nodes specified in the destination. The last node of the source does not have any destination. Some intermediate services can be assigned to process data when it is transmitted from source to destination. In the scenario presented in Figure 26, an intermediate service for compiling java source code classes and making a jar file is associated when artifacts are transmitted from development node 2 to testing node 1. After defining the nodes sequence and specifying intermediate services between nodes if needed, the process flow can be saved and enacted by pressing “Save and Enact” button. Once the process is enacted, a collaboration point and a traceability point are also established and enacted between every source and destination nodes. After successful deployment of services and establishment of collaboration and traceability point, access information is displayed as shown in Figure 27. In the scenario presented in diagrams, development node 1 is source for node 1 and node 2 is source for testing node 1.

The scenario depicted in the diagrams corresponds to a simple case in which there are three nodes in a process connected to each other in a waterfall fashion. The platform can accommodate complex scenarios where multiple nodes can be source or target for produced artifacts. In case that there are more than one applications or tools working on a collaboration point, the applications and services are required to have corresponding services that can be called whenever collaboration points are accesses. These services take care of how data is merged together and data integrity is maintained.

Source(s)	Destination(s)	Intermediate Service(s):
Development node 1	Development node 2	Select
Development node 2	testing node 1	Java Code Compiler
testing node 1	Select	Select

Figure 26: Defining Data Movement Sequence Between Nodes/Sites

<div style="border: 1px solid gray; padding: 5px; background-color: #e0e0e0;"> Menu <ul style="list-style-type: none"> Process Nodes Sequence <li style="background-color: #a0c0ff;"> Access Information </div>	<p>Node 1 Access Info: Eclipse IDE: ec2-54-216-10-165.eu-west-1.compute.amazonaws.com User: Administrator, Decrypted Password: Q\$CxB2(Y6FW</p> <p>Node 2 Access Info: Net Beans IDE: ec2-176-34-223-31.eu-west-1.compute.amazonaws.com User: Administrator, Decrypted Password: PD254Q\$</p> <p>Node 3 Access Info: Testing Computer: ec2-54-246-69-254.eu-west-1.compute.amazonaws.com User: Administrator, Decrypted Password: Lkjh_ouoj(/</p>
--	--

Figure 27: Displaying Access Information

5.7.3. Desktop Client

Desktop client is implemented to facilitate interaction of existing desktop-based software development tools with the platform and to demonstrate access to middleware platform from clients. Although, desktop applications and tools can also access the middleware through plug-in and extensions, but it is out of scope of the thesis to develop plug-in for applications/tools, that is why we have not implemented the plug-in for Eclipse and NetBeans IDEs. Software development process and information flow between different nodes of the process is maintained by the middle. The users of applications, or applications do not need to take care of how artifacts are passed and made available to the consumers. They only need to post the artifacts on

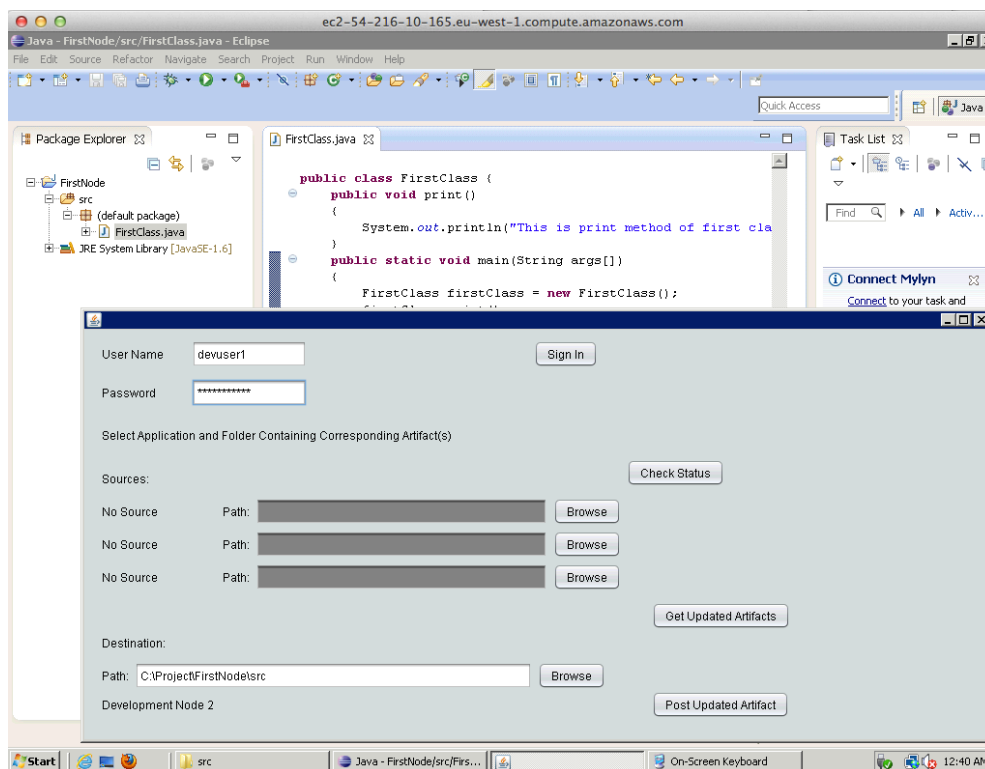


Figure 28: Desktop Client Supporting Collaboration and Traceability – (i)

corresponding collaboration points (information for change in artifacts in case of traceability points) and the middleware take care of making the artifacts available to the corresponding nodes and generating notifications (in case application has registered their RESTful callback methods with the middleware).

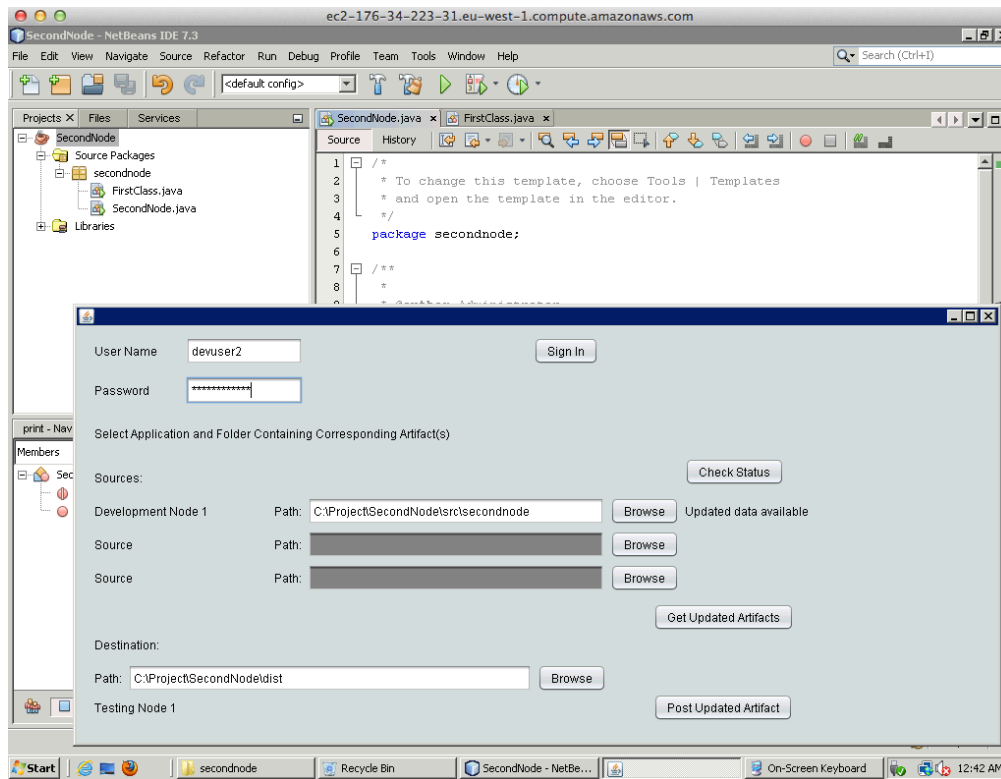


Figure 29: Desktop Client Supporting Collaboration and Traceability - (ii)

The screen shots of the enacted virtual machine instances of the scenario described in section 5.7.2 are shown in Figure 28, Figure 29 and Figure 30. Virtual machine images used for provisioning of desktop-based applications and tools are preconfigured with DesktopClient. Users become part of the development process when process workflow is enacted and get access to the resources attached with a development node including collaboration and traceability points by signing in on the desktop client. In case there are multiple users inside tenants assigned to a node, separate virtual machines are enacted for user. The machines have the application and tools configured on them (which are specified when workflow nodes are defines). In case of web-based applications, only one instance for each application is created for a tenant and all the users in the tenant have access to the applications. Figure 28 shows screenshot corresponding to Development Node 1 with eclipse IDE and desktop client installed on VM. “When devuser1” signs into desktop client, the information for user is sent to middleware for verification of credentials. If credentials are successfully verified, the DesktopClient fetches information of collaboration points associated with the node. Development Node 1 one is only a producer of an artifact; hence the DesktopClient only displays the name of destination node, which in this case is Development Node 2. If there are multiple destination nodes, there names are displayed accordingly. The location of the data artifacts that are produced is selected by pressing Browse button. Once artifacts are ready to be posted, they can be posted on the collaboration point by pressing post button. Details the internal working of collaboration can be found in section 4.3.5 and section 5.5.

Figure 29 corresponds to Development Node 2 with NetBeans IDE and DesktopClient. Development Node 2 is consumer of artifacts from the Development Node 1 and producer of artifacts for Testing Node 1, hence both source and destination are mentioned by the DesktopClient. When updated artifacts are available on Development Node 1, a message is displayed by the desktop client to raise awareness about the activity of the users from the Development Node 1. Users at Development Node 2 can get document on select location by pressing “Get Updated Artifacts” button. Produces artifacts can be posted to collaboration node between Development Node 2 and testing Node 1 when these are ready to be posted. Figure 30 shows the desktop client on Testing Node 1. Its works in the same fashion as described for Development Node 1 and Development Node 2. Testing Node 1 is the last node of the process that is why destination option is disabled. Java Code Compilation service is attached when data is passed from Development Node 2 to Testing Node 1. When user at Testing Node 1 press “Get Updated Artifact” button, source code available with collaboration point at node is compiled and send at Testing Node 1.

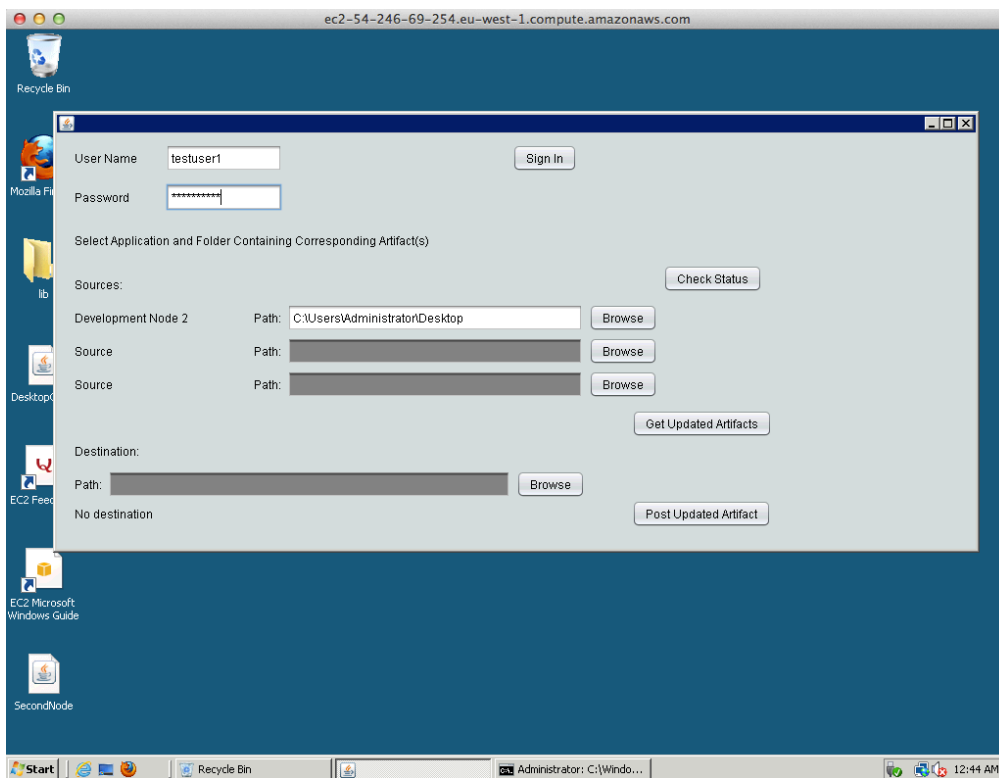


Figure 30: Desktop Client Supporting Collaboration and Traceability - (iii)

6. Conclusion and Directions for Future Work

In this thesis, we have presented the reference architecture and prototype implementation of middleware platform named PTaaS for providing software development applications and tools as services to address GSD challenges. The first phase of effort to provide a middleware platform that can provision GSD applications and tools as services resulted in a set of requirements to be considered while developing the middleware [22]. These requirements are further enhanced in this thesis and followed by a careful design of reference architecture of the middleware platform as well as its prototype implementation. In the following points, we provide an overview of how the requirements for the middleware platform are satisfied by the reference architecture and its prototype implementation.

- ***Supporting applications of different types***

The middleware platform APIs are exposed as REST interfaces which make it possible for applications and tools implemented in different programming languages to access the features of the middleware platform. Underlying resources (e.g. virtual machines) are acquired from the IaaS cloud as needed to host application and tools.

- ***Authentication and authorization of users within applications***

When applications and tools are provisioned by the platform, the authorization and authentication schemes need to be taken out of the application control because assignment of tenants and users with the tools is managed by the middleware platform. To facilitate authentication and authorization of application users, special services are provided by the platform. Application and tools hosted on the platform needs to modify their authentication schemes according to the specifications of the platform's authentication services.

- ***Managing applications and tools provision***

Lifecycle and Provisioning services along with Repository Management services of the middleware platform maintain the application and tools and enact them according to their location and multi-tenancy constraints.

- ***Alignment of tools with the processes***

On demand provisioning of software development applications and tools make it easy for organizations to acquire needed tools according to the processes. Moreover, the ability of PTaaS to capture the software development process, assign tenant (and users inside a tenant) and application/tools to nodes of the software development process supports applications and tools alignment with process and their seamless integration.

- ***Integration and support for collaboration & traceability***

To streamline the software development, software development applications need to interact with each other in order to exchange artefacts. PTaaS automatically establish collaboration and traceability services between all interaction points of the software development process to facilitate the integration between applications and tools. It provides applications and tools ability to integrate with each other, collaboratively work on artefacts and engage in workflow kind of activities.

- ***Support for Virtual Teams***

Virtual teams are characterized in GSD as teams located at distributed locations and yet able to engage in shared and collaborative tasks. Tenants (along with users inside tenants) are assigned to nodes (for example a distributed sites, or a department on a site involved in GSD project) of the process. Applications and tools are also assigned to the nodes that can collaborate with each other through collaboration points. These features of the middleware support Virtual Teams to effectively engage in GSD activities.

Having identifying requirements of the middleware platform through a review of literature on GSD, reference architecture for the middleware platform is proposed that facilitates provision of software development applications and tools as services according to location and multi-tenancy constraints specified by the tenants. The platform is capable of capturing software development process and makes sure that artefacts flow among applications according to the defined process. For facilitating heterogeneous applications and tools to collaborate with each other, collaboration points (services) are established between interacting nodes of the process through which applications and tools can exchange artefacts. Applications and tools maintain artefacts in their own data store units according to proprietary formats but the artefacts may be dependent upon other artefacts and need to be modified if the artefacts on which they are dependent are changed. To accommodate this kind of inter application traceability, the platform also established traceability points through which applications and tools can send or receive traceability notifications if artefacts in the traceability chain are modified. In short, this thesis makes three significant contributions:

1. It provides theoretical foundation for providing software development applications and tools as services.
2. It provides requirement that should be accommodated in order to make provisioning of applications and tools as services in GSD context.
3. It provides details on design strategies and a reference architecture, which consists of multiple architecture styles; for building the platform that can facilitate desired provisioning of applications and tools by using cloud-computing paradigm.

We have demonstrated feasibility of the proposed approach through prototype implementation of most important features of the reference architecture by using Amazon as underlying IaaS cloud. The implementation consists of a set of middleware platform services including authentication services, registration services (for users, tenants, applications and tools), process management and enactment services, and collaboration and traceability services. In order to facilitate provisioning of applications and tools as services, we have implemented a web based client application through which users can define their software workflow process, attach application and tools as well as middleware services to different nodes of the workflow and can specify tenants that should be allowed to have access to the applications and tools. The information to access the enacted resources is also displayed through web based interface. A client application is implemented to facilitate desktop-based tools hosted on virtual machines to interact with the platform.

In this thesis, we have not only established foundations for providing software development tools and applications as services but have also provided practical solutions in terms of reference architecture and prototype implementation. However,

we realise that the approach presented and strategies discussed in this thesis need to be further enhanced and evaluated for more complex software development scenarios. In future, we intend to extend the reference architecture and implementation for hybrid cloud environments where the platform can also incorporate applications and tools provisioning on private cloud, and support collaboration and traceability between them. In some cases, for example software testing, data artefacts are too large to transmit from one location to another. The more feasible approach in such cases is to enact applications and tools closer to data sources so that minimum data transmission is required. We intend to extend the architecture to accommodate enacting applications and tools closer to data sources. In current architecture and prototype implementation we are catering relation of artefacts with specific types of tools and applications (for example development tools, design tools), and their sub types (for example Visio or ArgoUML in case of design tools) for supporting collaboration and traceability, but we are not considering standardizations that these tools are following (for example design tools following a specification like Object Management Group's specification⁶ of UML 1.0 or 2.0, or SoaML). In future, we intend to enhance the architecture in a way that standardizations and corresponding meta-models are recognized by the platform and can be dynamically enhanced. Last but not the least, in some cases applications and tools generate executable artefacts (for example executable files). Such artefacts can cause security threats on consumer applications and tools if not properly validated before that are transmitted to them. It can be explicitly handled in current architecture and prototype implementation by specifying intermediated services to process data before it is transmitted to consumers; however we intend to extend this feature further into more implicit and a default security model of the platform.

⁶ <http://www.omg.org/>

References

- [1] M. Armbrust, *et al.*, "A view of cloud computing," *Commun. ACM*, vol. 53, pp. 50-58, 2010.
- [2] A. Lenk, *et al.*, "What's inside the Cloud? An architectural map of the Cloud landscape," in *Software Engineering Challenges of Cloud Computing, 2009. CLOUD '09. ICSE Workshop on*, 2009, pp. 23-31.
- [3] P. Louridas, "Up in the Air: Moving Your Applications to the Cloud," *Software, IEEE*, vol. 27, pp. 6-11, 2010.
- [4] L. Xiaolin, "An Approach to Service and Cloud Computing Oriented Web GIS Application," in *2010 International Conference on Internet Technology and Applications*, ed: IEEE, 2010, pp. 1-4.
- [5] L. Xiaolin, "Service and cloud computing oriented web GIS for labor and social security applications," in *2010 2nd International Conference on Information Science and Engineering (ICISE)*, ed: IEEE, 2010, pp. 4014-4017.
- [6] M. Azambuja, *et al.*, "An Architecture for Public and Open Submission Systems in the Cloud," in *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*, ed: IEEE, 2010, pp. 513-517.
- [7] R. Pereira, *et al.*, "An Architecture for Distributed High Performance Video Processing in the Cloud," in *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*, ed: IEEE, 2010, pp. 482-489.
- [8] R. Pereira and K. Breitman, "A Cloud Based Architecture for Improving Video Compression Time Efficiency: The Split & Merge Approach," in *Data Compression Conference (DCC), 2011*, ed: IEEE, 2011, pp. 471-471.
- [9] P. Rodriguez, *et al.*, "VaaS: Videoconference as a service," in *5th International Conference on Collaborative Computing: Networking, Applications and Worksharing, 2009. CollaborateCom 2009*, ed: IEEE, 2009, pp. 1-11.
- [10] W. Cellary and S. Strykowski, "e-government based on cloud computing and service-oriented architecture," presented at the Proceedings of the 3rd international conference on Theory and practice of electronic governance, Bogota, Colombia, 2009.
- [11] M. Pokharel, *et al.*, "Cloud Computing in System Architecture," in *International Symposium on Computer Network and Multimedia Technology, 2009. CNMT 2009*, ed: IEEE, 2009, pp. 1-5.
- [12] D. Zisis and D. Lekkas, "Securing e-Government and e-Voting with an open cloud computing architecture," *Government Information Quarterly*, vol. 28, pp. 239-251, 2011.
- [13] S. van der Burg, *et al.*, "Software deployment in a dynamic cloud: From device to service orientation in a hospital environment," in *ICSE Workshop on Software Engineering Challenges of Cloud Computing, 2009. CLOUD '09*, ed: IEEE, 2009, pp. 61-66.
- [14] N. Botts, *et al.*, "Cloud Computing Architectures for the Underserved: Public Health Cyberinfrastructures through a Network of HealthATMs," in *2010 43rd Hawaii International Conference on System Sciences (HICSS)*, ed: IEEE, 2010, pp. 1-10.
- [15] S. Chia-Ping, *et al.*, "Bio-signal analysis system design with support vector machines based on cloud computing service architecture," in *2010 Annual*

- International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, ed: IEEE, 2010, pp. 1421-1424.
- [16] F. Baiardi and D. Sgandurra, "Securing a Community Cloud," in *Distributed Computing Systems Workshops (ICDCSW), 2010 IEEE 30th International Conference on*, 2010, pp. 32-41.
- [17] A. Khajeh-Hosseini, *et al.*, "Cloud Migration: A Case Study of Migrating an Enterprise IT System to IaaS," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, 2010, pp. 450-457.
- [18] Q. Zhang, *et al.*, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, pp. 7-18, 2010.
- [19] "Amazon, <http://aws.amazon.com/> [May, 2013]."
- [20] M. A. Babar and M. A. Chauhan, "A tale of migration to cloud computing for sharing experiences and observations," presented at the Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing, Waikiki, Honolulu, HI, USA, 2011.
- [21] S. I. Hashmi, *et al.*, "Using the Cloud to Facilitate Global Software Development Challenges," in *Global Software Engineering Workshop (ICGSEW), 2011 Sixth IEEE International Conference on*, 2011, pp. 70-77.
- [22] M. A. Chauhan and M. A. Babar, "Cloud infrastructure for providing tools as a service: quality attributes and potential solutions," presented at the Proceedings of the WICSA/ECSA 2012 Companion Volume, Helsinki, Finland, 2012.
- [23] J. D. Herbsleb, "Global Software Engineering: The Future of Socio-technical Coordination," in *Future of Software Engineering, 2007. FOSE '07, 2007*, pp. 188-198.
- [24] J. D. Herbsleb and D. Moitra, "Global software development," *Software, IEEE*, vol. 18, pp. 16-20, 2001.
- [25] F. Lanubile, *et al.*, "Collaboration Tools for Global Software Engineering," *Software, IEEE*, vol. 27, pp. 52-55, 2010.
- [26] Rodri, *et al.*, "Technologies and Tools for Distributed Teams," *Software, IEEE*, vol. 27, pp. 10-14, 2010.
- [27] Tell P. and A. B. M., "A Systematic Mapping Study of Tools for Distributed Software Development Teams, TR-2012-161, Oct. 2012.."
- [28] I. Steinmacher, *et al.*, "Awareness support in global software development: a systematic review based on the 3C collaboration model," presented at the Proceedings of the 16th international conference on Collaboration and technology, Maastricht, The Netherlands, 2010.
- [29] J. Portillo-Rodríguez, *et al.*, "Tools used in Global Software Engineering: A systematic mapping review," *Information and Software Technology*, vol. 54, pp. 663-685, 2012.
- [30] P. Dourish and V. Bellotti, "Awareness and coordination in shared workspaces," presented at the Proceedings of the 1992 ACM conference on Computer-supported cooperative work, Toronto, Ontario, Canada, 1992.
- [31] J. T. Biehl, *et al.*, "FASTDash: a visual dashboard for fostering awareness in software teams," presented at the Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, San Jose, California, USA, 2007.
- [32] S. Dustdar and H. Gall, "Process awareness for distributed software development in virtual teams," in *Euromicro Conference, 2002. Proceedings. 28th*, 2002, pp. 244-250.

- [33] C. Gutwin, *et al.*, "Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets, and Evaluation," presented at the Proceedings of HCI on People and Computers XI, 1996.
- [34] L. Aversano, *et al.*, "Managing coordination and cooperation in distributed software processes: the GENESIS environment," *Software Process: Improvement and Practice*, vol. 9, pp. 239-263, 2004.
- [35] G. Booch and A. W. Brown, "Collaborative Development Environments," in *Advances in Computers*. vol. Volume 59, ed: Elsevier, 2003, pp. 1-27.
- [36] B. Bruegge, *et al.*, "Sysiphus: Enabling informal collaboration in global software development," in *Global Software Engineering, 2006. ICGSE '06. International Conference on*, 2006, pp. 139-148.
- [37] M. Cataldo, *et al.*, "CAMEL: A Tool for Collaborative Distributed Software Design," in *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on*, 2009, pp. 83-92.
- [38] A. Feng, *et al.*, "Updating semantic information to support coordination in distributed software development environments," in *System Sciences, 1993, Proceeding of the Twenty-Sixth Hawaii International Conference on*, 1993, pp. 13-22 vol.4.
- [39] S. B. Fonseca, *et al.*, "Exploring the Relationship between Dependencies and Coordination to Support Global Software Development Projects," in *Global Software Engineering, 2006. ICGSE '06. International Conference on*, 2006, pp. 243-243.
- [40] R. E. Grinter, *et al.*, "The geography of coordination: dealing with distance in R&D work," presented at the Proceedings of the international ACM SIGGROUP conference on Supporting group work, Phoenix, Arizona, USA, 1999.
- [41] A. Sarma, *et al.*, "Categorizing the Spectrum of Coordination Technology," *Computer*, vol. 43, pp. 61-67, 2010.
- [42] W. Maalej, "Task-First or Context-First? Tool Integration Revisited," in *Automated Software Engineering, 2009. ASE '09. 24th IEEE/ACM International Conference on*, 2009, pp. 344-355.
- [43] D. G. Boyer, *et al.*, "Virtual community presence awareness," *SIGGROUP Bull.*, vol. 19, pp. 11-14, 1998.
- [44] M. K. Brown, *et al.*, "Choosing the Right Tools for Your Virtual Team: Evaluating Wikis, Blogs, and Other Collaborative Tools," in *Professional Communication Conference, 2007. IPCC 2007. IEEE International*, 2007, pp. 1-4.
- [45] A. Ferscha and C. Scheiner, "Collective choice in virtual teams," in *Enabling Technologies: Infrastructure for Collaborative Enterprises, 1999. (WET ICE '99) Proceedings. IEEE 8th International Workshops on*, 1999, pp. 96-101.
- [46] R. E. Kraut, *et al.*, "The use of visual information in shared visual spaces: informing the development of virtual co-presence," presented at the Proceedings of the 2002 ACM conference on Computer supported cooperative work, New Orleans, Louisiana, USA, 2002.
- [47] A. Powell, *et al.*, "Virtual teams: a review of current literature and directions for future research," *SIGMIS Database*, vol. 35, pp. 6-36, 2004.
- [48] C. R. B. D. Souza, *et al.*, "Toward visualization and analysis of traceability relationships in distributed and offshore software development projects," presented at the Proceedings of the 1st international conference on Software

- engineering approaches for offshore and outsourced development, Zurich, Switzerland, 2007.
- [49] G. O. Wiredu, "A framework for the analysis of coordination in global software development," presented at the Proceedings of the 2006 international workshop on Global software development for the practitioner, Shanghai, China, 2006.
- [50] M. Cataldo, *et al.*, "On Coordination Mechanisms in Global Software Development," in *Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference on*, 2007, pp. 71-80.
- [51] J. M. Bhat, *et al.*, "Overcoming Requirements Engineering Challenges: Lessons from Offshore Outsourcing," *Software, IEEE*, vol. 23, pp. 38-44, 2006.
- [52] "IBM Jazz Platform, <http://www-01.ibm.com/software/rational/jazz/> [May, 2013]."
- [53] "AgileZen, <http://www.agilezen.com/> [May, 2013]."
- [54] "Lucid Chart, <https://www.lucidchart.com/> [May, 2013]."
- [55] "MeetingSphere, <http://www.meetingsphere.com/> [May, 2013]."
- [56] "Microsoft Live Meeting, <http://support.microsoft.com/ph/925> [May, 2013]."
- [57] "Microsoft Project, <http://office.microsoft.com/en-us/project/> [May, 2013]."
- [58] "Team Foundation Server, <http://msdn.microsoft.com/en-us/vstudio/ff637362.aspx> [May, 2013]."
- [59] "Pidoco, <https://pidoco.com/> [May, 2013]."
- [60] "IBM Rational Cloud Services, <http://www-01.ibm.com/software/rational/info/cloud-services/> [May, 2013]."
- [61] "Cloud9 IDE, <https://c9.io/> [May, 2013]."
- [62] "Eclipse Orion, <http://www.eclipse.org/orion/> [May, 2013]."
- [63] "eXo Platform, <http://www.exoplatform.com/company/en/home> [May, 2013]."
- [64] A. Azeez, *et al.*, "Multi-tenant SOA Middleware for Cloud Computing," in *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*, ed: IEEE, 2010, pp. 458-465.
- [65] Z. Pervez, *et al.*, "Multi-Tenant, Secure, Load Disseminated SaaS Architecture," in *2010 The 12th International Conference on Advanced Communication Technology (ICACT)* vol. 1, ed: IEEE, 2010, pp. 214-219.
- [66] E. J. Domingo, *et al.*, "CLOUDIO: A Cloud Computing-Oriented Multi-tenant Architecture for Business Information Systems," in *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*, ed: IEEE, 2010, pp. 532-533.
- [67] J. H. Christensen, "Using RESTful web-services and cloud computing to create next generation mobile applications," presented at the Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications, Orlando, Florida, USA, 2009.
- [68] W. Qian and R. Deters, "SOA's Last Mile-Connecting Smartphones to the Service Cloud," in *IEEE International Conference on Cloud Computing, 2009. CLOUD '09*, ed: IEEE, 2009, pp. 80-87.
- [69] P. Papakos, *et al.*, "VOLARE: context-aware adaptive cloud service discovery for mobile systems," presented at the Proceedings of the 9th International Workshop on Adaptive and Reflective Middleware, Bangalore, India, 2010.
- [70] I. Giurgiu, *et al.*, "Calling the Cloud: Enabling Mobile Phones as Interfaces to Cloud Applications," in *Middleware 2009*. vol. 5896, J. Bacon and B. Cooper, Eds., ed: Springer Berlin / Heidelberg, 2009, pp. 83-102.

- [71] L. Qingfeng, *et al.*, "An Optimized Solution for Mobile Environment Using Mobile Cloud Computing," in *5th International Conference on Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09*, ed: IEEE, 2009, pp. 1-5.
- [72] S. V. Gogouvtis, *et al.*, "An Architectural Approach for Event-Based Execution Management in Service Oriented Infrastructures," in *2010 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, ed: IEEE, 2010, pp. 395-399.
- [73] A. Chazalet, "Service Level Agreements Compliance Checking in the Cloud Computing: Architectural Pattern, Prototype, and Validation," in *2010 Fifth International Conference on Software Engineering Advances (ICSEA)*, ed: IEEE, 2010, pp. 184-189.
- [74] H. Hyuck, *et al.*, "A RESTful Approach to the Management of Cloud Infrastructure," in *IEEE International Conference on Cloud Computing, 2009. CLOUD '09*, ed: IEEE, 2009, pp. 139-142.
- [75] H. Ludwig, *et al.*, "REST-based management of loosely coupled services," presented at the Proceedings of the 18th international conference on World wide web, Madrid, Spain, 2009.
- [76] E. M. Maximilien, *et al.*, "IBM altocumulus: a cross-cloud middleware and platform," presented at the Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications, Orlando, Florida, USA, 2009.
- [77] L. Shang, *et al.*, "Extending YML to Be a Middleware for Scientific Cloud Computing," in *Cloud Computing*. vol. 5931, M. Jaatun, *et al.*, Eds., ed: Springer Berlin / Heidelberg, 2009, pp. 662-667.
- [78] Y. Badr and G. Caplat, "Software-as-a-Service and Versionology: Towards Innovative Service Differentiation," in *2010 24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, ed: IEEE, 2010, pp. 237-243.
- [79] P. Belimpassakis and S. Moloney, "A platform for proving family oriented RESTful services hosted at home," *IEEE Transactions on Consumer Electronics*, vol. 55, pp. 690-698, 2009.
- [80] J. Du, *et al.*, "Towards secure dataflow processing in open distributed systems," presented at the Proceedings of the 2009 ACM workshop on Scalable trusted computing, Chicago, Illinois, USA, 2009.
- [81] I. Gorton, *et al.*, "Exploring Architecture Options for a Federated, Cloud-Based System Biology Knowledgebase," in *2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, ed: IEEE, 2010, pp. 218-225.
- [82] Y.-B. Han, *et al.*, "A Cloud-Based BPM Architecture with User-End Distribution of Non-Compute-Intensive Activities and Sensitive Data," *Journal of Computer Science and Technology*, vol. 25, pp. 1157-1167, 2010.
- [83] L. Jie, *et al.*, "eScience in the cloud: A MODIS satellite data reprojection and reduction pipeline in the Windows Azure platform," in *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, ed: IEEE, 2010, pp. 1-10.
- [84] K. Kim, "A model-driven workflow fragmentation framework for collaborative workflow architectures and systems," *Journal of Network and Computer Applications*, vol. In Press, Corrected Proof %U <http://www.sciencedirect.com/science/article/pii/S108480451100083X>.

- [85] Z. Liang-Jie and Z. Qun, "CCOA: Cloud Computing Open Architecture," in *IEEE International Conference on Web Services, 2009. ICWS 2009*, ed: IEEE, 2009, pp. 607-616.
- [86] B. Sodhi and T. V. Prabhakar, "Application architecture considerations for cloud platforms," in *2011 Third International Conference on Communication Systems and Networks (COMSNETS)*, ed: IEEE, 2011, pp. 1-4.
- [87] D. Kossmann, *et al.*, "An evaluation of alternative architectures for transaction processing in the cloud," presented at the Proceedings of the 2010 international conference on Management of data, Indianapolis, Indiana, USA, 2010.
- [88] Z. Wenjun, "2-Tier Cloud Architecture with maximized RIA and SimpleDB via minimized REST," in *2010 2nd International Conference on Computer Engineering and Technology (ICCET)* vol. 6, ed: IEEE, 2010, pp. V6-52-V6-56.
- [89] J. Schaffner, *et al.*, "Towards enterprise software as a service in the cloud," in *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW)*, ed: IEEE, 2010, pp. 52-59.
- [90] T. Wei-Tek, *et al.*, "Real-Time Service-Oriented Cloud Computing," in *2010 6th World Congress on Services (SERVICES-1)*, ed: IEEE, 2010, pp. 473-478.
- [91] B. Liver, *et al.*, "Privacy in Service Oriented Architectures: SOA Boundary Identity Masking for Enterprises," in *2010 IEEE 12th Conference on Commerce and Enterprise Computing (CEC)*, ed: IEEE, 2010, pp. 204-211.
- [92] X. Jia, "Google Cloud Computing Platform Technology Architecture and the Impact of Its Cost," in *2010 Second World Congress on Software Engineering (WCSE)* vol. 2, ed: IEEE, 2010, pp. 17-20.
- [93] S. Wei, *et al.*, "Design Aspects of Software as a Service to Enable E-Business through Cloud Platform," in *2010 IEEE 7th International Conference on e-Business Engineering (ICEBE)*, ed: IEEE, 2010, pp. 456-461.
- [94] R. Clarke, "User Requirements for Cloud Computing Architecture," in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, ed: IEEE, 2010, pp. 625-630.
- [95] L. Hyun Jung, *et al.*, "Technical Challenges and Solution Space for Developing SaaS and Mash-Up Cloud Services," in *IEEE International Conference on e-Business Engineering, 2009. ICEBE '09*, ed: IEEE, 2009, pp. 359-364.
- [96] C. Chapman, *et al.*, "Software architecture definition for on-demand cloud provisioning," *Cluster Computing*, pp. 1-22, 2011.
- [97] T. C. Chieu, *et al.*, "Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment," in *IEEE International Conference on e-Business Engineering, 2009. ICEBE '09*, ed: IEEE, 2009, pp. 281-286.
- [98] Z. Changli and Y. Maode, "Insurance-Based Cloud Computing-Architecture, Risk Analysis and Experiment," in *2010 International Conference on Computational Intelligence and Software Engineering (CiSE)*, ed: IEEE, 2010, pp. 1-4.
- [99] D. Bonetta and C. Pautasso, "Towards liquid service oriented architectures," presented at the Proceedings of the 20th international conference companion on World wide web, Hyderabad, India, 2011.
- [100] I. Brandic, *et al.*, "Service mediation and negotiation bootstrapping as first achievements towards self-adaptable grid and cloud services," presented at the

- Proceedings of the 6th international conference industry session on Grids meets autonomic computing, Barcelona, Spain, 2009.
- [101] J. C. Duenas, *et al.*, "System Virtualization Tools for Software Development," *Internet Computing, IEEE*, vol. 13, pp. 52-59, 2009.
 - [102] R. Martignoni, "Global Sourcing of Software Development - A Review of Tools and Services," in *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on*, 2009, pp. 303-308.
 - [103] I. Richardson, *et al.*, "Global Software Engineering: A Software Process Approach," in *Collaborative Software Engineering*, I. Mistr^v≠k, *et al.*, Eds., ed: Springer Berlin Heidelberg, 2010, pp. 35-56.
 - [104] R. T. Fielding, "Architectural styles and the design of network-based software architectures," University of California, Irvine, 2000.
 - [105] P. J. Leu and B. Bhargava, "Multidimensional Timestamp Protocols for Concurrency Control," *Software Engineering, IEEE Transactions on*, vol. SE-13, pp. 1238-1253, 1987.
 - [106] P. B. Kruchten, "The 4+1 View Model of architecture," *Software, IEEE*, vol. 12, pp. 42-50, 1995.
 - [107] A. E. Thijs Metsch, "Open Cloud Computing Interface - RESTful HTTP Rendering," 2011.
 - [108] "JSR 252, <http://www.jcp.org/en/jsr/detail?id=252> [May, 2013]."
 - [109] "Java Persistence APIs, <http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html> [May, 2013]."
 - [110] "Jersey, <http://jersey.java.net/> [May, 2013]."
 - [111] "Amazon Cloud Watch, <http://aws.amazon.com/cloudwatch/> [May, 2013]."
 - [112] "Amazon Elastic Load Balancing, <http://aws.amazon.com/elasticloadbalancing/> [May, 2013]."

7. Appendix

A.1. PTaaS APIs

This chapter explains the Application Programmable Interfaces (APIs) to access important features of PTaaS. We have followed Open Cloud Computing Interface (OCCI) RESTful HTTP Rendering specification [107]. APIs of the infrastructure are provided in a RESTful [104] style to provide easy access to platform resources. REST APIs are accessed using HTTP protocols and our decision to choose REST style will enable different types of application and clients to have access to platform resources through unified interface. In this chapter, we first provide a brief overview of OCCI specifications and its important components as described in [107]. After that we described APIs detailed of different platform interfaces to access to its features.

A.1.1. Overview of OCCI Specification

OCCI specifications follow a RESTful protocol and APIs to perform all types operations on resources. The specifications were originally developed to performed management operations on IaaS cloud. Later, the specifications were adapted to other types of cloud service model including PaaS and SaaS. The specifications adopt REST, therefore it is taking use of foundation protocols of HTTP. An implementation of the OCCI standards much adheres to following characteristics.

Path:

Every resource being exposed using OCCI specifications should have a uniquely identifiable resource path in form of URI. E.g. <http://resourcepath/servicename>.

Support filtering mechanism:

Resources that represent collections (of a kind), must be able to support filtering mechanism to identify which collection of the resource instances are target of a request. E.g. a resource associated with a uri [http://resourcepath/\(servicecollection\)/](http://resourcepath/(servicecollection)/) should be able to filter parameters passed in “(servicecollection)” should be able filter and find out which service collection of the request.

GET:

External applications and services should be able to retrieve the resource using HTTP GET methods on the URI.

POST:

External applications and services should be able to update a resource, create a new instance of a resource or update an instance of a resource using HTTP POST method on the URI.

POST (actionQuery = ?parameter1=value1, parameter1=value1.....):

External applications and services should be able to perform an action on a resource or on a collection of the resource using HTTP POST method with query parameters on a resource URI.

PUT:

External applications and services should be able to add a new resource or add a new collection of the resources using HTTP PUT method on a resource URI.

DELETE:

External applications and services should be able to delete a resource, remove a single entry for a collection of the resources and remove a subset from a collection of the resources using HTTP DELETE method on a resource URI.

Resource's Capabilities:

External applications and services should also be able to get, add and delete capabilities of the resources using using HTTP GET, POST and DELETE operations.

Additional Information and Parameters:

Additional information and parameters can be passed to resources through HTTP additional parameters in HTTP header either as a single attribute:

OCCI-Attribute: `occi.service.id = serviceId`

OCCI-Attribute: `occi.service.name = serviceName`

OCCI-Attribute: `occi.service.user = userId`

OCCI-Attribute: `occi.service.password = password`

or as a collection of attributes:

OCCI-Attribute: `occi.service.id = serviceName, occi.service.name = serviceName`

OCCI-Attribute: `occi.service.user = userId, occi.service.password = password`

Return Codes:

For return codes we are using HTTP return codes as described in [107]. Following is brief description of codes and their interpretation in context of our platform.

Response code: 200

Description: OK

Interpretation: Indicates that request was successful and response contains generated data.

Response code: 201

Description: OK

Interpretation: Indicates that request was successful and response contains http location header information to access generated data.

Response code: 202

Description: Accepted

Interpretation: Indicated that synchronous messaging request is accepted.

Response code: 204

Description: OK, without any return data

Interpretation: This indicates that request is successful but return collection does not contain any information.

Response code: 400

Description: Bad Request

Interpretation: This indicates if there occurs an error while parsing request and/or data that was sent with request. It applied both on individual data sets as well as on collections.

Response code: 401

Description: Unauthorized

Interpretation: This indicates that access denied because client does not have required permission or have provided wrong credentials.

Response code: 404

Description: Not found

Interpretation: This indicates that requested information was not found.

Response code: 409

Description: Conflict

Interpretation: This indicates that request contained data that resulted in a conflict with existing data because of resource internal constraints.

Response code: 410

Description: Gone

Interpretation: This indicates that requested resource instance does exist.

Response code: 500

Description: Internal server error

Interpretation: This indicated that client should restore itself to the state before request was sent.

Response code: 501

Description: Not Implemented

Interpretation: It indicates that requested functionality is not implemented yet.

Response code: 503

Description: Service Unavailable

Interpretation: This response code indicates that service is temporarily unavailable.

Return Values:

Resources can return information after processing a request. The information that is returned contains return code and additional information about the contents that is produced as a result of the response. Following is an example response returned from a resource.

```
<response>
  <responsecode>
    One of the response code as described above.
  </responsecode>
  <responsemessage>
    One of the response messages as described above.
  </responsemessage>
  <returndata>
    <contenttype>
      text/plain or XML
    </contenttype>
    <location>
      http://example.com/users/
```

```

</location>
<data>
    It contains data generated by the resource and returned as a
    result of an operation. In the subsequent parts of the report, we
    will only specify the information that is to be placed inside
    <data> tags.
</data>
<accessinformation>
    < id >tempid< /id >
    < password>password</password>
</accessinformation >
<additionalinformation>
</additionalinformation>
</returndata>
</response>

```

A.1.2. APIs to Access Platform Features

The APIs for the platform are defined in terms of the features that platform will support. Apart from registration of tenants and users, all other APIs require authentication credentials of tenants or users or both. To avoid redundant information in the document, we are describing authentication mechanism in a separate section. In other APIs where authentication is required, we will only refer if user authentication is required or tenant authentication is needed. APIs for accessing features are explained in terms of five sub-sections as described in Table 2.

Table 2: Legend of API Description Subsections

Section	Purpose
URI	It specifies address on which resource is hosted.
Method	One of the four HTTP methods: GET, PUT, POST or DELETE.
Functionality	It describes the functionality that is performed by the resource.
Inputs	It elaborates inputs that are sent to the resource.
Return Values	It explains outputs that are returned from the resource.

In order to avoid redundancy while describing return values of APIs, we only provide information that is tagged inside <data></data>. Other values in the response message are populated as described in subsection Return Values in section A.1.1.

A.1.2.1. Authentication Mechanism

The platform services need to authenticate incoming requests in order to verify whether request is coming from a valid source or not. The services perform two types of authentications; verify users while performing user specific operations and verify tenants while performing tenant specific operations. The information of users and tenants along with their security credentials is passed in HTTP request header. The HTTP request header expects four types of attributes as described in Table 3.

Table 3: Authentication Attributes

Attribute	Purpose
resource.tenant.id = tenantId	Contains tenant id
resource.user.id = userId	Contains user id
resource.tenant.password = tenantPassword	Contains tenant password or authentication code
resource.user.password = userPassword	Contains user password or authentication code

A.1.3. Platform APIs

This section defines APIs of the main components of the platform. We have divided APIs description into five sections: Tenant Management, User Management, Application/Tools/Services Management, Collaboration Service and Workflow Management.

A.1.3.1. APIs for Tenant Management

APIs explained in this section are associated with management of tenant on platform.

A.1.3.1.1. Get Information of an Existing Tenant

URI: (host)/tenant/(id)

Method: GET

Functionality: It returns tenants detail after verification of access id (user id) and password provided in HTTP header.

Input: Id of the tenant whose information is required.

Output: It returns following XML structure in data part of the return structure. Other parts of the return XML structure will be same as explained in Return Values in Section A.1.1.

```
<data><tenant>
  <id>tenantid</id>
  <name>tenantname</name>
  <temppassword> this field will be empty</ temppassword >
  <password>this field will be empty</password>
  <confirmpassword>this field will be empty</confirmpassword>
  <email>tenant@email.com</email>
  <webaddress>www.exampletenant.com</webaddress>
  <physicaladdress></physicaladdress>
  <status>Y for active, N for inactive</status>
  <sharingpreference>E for exclusive, S for shared</sharingpreference>
  <locationpreference>
    Information of geographic region where tenant want to store its data
    and services.
  </locationpreference>
  <encryption>Y for yes, N for no</encryption>
</tenant></data>
```

A.1.3.1.2. Register a New Tenant

URI: (host)/tenant/(tenantInformation)

Method: PUT

Functionality: It registers a new tenant with the platform. After first time registration an auto generated email will be sent to tenant with a temporary password. Tenant need to update its password after that in order to have access to platform resources.

Input: Tenant information is passed in following structure.

```

<tenant>
  <name>tenantname</name>
  <temppassword> this field will be empty</ temppassword >
  <password>this field will be empty</password>
  <confirmpassword>this field will be empty</confirmpassword>
  <email>tenant@email.com</email>
  <webaddress>www.exampletenant.com</webaddress>
  <physicaladdress></physicaladdress>
  <status>Y for active, N for inactive</status>
  <sharingpreference>E for exclusive, S for shared</sharingpreference>
  <locationpreference>
    Information of geographic region where tenant want to store its data
    and services. E for Europe, US for united states and N for not
    applicable.
  </locationpreference>
  <encryption>Y for yes, N for no</encryption>
</tenant>

```

Output: It returns XML structure as explained in Return Values in Section A.1.1 along with additional information as described below.

```

<data>
  <tenant>
    <id>tenantid</id>
  </tenant>
</data>

```

A.1.3.1.3. Update Information of an Existing Tenant

URI: (host)/tenant/(tenantInformation)

Method: POST

Functionality: It updates information of the tenant already registered with the platform.

Input: Tenant information is passed in following structure. Tenant's id and password fields are mandatory to be filled in. Only filled fields will be updated. Empty fields will be ignored and will retain old values in the platform.

```

<data></tenant>
  <id>tenantid</id>
  <name>tenantname</name>
  <temppassword> this field will be empty</ temppassword >
  <password>this field contains tenant's password</password>
  <confirmpassword>this field will be empty</confirmpassword>
  <email>tenant@email.com</email>
  <webaddress>www.exampletenant.com</webaddress>
  <physicaladdress></physicaladdress>
  <status>Y for active, N for inactive</status>
  <sharingpreference>E for exclusive, S for shared</sharingpreference>
  <locationpreference>
    Information of geographic region where tenant want to store its data
    and services. E for Europe, US for united states and N for not
    applicable.
  </locationpreference>
  <encryption>Y for yes, N for no</encryption>

```


</tenant></data>

Output: It returns XML structure as explained in Return Values in Section A.1.1.

A.1.3.1.4. Delete a Tenant

URI: (host)/tenant/(tenantInformation)

Method: DELETE

Functionality: It inactivates the tenant that is already registered with platform.

Input: Tenant information is passed in following structure. Tenant's id and password fields are mandatory to be filled in.

```
<data></tenant>
  <id>tenantid</id>
</tenant></data>
```

Output: It returns XML structure as explained in Return Values in Section A.1.1.

A.1.3.2. User Management

APIs explained in this section are associated with management of users on platform. Users email address is treated as id in the platform.

A.1.3.2.1. Get Information of an Existing User

URI: (host)/user/(email)

Method: GET

Functionality: It returns user's detail after verification of user email and password provided in HTTP header.

Input: Email of the user whose information is required.

Output: It returns following XML structure in data part of the return structure. Other parts of the return XML structure will be same as explained in Return Values in Section A.1.1.

```
<data><user>
  <email>useremail@email.com</email >
  <name>username</name>
  <temppassword> this field will be empty</ temppassword >
  <password>this field will be empty</password>
  <confirmpassword>this field will be empty</confirmpassword>
  <address>user address</address>
  <status>Y for active, N for inactive</status>
</user></data>
```

A.1.3.2.2. Register a New User

URI: (host)/user/(userInformation)

Method: PUT

Functionality: It registers a new user with the platform. After first time registration an auto generated email will be sent to users with a temporary password. User need to update her password after that in order to have access to platform resources.

Input: User information is passed in following structure.

```
<data><user>
  <email>useremail@email.com</email >
  <name>username</name>
  <temppassword> this field will be empty</ temppassword >
  <password>this field will be empty</password>
  <confirmpassword>this field will be empty</confirmpassword>
  <address>user address</address>
```

<status>Y for active, N for inactive</status>
</user></data>

Output: It returns XML structure as explained in Return Values in Section A.1.1.

A.1.3.2.3. Update Information of an Existing User

URI: (host)/user/(userInformation)

Method: POST

Functionality: It updates information of the user already registered with the platform.

Input: User's information is passed in following structure. User's email and password fields are mandatory to be filled in. Only filled fields will be updated. Empty fields will be ignored and will retain old values in the platform.

```
<data><user>
  <email>useremail@email.com</email >
  <name>username</name>
  <temppassword>
    If user is updating its auto generated system password then this field is
    mandatory.
  </ temppassword >
  <password>User's password</password>
  <confirmpassword>
    If user is updating its auto generated system password then this field is
    mandatory.
  </confirmpassword>
  <address>user address</address>
  <status>Y for active, N for inactive</status>
```

```
</user></data>
```

Output: It returns XML structure as explained in Return Values in Section A.1.1.

A.1.3.2.4. Delete a User

URI: (host)/user/(userInformation)

Method: DELETE

Functionality: It inactivates the user who is already registered with platform.

Input: User's information is passed in following structure. User's email and password fields are mandatory to be filled in.

```
<data><user>
  <email>useremail@email.com</email >
</user></data>
```

Output: It returns XML structure as explained in Return Values in Section A.1.1.

A.1.3.3. Registration of Applications, Tools and Services

APIs explained in this section are associated with management of applications, tools and services on the platform.

A.1.3.3.1. Get Information of a Registered Application, Tool or Service

URI: (host)/apptoolservice/(id)

Method: GET

Functionality: It returns information of already registered application, tool or services in the platform. Tenant verification is required in order to access the API.

Input: Uniquely identifiable information about application tool or service.

Output: It returns following XML structure in data part of the return structure. Other parts of the return XML structure will be same as explained in Return Values in Section A.1.1.

```
<data><apptoolservice>
  <apptoolserviceid>Service identity</apptoolserviceid>
  <servicename>Service name</servicename >
  <servicetype>
    Service type. W for a web based application, MW for application that
    supports both mobile and web based interfaces. D for a desktop
    application. S for a web service that performs some business logic can
    be integrated with other applications and services.
  </servicetype>
  <vmtemplateid>
    Uniquely identifiable virtual machine template that is stored in
    underlying IaaS cloud with application, tools and/or services stored on
    it.
  </vmtemplateid>
  <executablebundle>
    Executable file of application tool or service (.exe, .war, .jar etc) that
    can be deployed on cloud. This is used when vmtemplate hosting
    service is not available on underlying IaaS cloud.
  </executablebundle/>
  <deploymentinstallationscript>
    <script>
      Deployment or installation scripts that can be used to install or
      deploy applications, tools and services.
    </script>
    <type>
      Deployment script type. A for Ant and M for Maven.
    </type>
  </deploymentinstallationscript>
  <interfaces>
    Information on interfaces of services. E.g. Web Service Descriptive
    Language (WSDL).
  </interfaces>
  <prerequisiteservices>
    Comma separated list of service IDs that should be invoked before this
    service.
  </prerequisiteservices>
</apptoolservice></data>
```

A.1.3.3.2. Registered a new Application, Tool or Service

URI: (host)/apptoolservice/(information)

Method: PUT

Functionality: It registers a new application, tool or service with the platform. Tenant verification is required in order to access the API.

Input: Following information needs to be provided to do registration.

```
<apptoolservice>
  <servicename>Service name</servicename >
  <servicetype>
```

Service type. W for a web based application, MW for application that supports both mobile and web based interfaces. D for a desktop application. S for a web service that performs some business logic can be integrated with other applications and services.

</servicetype>

<vmtemplateid>

Uniquely identifiable virtual machine template that is stored in underlying IaaS cloud with application, tools and/or services stored on it.

</vmtemplateid>

<executablebundle>

Executable file of application tool or service (.exe, .war, .jar etc) that can be deployed on cloud. This is used when vmtemplate hosting service is not available on underlying IaaS cloud.

</executablebundle/>

<deploymentinstallationscript>

<script>

Deployment or installation scripts that can be used to install or deploy applications, tools and services.

</script>

<type>

Deployment script type. A for Ant and M for Maven.

</type>

</deploymentinstallationscript>

<interfaces>

Information on interfaces of services. E.g. Web Service Descriptive Language (WSDL).

</interfaces>

<prerequisiteservices>

Comma separated list of service IDs that should be invoked before this service.

</prerequisiteservices>

</apptoolservice>

Output: It returns XML document structures in the same way as explained in Section A.1.1 containing additional information of the service id.

<data>

<apptoolservice>

<apptoolserviceid>

Id of the registered application, tool or service

</apptoolserviceid>

</apptoolservice>

</data>

A.1.3.3.3. Update a Registered Application, Tool or Service

URI: (host)/apptoolservice/(information)

Method: POST

Functionality: It updates information of an existing application, tool or service in the platform. Tenant verification is required in order to access the API. Only <apptoolserviceid> and at least of the other tag is mandatory to have a value. If other

tags contain a valid value it is updated, if they do not have a valid value or are empty, the value that is already there is platform is retained.

Input: Following information needs to be provided to do registration.

<apptoolservice>

<apptoolserviceid>Service Identify</apptoolserviceid>

<servicename>Service name</servicename >

<servicetype>

Service type. W for a web based application, MW for application that supports both mobile and web based interfaces. D for a desktop application. S for a web service that performs some business logic can be integrated with other applications and services.

</servicetype>

<vmtemplateid>

Uniquely identifiable virtual machine template that is stored in underlying IaaS cloud with application, tools and/or services stored on it.

</vmtemplateid>

<executablebundle>

Executable file of application tool or service (.exe, .war, .jar etc) that can be deployed on cloud. This is used when vmtemplate hosting service is not available on underlying IaaS cloud.

</executablebundle/>

<deploymentinstallationscript>

<script>

Deployment or installation scripts that can be used to install or deploy applications, tools and services.

</script>

<type>

Deployment script type. A for Ant and M for Maven.

</type>

</deploymentinstallationscript>

<interfaces>

Information on interfaces of services. E.g. Web Service Descriptive Language (WSDL).

</interfaces>

<prerequisiteservices>

Comma separated list of service IDs that should be invoked before this service.

</prerequisiteservices>

</apptoolservice>

Output: It returns XML document structures in the same way as explained in Section A.1.1.

A.1.3.3.4. Delete an Application, Tool or Service

URI: (host)/apptoolservice/(information)

Method: DELETE

Functionality: It unregisters an existing application, tool or service in the platform. Tenant verification is required in order to access the API. Only service identity is needed to be provided.

Input: Following information needs to be provided to do registration.

```
<apptoolservice>
  <apptoolserviceid>Service Identify</apptoolserviceid>
</apptoolservice>
```

Output: It returns XML document structures in the same way as explained in Section A.1.1.

A.1.3.4. Collaboration APIs

This section explains APIs to support collaboration among applications hosted on the platform.

A.1.3.4.1. Initiate Collaboration Activity

URI: (host)/ collaboration/(information)

Method: PUT

Functionality: This API supports initialization of a new collaboration activity.

Input: Following information needs to be provided.

```
<collaboration>
  <name>Name</servicename >
  <type>
    A for asynchronous collaboration and S for synchronous collaboration.
  </type>
</collaboration>
```

Output: It returns XML document structures in the same way as explained in Section A.1.1 along with containing additional information of the initiated collaboration id.

```
<data>
  <collaboration>
    <id>
      Id of the newly instantiated collaboration activity.
    </id>
  </collaboration>
</data>
```

A.1.3.4.2. Register Application, Tool or Service in a Collaboration Activity

URI: (host)/collaboration/(information)

Method: POST(if information = register)

Functionality: This API supports addition of new clients application, tool and service into collaboration. If a client application, too and service is already register; its information is updated with new information passed in parameters.

Input: Following information needs to be provided.

```
<collaboration>
  <id>collaboration id</id>
  <client>
    <id>Application, tool or service id</id >
    <iterfacetypes>
      R for rest, S for SOAP Web Service.
    </iterfacetypes>
    <callbackmethod>
      <information>
        It contains call method address and signature. For
        example in case of REST it contains resource URI.
      <information>
```

```
        </callbackmethod>
    </client>
</collaboration>
```

Output: It returns XML document structures in the same way as explained in Section A.1.1.

A.1.3.4.3. Unregister Application, Tool or Service in a Collaboration Activity

URI: (host)/collaboration/(information)

Method: POST(if information = unregister)

Functionality: This API supports initialization of a new collaboration activity. This API can also accept list of clients.

Input: Following information needs to be provided.

```
<collaboration>
    <id>collaboration id</id>
    <client>
        <id>Application, tool or service id</id >
    </client>
</collaboration>
```

Output: It returns XML document structures in the same way as explained in Section A.1.1.

A.1.3.4.4. Post Data in a Collaboration Activity

URI: (host)/collaboration/(information)

Method: POST(if information = postdata)

Functionality: This API supports adding data to collaboration activity.

Input: Following information needs to be provided.

```
<collaboration>
    <id>collaboration id</id>
    <data>
        It contains collaboration data.
    </data>
</collaboration>
```

Output: It returns XML document structures in the same way as explained in Section A.1.1.

A.1.3.4.5. Get Data from a Collaboration Activity

URI: (host)/collaboration/(information)

Method: GET(if information = getdata)

Functionality: This API provides latest copy of data placed in a collaboration activity.

Input: Following information needs to be provided.

```
<collaboration>
    <id>collaboration id</id>
</collaboration>
```

Output: It returns XML document structures in the same way as explained in Section A.1.1.

```
<data>
    <collaborationdata>
        <type>
```

Data format: T for text, X for XML document and O for all other types of data.

```

        </type>
        <data>
        It contains collaboration data.
        </data>
    </collaborationdata>
</data>

```

A.1.3.4.6. Terminate a Collaboration Activity

URI: (host)/collaboration/(information)

Method: DELETE

Functionality: This API terminates a collaboration activity. To terminate collaboration, tenant authentication parameters passed in HTTP header for authorization should match with one passed for creating an activity

Input: Following information needs to be provided.

```

<collaboration>
    <id>collaboration id</id>
</collaboration>

```

Output: It returns XML document structures in the same way as explained in Section A.1.1.

A.1.3.5. Workflow Management

This section provides overview of APIs to define and manage process workflows. The APIs are classified into three sub groups describing APIs associated with creation and management of actual process flows, APIs to add tools to workflow nodes and APIs to attach tenants to individual processing points of workflows.

A.1.3.5.1. Process Workflow Management

A.1.3.5.1.1. Create Process Workflow

URI: (host)/processworkflow/

Method: PUT

Functionality: This API supports initialization of a new process workflow.

Input: Following information needs to be provided.

```

<processworkflow>
    <name>Name</name>
</processworkflow >

```

Output: It returns XML document structures in the same way as explained in Section A.1.1 along with containing additional information of the newly created workflow id.

```

<data>
    <processworkflow>
        <id>Workflow id</id>
    </processworkflow >

```

```

</data>

```

A.1.3.5.1.2. Get Process Workflow

URI: (host)/processworkflow/

Method: GET

Functionality: This API supports to get information of an existing workflow.

Input: Following information needs to be provided.


```
<processworkflow>
  <id>Workflow id</id>
</processworkflow >
```

Output: It returns XML document structures in the same way as explained in Section A.1.1 along with containing additional information of the newly created workflow id.

```
<data>
  <processworkflow>
    <id>Workflow id</id>
    <name>Workflow name</name>
  </processworkflow >
</data>
```

A.1.3.5.1.3. Update Process Workflow

URI: (host)/processworkflow/

Method: POST

Functionality: This API provides update operation on an existing workflow.

Input: Following information needs to be provided.

```
<processworkflow>
  <id>Workflow id</id>
  <name>Name</name >
</processworkflow >
```

Output: It returns XML document structures in the same way as explained in Section A.1.1.

A.1.3.5.1.4. Delete Process Workflow

URI: (host)/processworkflow/

Method: DELETE

Functionality: This API supports initialization of a new process workflow.

Input: Following information needs to be provided.

```
<processworkflow>
  <id>Workflow id</id>
</processworkflow >
```

Output: It returns XML document structures in the same way as explained in Section A.1.1.

A.1.3.5.2. Node Managements on a Process Workflow

A.1.3.5.2.1. Add Node in a Process Workflow

URI: (host)/processworkflow/node

Method: PUT

Functionality: This API supports initialization of a new node in the process workflow.

Input: Following information needs to be provided.

```
<processworkflownode>
  <processworkflow>
    <id>workflow id</id>
  </processworkflow>
  <name>
```

```

        Name of the process workflow node.
    </name>
    <parentnode>
        Parents node id.
    </parentnode>
    <preceedingnode>
        <id>
            Id of the preceeding node of workflow.
        </id>
    <proceedingnode>
    <proceedingnode>
        <id>
            Id of the preceeding node of workflow.
        </id>
    <proceedingnode>
</processworkflownode>

```

Output: It returns XML document structures in the same way as explained in Section A.1.1 along with containing additional information of the newly created node's id.

```

<data>
    <processworkflownode>
        <id>
            Node id
        </id>
    </processworkflownode>
</data>

```

A.1.3.5.2.2. Get Node in a Process Workflow

URI: (host)/processworkflow/node

Method: GET

Functionality: This API supports get information of a node in the process workflow.

Input: Following information needs to be provided.

```

<processworkflownode>
    <id>
        Node id.
    </id>
</processworkflownode>

```

Output: It returns XML document structures in the same way as explained in Section A.1.1 along with containing additional information of the node.

```

<data><processworkflownode>
    <processworkflow>
        <id>workflow id</id>
    </processworkflow>
    <id>
        Node id.
    </id>
    <name>
        Name of the process workflow node.
    </name>

```

```

    <parentnode>
        Parents node id.
    </parentnode>
    <preceedingnode>
        <id>
            Id of the preceeding node of workflow.
        </id>
    </preceedingnode>
    <proceedingnode>
        <id>
            Id of the preceeding node of workflow.
        </id>
    </proceedingnode>
</processworkflownode></data>

```

A.1.3.5.2.3. Update Node in a Process Workflow

URI: (host)/processworkflow/node

Method: POST

Functionality: This API supports update of a node in the process workflow.

Input: Following information needs to be provided.

```

<processworkflownode>
    <processworkflow>
        <id>workflow id</id>
    </processworkflow>
    <id>
        Node id
    </id>
    <name>
        Name of the process workflow node.
    </name>
    <parentnode>
        Parents node id.
    </parentnode>
    <preceedingnode>
        <id>
            Id of the preceeding node of workflow.
        </id>
    </preceedingnode>
    <proceedingnode>
        <id>
            Id of the preceeding node of workflow.
        </id>
    </proceedingnode>
</processworkflownode>

```

Output: It returns XML document structures in the same way as explained in Section A.1.1.

A.1.3.5.2.4. Delete Node in a Process Workflow

URI: (host)/processworkflow/node

Method: DELETE

Functionality: This API removes node from the process workflow.

Input: Following information needs to be provided.

```
<processworkflownode>
  <id>
    Node id.
  </id>
</processworkflownode>
```

Output: It returns XML document structures in the same way as explained in Section A.1.1.

A.1.3.5.3. Managing Tools assigned to the Workflow Node

A.1.3.5.3.1. Assign Application, Tools or Services to a Node

URI: (host)/processworkflow/tool

Method: PUT

Functionality: This API assigns the list of tools to a node in the process workflow.

Input: Following information needs to be provided.

```
<processworkflow>
  <processworkflownode>
    <id>workflow id</id>
  </processworkflownode>
  <apptoolservicelist>
    <apptoolservice>
      <id>application, tool or service id 1</id>
    </apptoolservice>
    <apptoolservice>
      <id>application, tool or service id 2</id>
    </apptoolservice>
    .....
    .....
    .....
  </apptoolservicelist>
</processworkflownode>
```

Output: It returns XML document structures in the same way as explained in Section A.1.1.

A.1.3.5.3.2. Get Application, Tools or Services to a Node

URI: (host)/processworkflow/tool/

Method: GET

Functionality: This API returns a list of applications, tools or services registered assigned to a node of the process workflow.

Input: Following information needs to be provided.

```
<processworkflow>
  <processworkflownode>
    <id>workflow id</id>
  </processworkflownode>
</processworkflownode>
```

Output: It returns XML document structures in the same way as explained in Section A.1.1 along with additional information of tools that attached to a processing node.

```
<data><processworkflow>
  <apptoolservicelist>
    <apptoolservice>
      <id>application, tool or service id 1</id>
    </apptoolservice>
    <apptoolservice>
      <id>application, tool or service id 2</id>
    </apptoolservice>
    .....
    .....
    .....
  </apptoolservicelist>
</processworkflow></data>
```

A.1.3.5.3.3. Update Application, Tools or Services to a Node

URI: (host)/processworkflow/tool

Method: POST

Functionality: This API updates the list of applications, tools and services assigned to a node in the process workflow.

Input: Following information needs to be provided.

```
<processworkflow>
  <processworkflownode>
    <id>workflow id</id>
  </processworkflownode>
  <apptoolservicelist>
    <apptoolservice>
      <id>application, tool or service id 1</id>
    </apptoolservice>
    <apptoolservice>
      <id>application, tool or service id 2</id>
    </apptoolservice>
    .....
    .....
    .....
  </apptoolservicelist>
</processworkflownode>
```

Output: It returns XML document structures in the same way as explained in Section A.1.1.

A.1.3.5.3.4. Update Application, Tools or Services to a Node

URI: (host)/processworkflow/tool

Method: DELETE

Functionality: This API deletes the list of applications, tools and services assigned to a node in the process workflow.

Input: Following information needs to be provided.

```
<processworkflow>
  </processworkflownode>
  <id>workflow id</id>
```

```
</processworkflownode>
</processworkflownode>
```

Output: It returns XML document structures in the same way as explained in Section A.1.1.

A.1.3.5.4. Managing Tenants assignment to the Nodes in the Workflow

A.1.3.5.4.1. Assign Tenants to a Node

URI: (host)/processworkflow/tenants

Method: PUT

Functionality: This API assigns the list of tenant to a node in the process workflow.

Input: Following information needs to be provided.

```
<processworkflow>
  <processworkflownode>
    <id>workflow id</id>
  </processworkflownode>
  <tenantlist>
    <tenant>
      <id>tenant id 1</id>
    <tenant>
    <tenant>
      <id>tenant id 2</id>
    <tenant>
    .....
    .....
    .....
  </tenantlist>
</processworkflownode>
```

Output: It returns XML document structures in the same way as explained in Section A.1.1.

A.1.3.5.4.2. Get Tenants Assigned to a Node

URI: (host)/processworkflow/tenants

Method: GET

Functionality: This API returns the list of tenant to a node in the process workflow.

Input: Following information needs to be provided.

```
<processworkflow>
  <processworkflownode>
    <id>workflow id</id>
  </processworkflownode>
</processworkflownode>
```

Output: It returns XML document structures in the same way as explained in Section A.1.1 along with information of tenants.

```
<data>
  <tenantlist>
    <tenant>
```

```

        <id>tenant id 1</id>
    <tenant>
    <tenant>
        <id>tenant id 2</id>
    <tenant>
    .....
    .....
    .....
</tenantlist>
</data>

```

A.1.3.5.4.3. Update Tenants Assigned to a Node

URI: (host)/processworkflow/tenants

Method: POST

Functionality: This API updates the list of tenant to a node in the process workflow.

Input: Following information needs to be provided.

```

<processworkflow>
    <processworkflownode>
        <id>workflow id</id>
    </processworkflownode>
    <tenantlist>
        <tenant>
            <id>tenant id 1</id>
        <tenant>
        <tenant>
            <id>tenant id 2</id>
        <tenant>
        .....
        .....
        .....
    </tenantlist>
</processworkflownode>

```

Output: It returns XML document structures in the same way as explained in Section A.1.1.

A.1.3.5.4.4. Delete Tenants Assigned to a Node

URI: (host)/processworkflow/tenants

Method: DELETE

Functionality: This API deletes the list of tenant to a node in the process workflow.

Input: Following information needs to be provided.

```

<processworkflow>
    <processworkflownode>
        <id>workflow id</id>
    </processworkflownode>
</processworkflownode>

```

Output: It returns XML document structures in the same way as explained in Section A.1.1.

A.1.3.6. APIs for Supporting Traceability

This section explains the APIs for providing traceability support by the platform.

A.1.3.6.1. Register Artifact for Traceability and Get Identifier

URI: (host)/ traceabilityartifact/(artifactInformation)

Method: PUT

Functionality: This API registers a new artifact for traceability in the platform. After successful registration of the artifact, a unique identity is returned which is used for subsequent traceability operations.

Input: Artifact information is passed in following structure.

```
<artifact>
  <name>
    Artifact name that is uniquely identifiable by the application that is
    registering it for traceability.
  </name>
  <type>
    Specifies type of artifact. E.g. Design document, text document etc.
  </type>
  <category>
    This field contains category of the artifact. For example, design
    document can be any of the class diagram type, component diagram
    type, sequence diagram type, collaboration diagram type etc.
  </category>
  <applicationid>
    Identity of the application/tool registering artifact for traceability.
  </applicationid>
</artifact >
```

Output: It returns XML structure as explained in Return Values in Section A.1.1 along with additional information as described below.

```
<data>
  <artifact>
    <id>artifactId</id>
  </artifact>
</data>
```

A.1.3.6.2. Get Identifier of an already Registered Artifact

URI: (host)/ traceabilityartifact/(artifactInformation)

Method: GET

Functionality: This API returns identifier of an already registered artifact.

Input: Artifact information is passed in following structure.

```
<artifact>
  <name>
    Artifact name that is uniquely identifiable by the application that is
    registering it for traceability.
  </name>
  <applicationid>
    Identity of the application/tool registering artifact for traceability.
  </applicationid>
</artifact >
```


Output: It returns XML structure as explained in Return Values in Section A.1.1 along with additional information as described below.

```
<data>
  <artifact>
    <id>artifactId</id>
  </artifact>
</data>
```

A.1.3.6.3. Update an already Registered Artifact

URI: (host)/ traceabilityartifact/(artifactInformation)

Method: POST

Functionality: This API updates name of an already registered artifact.

Input: Artifact information is passed in following structure.

```
<artifact>
  <id>artifactId</id>
  <name>
    Artifact name that is uniquely identifiable by the application that is
    registering it for traceability.
  </name>
  <applicationid>
    Identity of the application/tool registering artifact for traceability.
  </applicationid>
</artifact >
```

Output: It returns XML structure as explained in Return Values in Section A.1.1.

A.1.3.6.4. Delete a Registered Artifact

URI: (host)/ traceabilityartifact/(artifactInformation)

Method: DELETE

Functionality: This API deletes an already registered artifact after verifying application identity.

Input: Artifact information is passed in following structure.

```
<artifact>
  <id>artifactId</id>
  <applicationid>
    Identity of the application/tool that registered artifact for traceability.
  </applicationid>
</artifact>
```

Output: It returns XML structure as explained in Return Values in Section A.1.1.

A.1.3.6.5. Register Callback Method for Traceability

URI: (host)/traceabilitycallback/(methodInformation)

Method: PUT

Functionality: This API registers a new call method to notify application/tools whenever there is an update in the linked document.

Input: Method information is passed in following structure.

```
<callbackmethod>
  <applicationtoolidentity>
    Unique identity of the application or tool (as described in section
    A.1.3.3.2).
  </ applicationtoolidentity >
  <artifactid>
```

This is a unique identity of the artifact registered in the platform for traceability (as explained in section A.1.3.6.1).

</artifactid>

<callbackmethod>

<type>

This field specifies type of callback method. It is either of two types: remote method invocation (RMI) and REST.

</type>

<address>

This field specifies address of callback method

</address>

<signature>

This field specifies name of callback method.

</signature>

</callbackmethod>

</callbackmethod>

Output: It returns XML structure as explained in Return Values in Section A.1.1.

<data>

<callbackmethod>

<id>Unique callbackmethod id</id>

</callbackmethod>

</data>