Design and Implementation of Peer Collaboration Service Framework on Cloud

by

Dong Dong

Bachelor of Software Engineering,
Jilin University, China, 2008

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Computer Science

In the Graduate Academic Unit of Computer Science

Supervisor(s):	Weichang Du, Ph.D., Computer Science
Examining Board:	Wei Song, Ph.D., Computer Science, Chair
	Huajie Zhang, Ph.D., Computer Science
External Examiner:	Fan-Rui Meng, Ph.D., Forestry & Environmental Management

This thesis is accepted by the

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

May, 2013

©Dong Dong, 2013

Abstract

Most of the key tasks or work in today's business are strongly related to collaboration. One of the important reasons that people collaborate is to complete a task which is hard to be done by individuals independently. With the prevalence of the Internet and mobile devices accessing the Internet with high-bandwidth network, it is easier for people in different locations to form groups anywhere and anytime. However, there are few methods to manage these dynamic web based collaborations. This thesis describes implementation of a framework named "Peer Collaboration Service Framework" providing a systematic approach to create and manage network based dynamic peer collaborations. The framework consists of three layers: (1) collaboration as a service layer, consisting of services to generate peer collaborations; (2) collaboration service layer, consisting of services running at the back end of collaborations to support them; (3) collaboration instance layer, supporting the generated collaboration application instances used by participants. This framework is implemented on Amazon EC2 cloud computing platform and employs several other web services offered by Amazon. A case study on collaborative software testing applications and experiments are also presented in the thesis.

Acknowledgements

I would like to express my great thanks to my supervisor, Dr. Weichang Du, who gave me his patient, encouragement, guidance, and comments on almost every detail and aspect of my thesis.

I would also like to thank to the University of New Brunswick, the faculty of computer science, for offering me all the facilities and resources that necessary to complete my degree.

Finally, special thanks to my family far away back in China, for their sincere support, encouragement and love. And also special thanks to all my friends, for spending time with me on holidays and bringing me happiness.

Table of Contents

Abstract ii

Acknowledgments iii

Table of Contents viii

List of Tables ix

List of Figures xii

Abbreviations xiii

1 Introduction 1

1.1 Motivation 1

1.2 Thesis Overview 3

1.2.1 Objectives 3

1.2.2 Framework Overview 4

1.3 Thesis Structure 5

2 Background and Related Work 6

2.1 Background 7

2.1.1 Conventional Collaboration Technologies and Systems 7

2.1.1.1 Virtual Team 7

2.1.1.2 Computer Supported Cooperative Work 8

2.1.1.3 Groupware 9

2.1.1.4 Network Based Collaboration Technology 12
2.1.2 Collaboration Enabling Technologies 13
2.1.2.1 Cloud Computing 13
2.1.2.2 SOA and Web Service 17
2.1.2.3 Workflow Management 20
2.2 Related Work 24
2.2.1 SOMICAS 25
2.2.1.1 Collaboration Client Layer 25
2.2.1.2 Collaboration Server Layer 28
2.2.1.3 Medical Image Server Layer 28

3 Framework Design **29**
3.1 Overview 29
3.2 Overview of Services 30
3.3 Requirements 31
3.4 Collaboration as a Service Layer 32
3.5 Collaboration Service Layer 34
3.6 Collaboration Service Instance Layer 37
3.7 Collaboration Lifecycle 38

4 Framework Implementation **41**
4.1 Overview 41
4.1.1 PCSF Architecture 41
4.2 Data Model Design and Implementation 43
4.2.1 Domain Creator 44
4.2.2 Domain Participant 45
4.2.3 Domain Collaboration 46
4.3 CaaS Implementation 47

4.3.1 Creation Service 47
4.3.1.1 Design 48
4.3.1.2 Implementation 49
4.3.2 Instance Management Service 52
4.3.2.1 Design 52
4.3.2.2 Implementation 53
4.4 Collaboration Services Implementation 53
4.4.1 Registration Service 54
4.4.1.1 Design 54
4.4.1.2 Implementation 55
4.4.2 Schedule and Coordinate Service 57
4.4.2.1 Design 58
4.4.2.2 Implementation 61
4.4.3 Data Transfer Service 63
4.4.3.1 Design 64
4.4.3.2 Implementation 65
4.4.4 Monitor and Control Service 66
4.4.4.1 Design 66
4.4.4.2 Implementation 67

5 Case Study: Collaborative Testing 69

5.1 Collaborative Testing and Application 70
5.2 CaaS Realization 72
5.3 Amazon EC2 Implementation 76
5.4 Experiments 78
5.4.1 Scenario 1: Create A Collaboration 78
5.4.2 Scenario 2: Generate A Collaboration Instance 80
5.4.3 Scenario 3: Participant Registration 84

5.4.4 Scenario 4: Run Collaboration Instance 86
5.4.5 Scenario 5: Submit Task 88
5.4.6 Scenario 6: Multiple Collaborations and Instances 90

6 Evaluation **94**
6.1 Functionality 94
6.1.1 What Collaboration Systems Support 94
6.1.2 What CaaS Supports 95
6.2 Quality 96
6.2.1 Reliability 96
6.2.2 Security 96
6.2.3 Performance 97
6.2.4 Modifiability 97
6.2.5 Usability 98

7 Conclusion **99**
7.1 Summary 99
7.2 Contributions 100
7.3 Future Work 101

Bibliography **109**

A CaaS Implementation Codes **110**
A.1 Creation Service 110
A.2 Instance Management Service 123

B CS Implementation Codes **127**
B.1 Registration Service 127
B.2 Schedule and Coordination Service 133
B.3 Data Transfer Service 139

B.4 Monitor and Control Service 141

C Constants Definition **147**

Vita

List of Tables

2.1 Three Layers Architecture of Cloud 15

2.2 BPMN, BPEL and XPDL 24

3.1 Six Services in PCSF 31

4.1 Domain Creator 44

4.2 Domain Participant 45

4.3 Domain Collaboration 46

5.1 The List of Roles and Groups 72

5.2 Task List 76

5.3 Characteristics of Amazon EC2 and RDS Used in Case Study 77

5.4 Other Amazon Web Services Used in Case Study 77

List of Figures

1.1 Overview of PCSF 4

2.1 Time-Space Matrix 10
2.2 Cloud used to represent Internet 14
2.3 Three categories of cloud computing services 14
2.4 Software as a Service 15
2.5 Platform as a Service 16
2.6 Infrastructure as a Service 17
2.7 Basic SOA Architecture 18
2.8 SOAP Structure 19
2.9 Workflow sequence 21
2.10 BPMN 22
2.11 BPEL 23
2.12 XPDL 23
2.13 Concept of SOMICAS 26
2.14 Architecture of SOMICAS 27

3.1 Three Layers of PCSF 30
3.2 Collaboration as a Service Layer 33
3.3 Collaboration Service Layer 36
3.4 Collaboration Service Instance Layer 37
3.5 Collaboration Lifecycle 38
3.6 How PCSF works 40

4.1 The PCSF Architecture 42
4.2 Domain Relationship 44
4.3 Data Model Design Detail 47
4.4 Creation Service Functionality 48
4.5 Sequence Diagram for Create Collaboration 50
4.6 Sequence Diagram for Deploy Collaboration 51
4.7 Instance Management Service Functionality 52
4.8 Run Collaboration Instance 54
4.9 Registration Service Functionality 55
4.10 Sequence Diagram for Participant Registration 56
4.11 Schedule and Coordinate Service Functionality 58
4.12 How Schedule and Coordinate Service Works 60
4.13 Class Diagram for Engine Factory 60
4.14 BPMN Example 61
4.15 Sequence Diagram for Process Deployment 62
4.16 Sequence Diagram for Process Start 63
4.17 Data Transfer Service Functionality 64
4.18 Monitor And Control Service Functionality 66
4.19 Timer And Task 68

5.1 Example Of Test Case Design 71
5.2 Workflow Diagram 73
5.3 Screenshot for Creator Page: Ready to Create Collaboration 78
5.4 Screenshot for Creator Page: Collaboration Created 79
5.5 Screenshot for Amazon S3 View 79
5.6 Screenshot for Creator Page: Instance Generated 80
5.7 Screenshot for Creator Page: Add Participant 81
5.8 Screenshot for Amazon Simple DB View 82

5.9 Screenshot for Creator Page: Waiting For Registration 83
5.10 Screenshot for Participant Page: Not Registered 84
5.11 Screenshot for Participant Page: Done Registration 85
5.12 Screenshot for Creator Page: Ready To Run The Collaboration . . . 86
5.13 Screenshot for Creator Page: Running Collaboration 87
5.14 Screenshot for Participant Page: View Task 88
5.15 Screenshot for File Upload Page 89
5.16 Screenshot for File Download Page 89
5.17 Screenshot for Creator Page: Multiple Collaboration and Instance . . 90
5.18 Screenshot for Creator Page: Running Instance “leave1” 91
5.19 Screenshot for Creator Page: Running Instance “testing1” 92
5.20 Screenshot for Creator Page: Running Instance “testing2” 93

Abbreviations

Ajax	-	Asynchronous JavaScript and XML
API	-	Application Programming Interface
BPEL	-	Business Process Execution Language for Web Services
BPMN	-	Business Process Modeling Notation
CaaS	-	Collaboration as a Service
Cloud	-	Resources are provided as services on line
CS	-	Collaboration Service
CSCW	-	Computer Supported Cooperative Work
CSI	-	Collaboration Service Instance
EC2	-	Elastic Compute Cloud
IaaS	-	Infrastructure as a Service
IBM	-	International Business Machines Corporation
JAX-RS	-	Java API for RESTful Web Services
JAX-WS	-	Java API for XML Web Services
jBPM	-	Java Business Process Management
P2P	-	Peer to Peer
PaaS	-	Platform as a Service
PCSF	-	Peer Collaboration Service Framework
RDS	-	Relational Database Service
S3	-	Simple Storage Service

SaaS	-	Software as a Service
SES	-	Simple Email Service
SOA	-	Service Oriented Architecture
SOAP	-	Simple Object Access Protocol
SOMICAS	-	Service Oriented Medical Image Synchronous Collaborative Analysis System
UDDI	-	Universal Description, Discovery, and Integration
URI	-	Uniform Resource Identifier
WFMC	-	Workflow Management Coalition
WFMS	-	Workflow Management System
WSDL	-	Web Services Description Language
XML	-	Extensible Markup Language
XPDL	-	XML Processing Description Language

Chapter 1

Introduction

In this thesis, Peer Collaboration Service Framework (PCSF), is proposed and implemented to facilitate web based peer collaborations. In this chapter, a brief introduction for this framework is given.

Section 1.1 shows the motivation of this research; section 1.2 includes the objectives and an overview of the framework design and implementation; finally the chapter will end with the structure of this thesis in section 1.3.

1.1 Motivation

Collaboration involves more than one person working together toward a common piece of task or a common goal [2], such as reviewing a book, designing a building, or analysing a system. It may happen locally or even across multiple areas, may take place in real time or over a period of time. There is no doubt that in today's business, most of the key tasks are strongly related to collaboration [9]. One of the important reasons that people are willing to collaborate is that they need to exchange ideas and information to solve problems which are hard to be done by individuals independently. Benefit of collaboration is that it breaks obstacles amongst resources (networks, information, people and so forth) and makes them work coordinately to

achieve a common target. Therefore collaboration is a good way to share resources among individuals who are scattered in locations.

There are many different technologies supporting collaboration. Traditionally, probably the easiest way to form a collaboration is to set up a face-to-face meeting. These kinds of collaborations are mostly centralized and synchronized, where each participant will meet at the same location and at the same time [47]. Another way is to employ collaboration support applications [20] establishing a virtual group to connect each participant, decentralizing the collaboration so that group members can be located in different places. Example of this is remote Audio/Web/Video conferences where dispersive users exchange their ideas and knowledge.

With the prevalence of the Internet and increasing multi-user interactions taking place via web every day, web-based collaboration has become important in team-based organizations, and thereby more attention is paid to it. The most often used examples are Instant Message applications, for users all over the world to chat on line with friends or families and get replies immediately. In addition, it doesn't have to be synchronized and people can choose to collaborate at different times. Examples of this type of collaboration are email, blogs or discussion boards where communications may take place over an extended period of time.

Web-based collaboration has been greatly enriched since the proliferation of mobile devices accessing the Internet with high-bandwidth networks [24] and powerful systems running at the backend, like cloud, to support it. With this added mobility feature, participants can form an ad-hoc groups anytime and anywhere. As Esbjrnsoon and Ostergren suggest, we are in the era of "spontaneous collaboratio" [16], where "human and/or computational participants coincide temporarily at a location and interoperate to satisfy immediate needs [or goals". Furthermore, with the P2P [56] approach, peers from different areas can share their work spaces without connecting to a center server or database [38].

However, it must be noted that collaborations are now much more flexible, dynamic, and subject to changes [24]. Participant will join and leave teams frequently [13]. To overcome these challenges, we will investigate how to manage these dynamic web-based collaborations and present a collaboration service framework implemented on cloud - Peer Collaboration Service Framework - to easily support peer collaborations specified by certain types of workflow definition and effectively facilitate the collaborations.

1.2 Thesis Overview

1.2.1 Objectives

In our research, we will investigate a systematic approach to manage network based dynamic peer collaborations. Here, peer collaboration is a collaboration by directly communicating and sharing resources between participants.

The goal is to make the peer collaboration easily created and managed by creators and easily used by participants, and to raise the quality of reliability and performance. By doing this, under the notion of Software as a Service (SaaS) and cloud computing, we propose a framework named "Peer Collaboration Service Framework" and also the concept of CaaS which is short for "Collaboration as a Service".

The framework contains CaaS which is software service(s) to dynamically generate and manage network based peer collaborations created and specified by users. The framework also includes a set of collaboration services, each of which are independent software services. The framework provides a mechanism to allow these collaboration services working effectively together to support CaaS.

1.2.2 Framework Overview

The PCSF is a framework implemented in cloud, which can create network based peer collaborations based on the information the creator specified, and at the same time generate a set of collaboration services to support the peer collaborations. Participants can register and take part in the collaborations no matter where they are located or when they are available. The overview design of PCSF is shown in Figure 1.1.

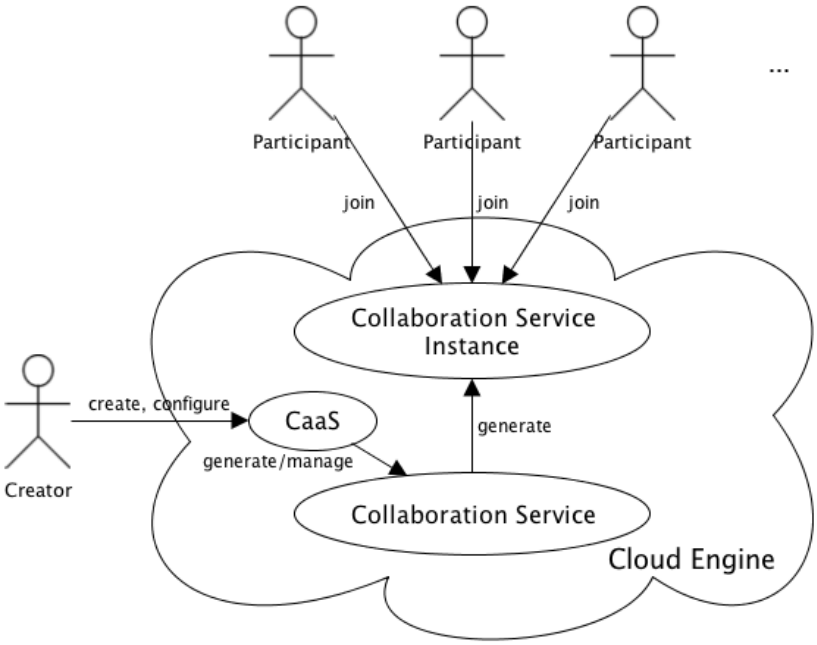

Figure 1.1: Overview of PCSF

There are two types of roles in this framework, collaboration creator and collaboration participant, as shown in Figure 1.1. The creator is in charge of establishing and manipulating the collaboration. To create a peer collaboration, creator must specify some configuration information:

- Peers (participants) who are about to take part in the collaboration
- A collaboration model, which is a process definition indicating how peers will

collaborate to finish the task

After the peer collaboration is created, a set of collaboration services are generated at the same time. This will include the following four services:

- Registration Service
- Schedule and Coordinate Service
- Data Transfer Service
- Monitor and Control Service

Every peer involved in the collaboration must register first as a participant for that collaboration. Only after all peers have registered, will the collaboration run under the support and supervision of these collaboration services.

1.3 Thesis Structure

The rest of the thesis is organized as follows: Chapter 2 is an introduction of the background support technologies including cloud, Service Oriented Architecture (SOA), web service, workflow, and network based collaborative technologies. Chapter 3 shows the high level design of the framework, including requirements and the three different layers of the framework. In chapter 4, more details on the framework design are discussed, as well as implementations of collaboration as a service, collaboration services that lies at the core of the framework. Chapter 5 gives a case study employing the framework to establish a collaborative testing environment. Next, in Chapter 6, evaluations of the framework are presented. Finally, in Chapter 7, the thesis will be summarized along with our contributions and future development possibilities.

Chapter 2

Background and Related Work

This chapter introduces background technologies and related work about collaborations applying external services. Section 2.1 lists all background technologies including:

- Conventional collaboration technologies and systems
- Cloud computing and the well-known three layer architecture
- SOA and web service, a service oriented way to implement the system
- Workflow, a key role in business process development to coordinate group work

Some previous and current researches and papers that are related to this thesis are introduced in section 2.2

2.1 Background

2.1.1 Conventional Collaboration Technologies and Systems

Collaboration technologies are seeing widespread adoption and implementation at all levels of organizations [43]. One of the most important reasons that people collaborate is that they need to exchange ideas and information to solve problems [34] which are difficult to overcome by individual independently.

2.1.1.1 Virtual Team

Lipnack and Stamps [29] define a virtual team as "a group of people who interact through interdependent tasks guided by a common purpose" and work "across space, time, and organizational boundaries with links strengthened by webs of communication technologies."

The members who form the virtual teams might not know each other or have worked together beforehand. They work in a virtual team supported and assisted by computer systems, enabling them to communicate with each other not only from different locations, but also across a span of time.

The potential payoffs to organizations from virtual teams are considerable. Firms can gain increased flexibility as teams with the proper expertise can be formed quickly without regard to any member's physical location. Firms can respond more rapidly to changes in their environment and reduce costs formerly caused by the need to move people [54].

There are a number of collaborative tools used to support virtual teams, ranging from synchronous such as video conferencing and live chat rooms, to asynchronous like newsgroups and email. As the development of web 2.0 technology, web based working environment appears where members in a virtual team use web browsers to share files and working spaces.

At the same time, virtual team technologies also experience some problems due to the challenges of time and space as listed [54]:

- Lack of awareness of information such as when teammates are available, on what activities they are working, or what else is occurring at the remote site that might influence teammates' work.
- Difficulties in forming good working relationships with remote teammates.
- Lack of trust in remote teammates.
- Difficulties in changing group communication practices due to the extra efforts required to introduce a new technology across locations.

2.1.1.2 Computer Supported Cooperative Work

The term CSCW, Computer Supported Cooperative Work, was originally coined by Greif and Cashman in 1984 as a shorthand way of referring to the interests of a number of researchers involved in the use of computers to support user groups [35]. CSCW addresses "how collaborative activities and their coordination can be supported by means of computer systems [6]."

The wide mix of researchers involved in CSCW makes it a broad discipline spanning many different research communities and cultures. Indeed it has been commented that the challenge and excitement of CSCW is its multi-disciplinary nature. It is important that those involved in CSCW recognise this challenge and learn to accept and value the skills of each discipline brings to CSCW. If the CSCW community fails to do so then a genuine opportunity for synergy will be lost [44].

CSCW provides a way of accessing, creating, gathering, managing, storing, and exchanging information [21]. Some of the CSCW systems contain only one concrete functionality, focusing on solving a specific problem, for instance, chat or data sharing. Other systems implement several functions, and this type of complex systems

usually provides more diverse functionalities, such as a document management system offers tools for version control, file sharing, check-in and check-out, publication, approval, etc. [61].

The CSCW systems are aiming to provide an integrated environment to several users in order to cooperate together in addressing a specific problem. Generally, CSCW systems can be described using two principal characteristics: time and space.

- Considering the concern of working during different hours, people either cooperate and interact synchronously or asynchronously. Synchronous interactions require people in the group who join the communication to appear at the same time, the example of this kind of cooperation is usually video conferencing. While in asynchronous interactions, cooperative users do not have to come together simultaneously, and it will span a much longer period in time.

- Relative to the space difference, the interactions are classified as either co-located or remote. Co-locate systems are usually in the form of face-to-face meeting while remote systems include more sophisticated distributed applications such as multimedia conferencing systems [35].

The above two principal characteristics can also be used to classify the CSCW systems, which is a well known conceptualization introduced by Johansen in 1988 called "Time-Space Matrix". It also appears in Baecker [3] as shown in Figure 2.1. It allows a technical method to category various CSCW systems.

2.1.1.3 Groupware

Groupware is a software to support work groups, where each worker in the group might has a different location or time zone. The purpose of groupware is to support the collaborative interactions and information sharing in groups based on computer networking technology, to improve the efficiency and quality of group work.

	One meeting site (same places)	**Multiple meeting sites (different places)**
Synchronous communication (same time)	**Face to Face Interactions** • Public computer displays • Electronic meeting rooms • Group decision support systems	**Remote Interactions** • Shared view desktop conferencing systems • Desktop conferencing with collaborative editors • Video conferencing • Media spaces
Asynchronous communication (different time)	**Ongoing** Tasks • Team rooms • Group displays • Shift work groupware • Project management	**Communication and Coordination** • Vanilla email • Asynchronous conferencing bulletin boards • Structured messaging systems • Workflow management • Version control • Meeting schedulers • Cooperative hypertext & organisational memory

Figure 2.1: Time-Space Matrix

Lotus company defines groupware as softwares that satisfy the "3C" requirements from users: Communication, Collaboration, and Coordination. Communication means message exchanging between users and groups, collaboration refers to information sharing in groups, and coordination is business process automation. This could be used as a criteria to check whether certain softwares should be categorized as groupware.

Based on the above definition, email, computerized bulletin board, video conferencing, workflow management all can be considered groupware. We can classify groupware according to their functionalities.

- **Messaging Passing**, such as Lotus Mail, Microsoft Mail, etc.
- **Process Automation**, such as IBM Flowmark, Floware, Workflow, etc.
- **Collaborative Interaction**, such as video conferencing, sharing blackboard, etc.
- **Integrated System**, such as Lotus Domino/Notes, Novel GroupWise, Microsoft Exchange, etc.

(1) **Lotus Domain/Notes**

Lotus Notes is a multi-user client-server cross-platform application runtime environment, the primary user-interface or client of the Lotus Domino/Notes suite [26].

Lotus Notes integrates quite a lot of functionalities including email, instant messaging, blogs, calendar, discussion forums, voice/video conferencing, web collaboration. In addition, it also provides users tools like IBM Lotus Domino Designer for developing additional applications that can be integrated into the system.

One of the most important features of Lotus Notes is database management functionality. The documents managed by Notes can be well structured or not well structured, therefore, it can execute and manage data that relational databases support, as well some data format they do not support. Notes also offers users the functionality of document templates to support complex rich-text, multimedia objects, searching, version control, etc. The latest version of Notes has added an extension functionality to manage information stored on the Internet.

(2) **Novel GroupWise**

Novel Groupwise extends an email system to be a more broad collaboration working environment. It may run on Intranet-Ware, WindowsNT and Unix. Groupwise integrates email, fax, voice mail, document, appointment, and task list into universalMailbox to save time and improve working efficiency.

Groupwise also provides remote access functionality including WebAccess and PhoneAccess. Users can connect to the office using WebAccess anywhere and anytime with Internet browser, PDA, mobile phone, etc. Or workers can call back to the PhoneAccess service in the office to get information from their

universalMailbox.

Besides, the Novel Groupwise supports a number of different data formats, making it very convenient and easy to read files from Exchange, SmartSuite, Word, CorelDraw and so forth.

(3) **Microsoft Exchange Server**

Microsoft Exchange Server is based on message exchanging. It provides the extended functionalities of eamil, meeting management, calendar management, task management, document management, video conferencing, and workflow management. All of the applications can be accessed through an Internet browser.

It is also a collaboration development platform, enabling clients to develop workflow, knowledge management systems, and web systems quickly by using tools like VisualInterDev together with some other Microsoft products such as IIS, SQL Server, etc.

2.1.1.4 Network Based Collaboration Technology

Collaboration technologies range from message and videoconferencing to group support systems and virtual communities. The traditional way of collaboration is to set up a face-to-face meeting and discuss the questions or apply groupware to the collaboration. However, the prevalence of the Internet and mobile technologies have dramatically changed the way people communicate and collaborate, which now could be considered network based collaboration or eCollaboration. Furthermore, the various network based collaborative technologies remove the barriers of time and space for scattered resources, and this makes collaboration easier and more convenient.

There are many different kinds of network based collaborative technologies and

they can be classified in terms of their functionality [2]:

- Collaborative Instant Messages Technologies. For example, MSN, Yahoo Messenger Programs, Google Talk, etc.
- Collaborative Voice and Video Technologies. For example, Skype VoIP, Meeting 3D as rooms for meetings, PalTalk Program, etc.
- Collaborative Writing Technologies. For example, Blogs, Google Docs which is a web based collaborative tool allowing people in different locations to edit the same document.
- Collaborative Technologies that have more than one function. For example, Sametime Connect Program from Lotus Quickr (IBM).

2.1.2 Collaboration Enabling Technologies

2.1.2.1 Cloud Computing

Cloud computing gets its name as a metaphor for the Internet because a cloud diagram is most frequently used to represent the Internet, indicating technologies not necessarily known by customers, as shown in Figure 2.2 [58]. The core concept of cloud computing is that resources, softwares and information are Internet-based and are provided to users or devices on-demand, and are accessed by using a browser instead of heavy terminals. In fact, cloud computing is not a totally new concept, it originated from the traditional computing and network technologies like distributed computing, grid computing, parallel computing, etc. [62]

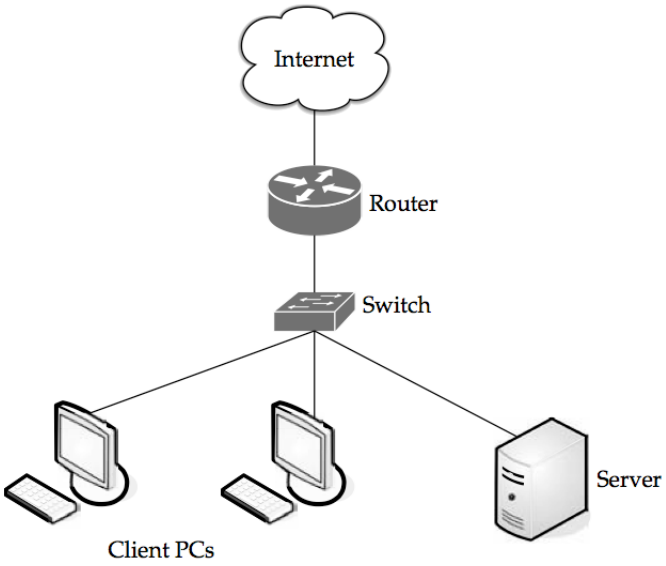

Figure 2.2: Cloud used to represent Internet

In cloud computing, everything is considered to be "as a service" which is short for "XaaS" [60]. For example, most generally cloud computing services are divided into three layers as shown in Figure 2.3 [5], and each layer is based on another layer.

Service Class	Main Access & Management Tool	Service content
SaaS	Web Browser	**Cloud Applications** Social networks, Office suites, CRM, Video processing
PaaS	Cloud Development Environment	**Cloud Platform** Programming languages, Frameworks, Mashups editors, Structured data
IaaS	Virtual Infrastructure Manager	**Cloud Infrastructure** Compute Servers, Data Storage, Firewall, Load Balancer

Figure 2.3: Three categories of cloud computing services

Table 2.1: Three Layers Architecture of Cloud

Layer	Description
SaaS (Software as a Service)	Applications or softwares are delivered as a service
PaaS (Platform as a Service)	Programming platform and tools like Java, MySql, Python and APIs are delivered as a service
IaaS (Infrastructure as a Service)	Hardware is delivered as a service, including processing power, storage, bandwidth, etc.

Software as a Service is also referred to Application as a Service. It aims to host the applications and softwares in cloud, and provide them to multiple users via internet with the "pay-as-you-go" model, as shown in Figure 2.4. In this way users can consume applications anywhere and anytime by using only a browser, and do not need to install software on computer themselves, or concern themselves with maintaining and updating the applications [60]. Some examples of SaaS are Salesforce automation and Customer Relationship Management (CRM) [46], NetSuite [36] and Google Office Productivity application [27].

Figure 2.4: Software as a Service

Platform as a Service is a bridge between hardware and application [22], providing developers a platform from the Internet containing all resources, systems and environment required to build applications, which developers can then use to build their own programs and then deliver them as Software as a Service [50] as shown in Figure 2.5. The responsibility of PaaS is to execute applications on the platform. It is also in charge of responses to external requests, as well as running scheduled jobs included in the application [50]. Since the platform is transparent to the users, the platform can share application servers among many users if they need only a low capacity, and will later dynamically scale the allocation when the loads increase. The well known examples are Google App Engine [23], Microsoft Azure Services Platform [33] and Salesforce App Exchange [46].

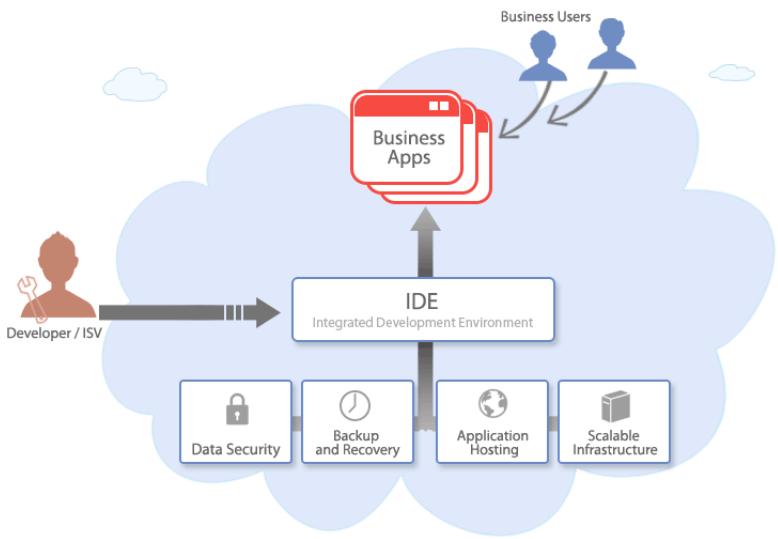

Figure 2.5: Platform as a Service

Infrastructure as a Service is sometimes called Hardware as a Service because it supplies users computing resources. For instance, the capacity of processing, network equipment, data center spaces or servers. And only the consumed services are charged for. In addition, those resources can be scaled up or down according to users' needs. This can happen "as a result of the rapid advances in hardware

virtualization" [62] as shown in Figure 2.6 [5]. This kind of model especially meets the needs of enterprise users since it relieving them from needing to spend money on building and maintaining data centers [60]. Examples of IaaS are Amazon Elastic Cloud (EC2) [12] and Amazon Simple Storage Service (S3) [45].

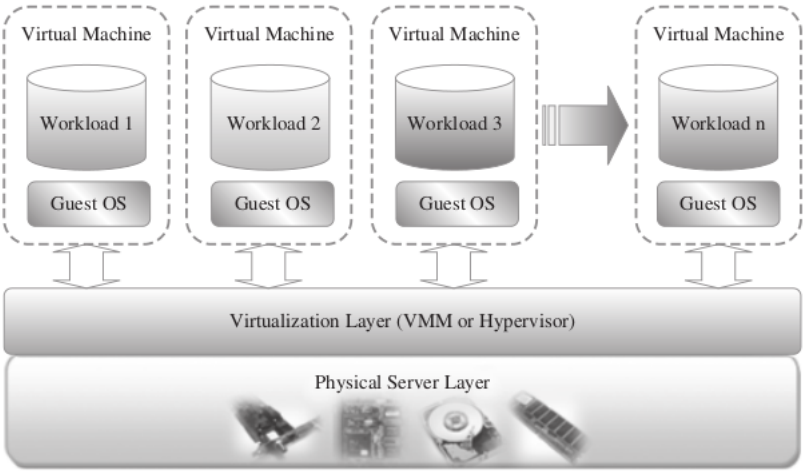

Figure 2.6: Infrastructure as a Service

All of these advantages make using cloud technologies an appropriate way to build applications using required resources like platform or hardware, but also offers the applications to more customers and users via the Internet.

2.1.2.2 SOA and Web Service

The definition Thomas Erl gives in his book Software-Oriented Architecture is solid and exactly describes what SOA means [15]:

> "Service-oriented architecture is a term that represents a model in which automation logic is decomposed into small, distinct units of logic. Collectively, these units comprise a larger piece of business automation logic. Individually, these units can be distributed."

In this definition, the units (also considered to be services) are well defined web services which are reusable and do not rely on other services [32]. They can be

accessed only by public interfaces and can be replaced or modified when required. With this kind of "loose couple" property, services can be plugged into and out from programs or applications as needed without worrying about incompatibility since they are built under common standards. The basic SOA architecture is shown in Figure 2.7.

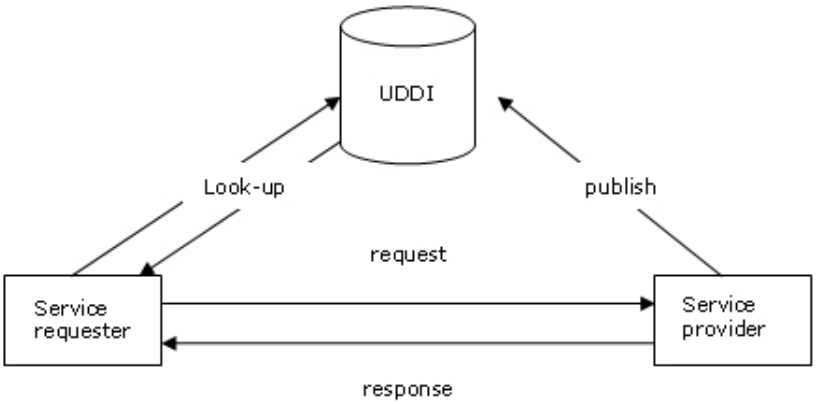

Figure 2.7: Basic SOA Architecture

Firstly, service providers publish their services to a service directory, which is UDDI. Secondly, service consumers match their needs by searching from directory. Finally, after finding the appropriate services, consumers will bind to the provider to make further communications.

Web service is one of the technologies that are used to implement SOA. The core web service standards include SOAP, WSDL, and UDDI which make the above three primary operations take place [39].

- **Web Service Description Language (WSDL)**

  An XML based language used to describe the details of the interfaces exposed by the web services, which can therefore be a way to access them. The details might include what a service does like the operations it provides, where it resides like the published URL, and how to invoke it like the data format and protocols used to access the operations.

- **Universal Description, Discovery, and Integration (UDDI)**

  Used to publish services. It contains a directory of all the services that are available in SOA system, including service location, service version, service expire date, etc.

- **Simple Object Access Protocol (SOAP)**

  An XML-based protocol that web services rely on to exchange messages. It uses common Internet transport protocol like HTTP to carry data, which can penetrate firewalls because most firewalls are default configured to accept HTTP requests.

Web services communicate with each other by sending and receiving SOAP messages. The structure of SOAP message is as shown in Figure 2.8. The request and response information are included in body.

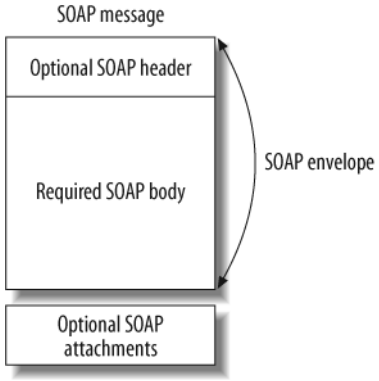

Figure 2.8: SOAP Structure

SOAP can be bond with HTTP, so the client can embed a SOAP request into HTTP message and send it to the server. Then the server side will return a HTTP response containing the result SOAP message, from which the client can abstract the results.

Example SOAP request message [53]:

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock">
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

The loose couple, reusability, standard interface and interoperability characteristics make SOA a top-of-mind choice for systems development in the industry [30], and many companies have been immigrating their legacy systems to SOA to give a new life to the systems [55]. Additionally, SOA can be greatly integrated with cloud computing, as they both utilise the concept of considering everything as a service.

2.1.2.3 Workflow Management

Workflow describes the order of a series of activities to finish a common task which takes place among various participants. During these activities, documents and information will be transported to different executors according to the predefined rules. It greatly improves the efficiency and productivity for business process coordination [59], and has also become especially important for those having large amounts of information to be dealt with. A workflow consists of a start point, end point, several tasks and the sequence of these tasks as shown in Figure 2.9 [59].

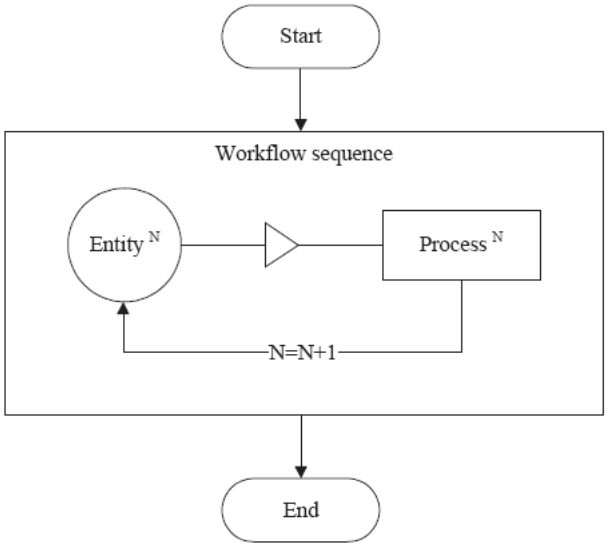

Figure 2.9: Workflow sequence

Workflow technologies include:

- **Workflow Management System (WFMS)**

  WFMS is a system consisting a set of tools to define, create, execute and manage workflows.

- **Workflow Engine**

  Workflow engine is the key component of WFMS, it manages and executes modeled processes.

- **Workflow Definition Languages**

  Used to describe the workflow. Some widely used are BPMN [37], BPEL [17], and XPDL [7].

(1) **BPMN**

The Business Process Modeling Notation (BPMN) is meant for use by business users who create the initial drafts of processes, technical developers who

are responsible for implementing processes and business people who manage and monitor processes [48].

It contains a set of graphic elements used to draw business process diagrams as the example shown in Figure 2.10. These elements are quite easily understood by all business roles. BPMN also provides an internal model to generate executable process language BPEL, therefore giving a standardized bridge for the gap between process design and process implementation [14].

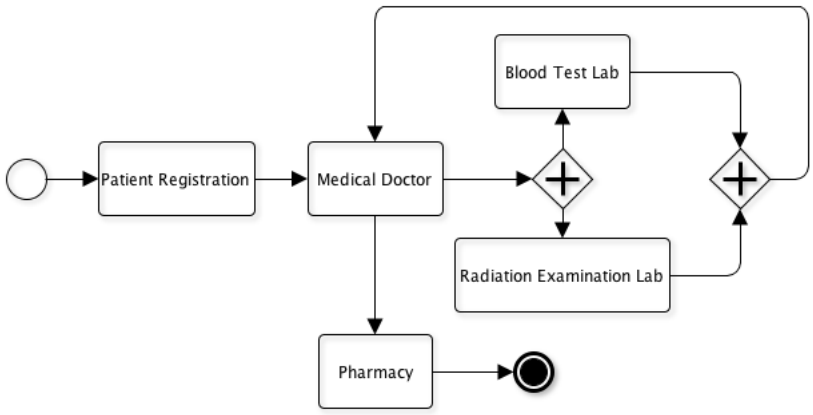

Figure 2.10: BPMN

(2) **BPEL**

Business Process Execution Language for Web Services (BPEL or BPEL4WS) now is the most widely used language to describe workflow [19]. It is an executable language which is designed to provide a definition of web services orchestration, especially the underlying sequence of interactions and the flow of data from peer to peer as shown in Figure 2.11 [57].

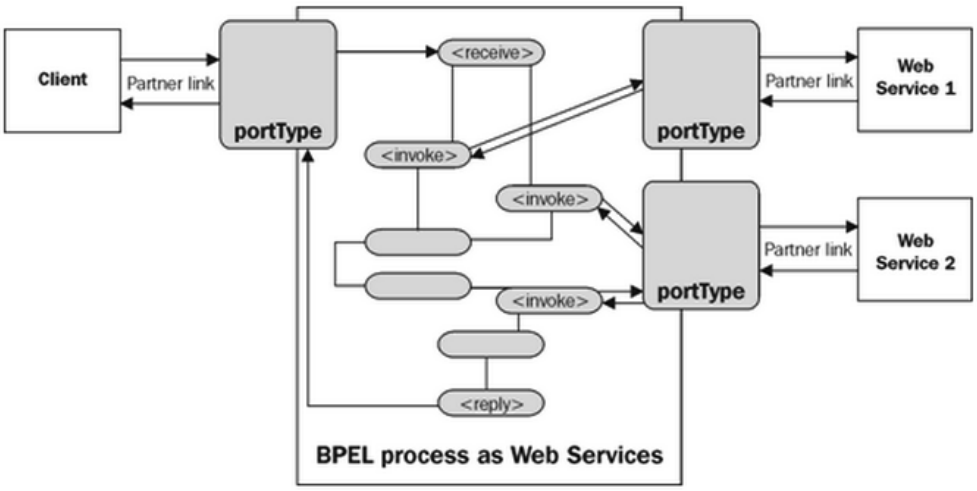

Figure 2.11: BPEL

(3) **XPDL**

XML Processing Description Language (XPDL) is a kind of new language standard put forward by WFMC organization [63]. It is generally used to store and exchange the process diagram and to represent the "drawing" of the process definition as shown in Figure 2.12.

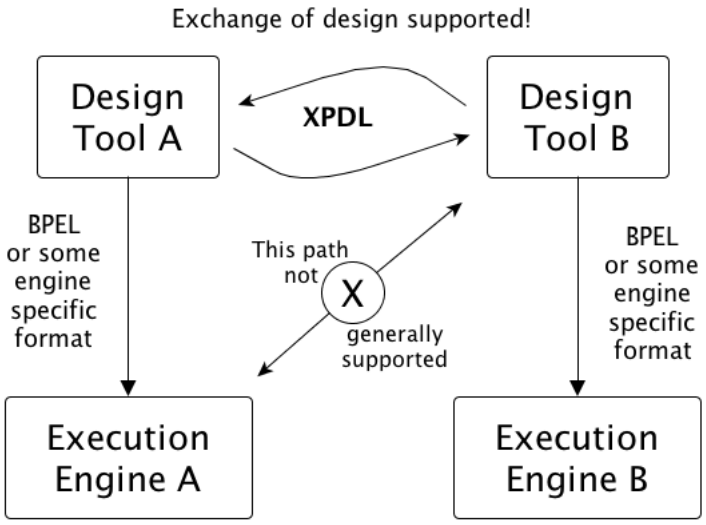

Figure 2.12: XPDL

Table 2.2 [41] shows the comparison of the above three languages.

Table 2.2: BPMN, BPEL and XPDL

	BPMN	**BPEL**	**XPDL**
Characteristics	Notation for drawing	Format to save processes	Format to save images
Primary Purpose	Visualization	Control over other systems	XMLization of BPMN diagrams
Diagram Information	Suitable	Impossible	Possible
Data Format	Image	XML	XML

The Workflow Management Coalition has also posted a white paper discussing the integration of workflow and the internet [11]. Since the internet is no longer being used only for scanning websites by individuals but is now a medium to support communications amongst people, it is believed to give a significant change to network based collaboration by associating "unprecedented information communication capabilities of the Internet with the strategic business processes automation and integration capabilities of workflow engines [11]".

2.2 Related Work

There are many investigations and researches regarding collaboration under the facilitate of external services or web services. For instance, SOMICAS [25] provides a system consisting of web services to support distributed physicians to collaborate together analysing medical imagines at the same time.

Some other related work is the Cloudbus toolkit [4], which is the workflow management system and cloud middleware developed at CLOUDS Lab, University of Melbourne. They focus on integrating workflow engines with cloud computing paradigm, since resources are limited when participants or work loads increase. With cloud integration, the resources can be provisioned on demand. Dustdar et al. [10] in his paper extended groupware systems with web service to integrate various group-

ware systems into a coherent and configurable architecture.

Next, we will explore more details of the SOMICAS system, showing its architecture and how it works.

2.2.1 SOMICAS

Service Oriented Medical Image Synchronous Collaborative Analysis System (SOMICAS) takes a stateful web services approach to build a medical image synchronously collaborative analysis system enables physicians analyzing medical image synchronously over geographic distances without the need of sending an image duplication to each peers. The concept of this system is shown in Figure 2.13.

- The medical image processing servers are connected to each hospital where the medical images as located. It contains services to access and process the image files.

- The collaboration server is deployed with collaboration services which manage and facilitate the collaboration sessions.

- The web server provides collaboration client portal for physicians at remote sites. Those distributed physicians can do collaborative medical image analysis by using browsers.

Figure 2.14 exhibits the architecture of SOMICAS, which is composed of three layers: the collaboration client layer, the collaboration sever layer and the medical image server layer.

2.2.1.1 Collaboration Client Layer

The collaboration client layer includes an Ajax-based user interface for physicians to join into a collaboration session and operate on images for displaying the result image that has been edited by others. The logic module and data module

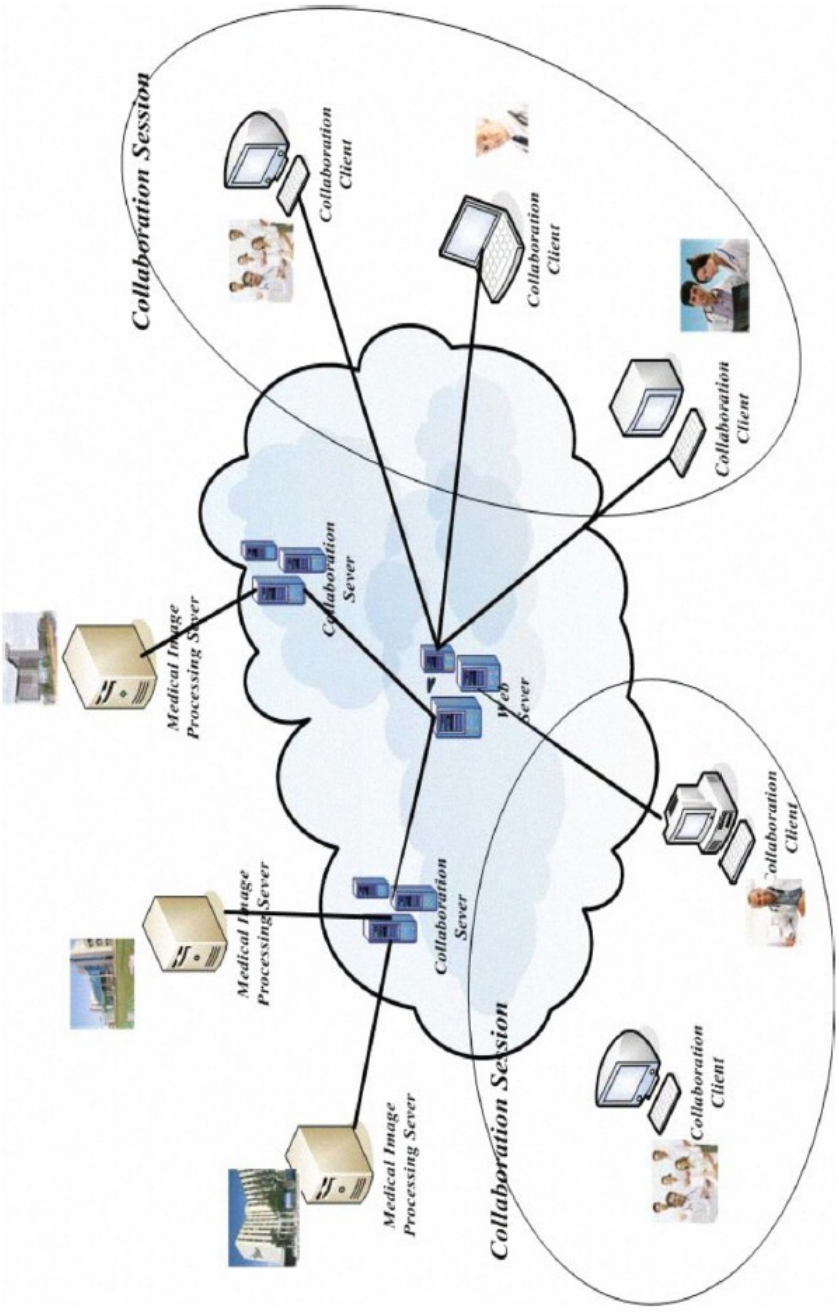

Figure 2.13: Concept of SOMICAS

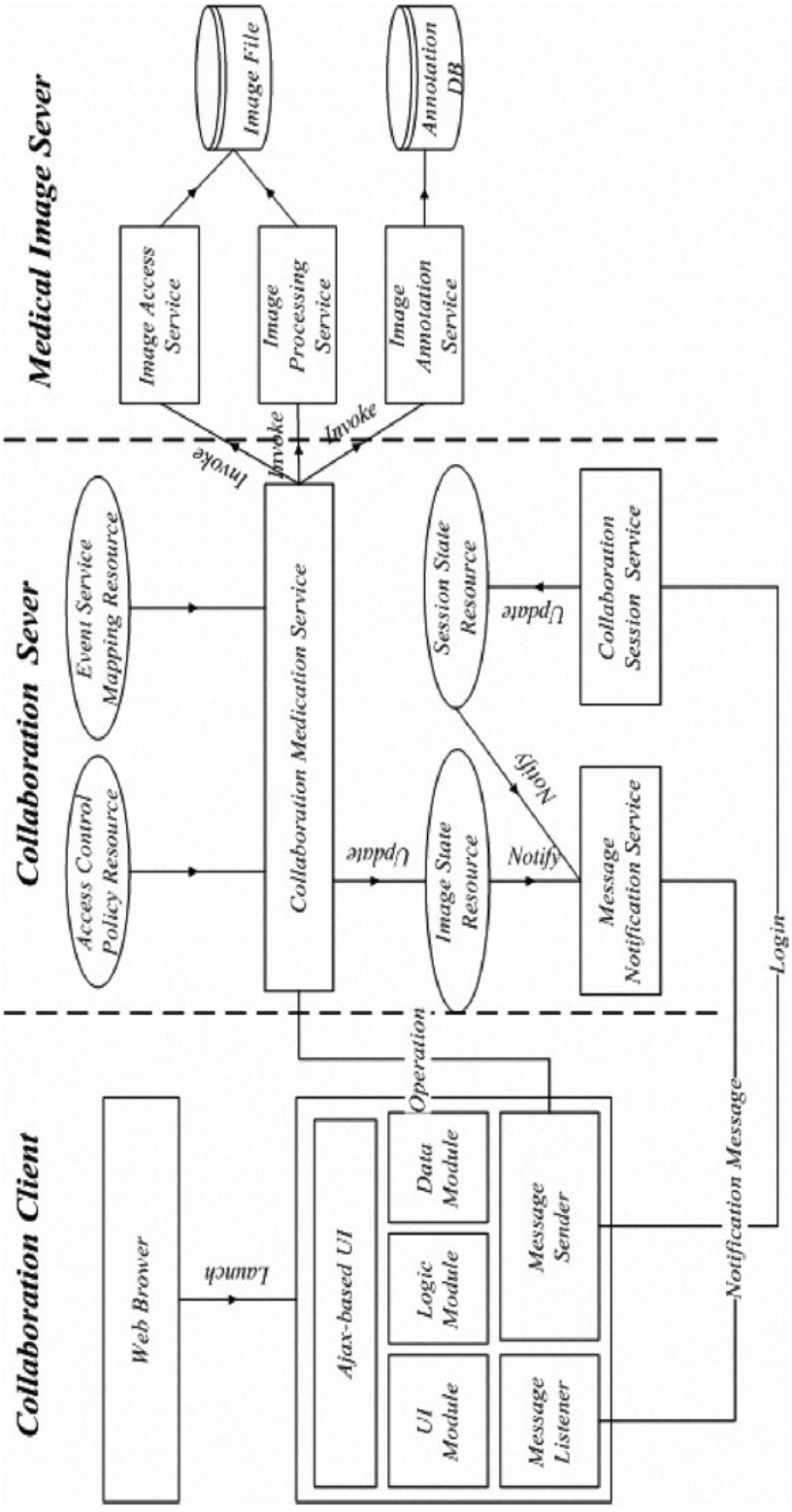

Figure 2.14: Architecture of SOMICAS

continuously tracks the logic and data of the client. Message listener and message sender are used to communicate with collaboration server layer to send interactive operation messages or receive notification messages otherwise.

2.2.1.2 Collaboration Server Layer

As shown in Figure 2.14, there are three services contained in the collaboration server layer: the collaboration medication service, the message notification service and the collaboration session service.

The collaboration session service is in charge of managing the collaboration sessions and also receiving messages from the client side to enable users to join in a collaboration session.

The collaboration medication service is like a bridge between client layer and medical image server layer. It will receive the interactive operations from collaboration clients and then invoke the corresponding services in the medical image sever layer.

The message notification service delivers the collaboration messages to the collaboration clients which will received by message listener.

2.2.1.3 Medical Image Server Layer

The image server layer contains image access services, image processing services and image annotation services. The image access services communicate with different data sources and provide unified interface to access the medical images. The medical image processing services are responsible for performing image processing tasks and rendering the result images. The annotation services enables collaboration clients to annotate the image and manage the annotated meta-data.

Chapter 3

Framework Design

We will give a high level view of the framework design in this chapter. In section 3.1 will give an overview of the framework, followed by the requirements that should be implemented in section 3.2. After that, there are three sections in the end to explain each of the three layers of this framework.

3.1 Overview

Peer Collaboration Service Framework is aiming to provide an easy way for users to dynamically create a peer collaboration and for participants to join in the collaboration. This framework is based on the notion of CaaS (Collaboration as a Service), which is a new concept proposed in this thesis.

CaaS is web service(s) that is in charge of creating peer collaborations. A set of collaboration services will be generated by it, along with establishing the collaboration. These collaboration services will help monitor, schedule, coordinate and control the collaboration, as well provide other services that are needed in the middle of the collaborative activities.

The collaboration the framework supports is peer collaboration, which means a collaboration by direct communications between participants and sharing individual

resources instead of common resources. Each participant is independent, and they may not know each other, or may locate in different areas. How they will work together to achieve a common goal is specified by a process definition file, which is the description of the steps in a procedure, supplied while creating the collaboration.

Thus, PCSF gives an efficient way to create ad hoc groups and at the same time provides a systematic way to manage and control them. The whole framework can be viewed as three layers as shown in Figure 3.1:

- Collaboration as a Service Layer
- Collaboration Services Layer
- Collaboration Service Instance Layer

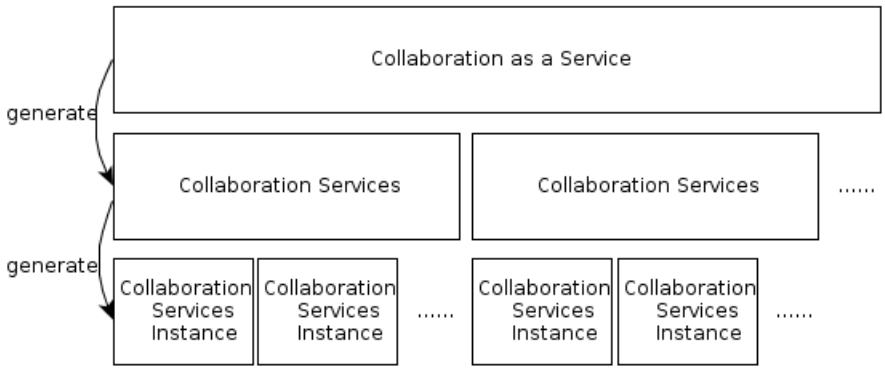

Figure 3.1: Three Layers of PCSF

3.2 Overview of Services

The collaboration as a service layer consists of firstly the creation service, which is used to build up peer collaborations, and instance management service, which is used to manage the collaboration instances. Collaboration services layer contains four web services that are used to support peer collaborations. The six services are the core part of the framework. The Table 3.1 gives an overview of each services.

Table 3.1: Six Services in PCSF

Service Name	**Service Layer**	**Responsibility**
Creation Service	*CaaS**	Create peer collaboration and generate collaboration services for it
Instance Management Service	CaaS	Manage the life cycle of generated instances
Registration Service	*CS***	Register participants into collaboraion
Schedule and Coordinate Service	CS	Manage the process of the collaborative activities; distribute tasks to participants
Monitor and Control Service	CS	Handle exceptions
Data Transfer Service	CS	Transfer data between participants

* Collaboration as a Service
** Collaboration Service

3.3 Requirements

To achieve the objectives, several requirements need to be satisfied. The most basic and crucial to be implemented are that:

- The framework consists of collaboration as a service to generate and manage peer collaborations.
- The framework consists of a set of collaboration services to support collaboration as a service.
- For each peer collaboration, collaboration as a service will generate a set of collaboration services for it.
- Collaboration services will provide all the necessary functionalities to facilitate the collaboration in the middle of the group work.

Besides the above requirements for the framework, the following elements of are also needed to be considered:

- The collaborations the framework creates and supports are dynamic network based peer collaborations.
- A process definition file is needed for each collaboration to direct how each participants coordinate together to finish the common task.
- Participants are considered to be individual peers in collaboration, which means resources are provided by them and they will also communicate directly.
- Each participant involved in the collaboration must explicitly register themselves, and only after that can the collaboration start to run.

Next, in the rest of this chapter, we will go through each service layer to discuss more about their roles and duties, and the relationship between them.

3.4 Collaboration as a Service Layer

Collaboration as a Service is a new concept that been proposed in this thesis, and it is also the core idea of this paper. "Something as a Service" is mostly common used under the notion of service computing and cloud computing. When something is designed as a service (which will usually be written in the way like "XaaS"), this 'X' will then be considered as a type of resource online that everyone can access and employ it with the connection to the Internet.

In this way, CaaS means collaboration is also a kind of resource that is available for everyone to get from internet. However it is must noted that CaaS does not deliver collaboration per se to users, which needs participants' activities to take part in. What CaaS delivers is a service that facilitates and manages the very process of the collaboration [31]. For example, a secretary can organize and schedule a

meeting for a manager and the clients, the secretary can help find a location and send invitations to all participants. But the meeting itself is not what he or she can provide, which needs the manager and all clients to attend and run it.

We define CaaS as software service(s), which can dynamically generate network based peer collaborations with the configuration information specified by the users. The configuration information includes a list of participants who will take part and a process definition file.

The framework also defines a group of independent software services to support CaaS, each of which is autonomous and stateless service and will provide a specific functionality to facilitate the peer collaboration. All these services will be brought together under the notion of CaaS to manage the collaboration towards that common goal [31].

In general, collaboration as a service layers contains two services: creation service and instance management service as shown in Figure 3.2.

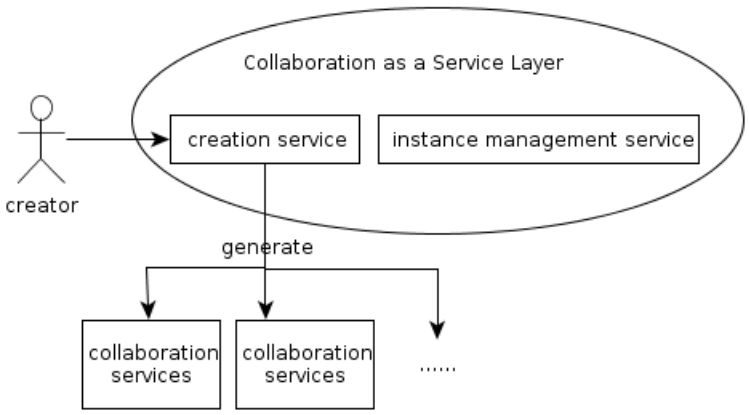

Figure 3.2: Collaboration as a Service Layer

(1) **Creation Service**

The creator will invoke creation service to build up a peer collaboration. To successfully do so, the creator needs to specify some information to configure this certain peer collaboration, including a workflow definition file. Then creation service will generate a collaboration service and bind the workflow file with it. This collaboration service will be used in the future to generate collaboration instances which are the concrete collaboration applications that participants can use. While generating collaboration instances, the creator should specify a list of participants for each. Therefore, different instances will follow the same workflow process but may have different participants involved.

(2) **Instance Management Service**

The framework also provides a mechanism to systematically manage the generated peer collaborations, and that is what instance management service does. It will monitor and manage the life cycle of all collaboration instances, as well offer a way to control those instances, such as start, stop, and delete commands.

3.5 Collaboration Service Layer

Collaboration service is a web service that provides a certain functionality to support peer collaboration. There are a lot of different aspects to be concerned to effectively support and facilitate a peer collaboration, some important parts of these might include:

- Registration, to register a certain participant into a peer collaboration.
- Resource Management, to manage the common data or documents which in the case of peer collaboration are owned by participants. It is also in charge of delivering data between participants.

- Task Management, to control the work flow and to coordinate the activities between each participants.

- Exception Management, to monitor the processing of the collaboration and handle the exceptions during running time. Some example exceptions could be time-out or fail to submit tasks.

- History Management, to log the completed activities and can be used to resume or roll back the collaboration when severe error occurs.

The collaboration management system will contain more functionalities and aspects than what is so far mentioned above. In a traditional management system, each could be implemented as a module, and then plugged into the system. But under the notion of SaaS and CaaS, we implement each of those modules as a web service, and thereby can collectively gather them together to facilitate and support the collaboration.

There are four build-in web services defined by the framework which will be bundled together as a collaboration service. That is, every time CaaS generates a collaboration service, these four web services will be packed into it. The only difference between each collaboration service is that they will be configured differently with their own specified workflow process file. Figure 3.3 shows the overview of collaboration service layer.

(1) **Registration Service**

Any peers that are asked to take part into the collaboration must first explicitly register themselves as a participant, and only after that can this collaboration occur between participants under the control of process definition file.

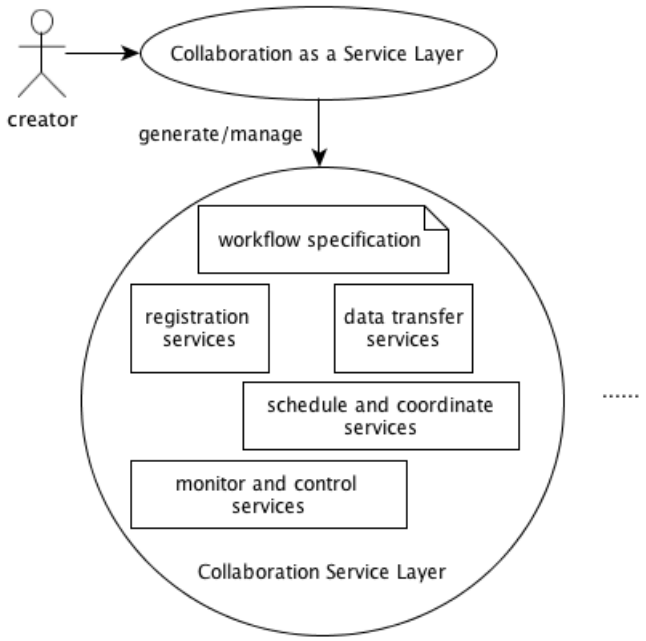

Figure 3.3: Collaboration Service Layer

(2) **Schedule and Coordinate Service**

Schedule and Coordinate Service is used to manage the process of the collaborative activities. It is also in charge of resource management to decide when to deliver, and who should be submitted to.

(3) **Monitor and Control Service**

The Monitor and Control Service is in charge of handling the exceptions. For example, when timeout exception occurs during workflow activity, the service may determine what appropriate actions should be performed. It could be sending the task to the participant one more time after a certain time period, or sending a message to all the participants informing that the task failed and has been terminated.

(4) **Data Transfer Service**

During the collaboration, there will be files being delivered from one partic-

ipant to another and all these documents will be transported by Data Transfer Service.

3.6 Collaboration Service Instance Layer

Collaboration service instance is a concrete peer collaboration application that has been generated by the framework. A collaboration service instance will include a process definition file, several participants, and a set of collaboration services. Figure 3.4 shows the collaboration service instance layer.

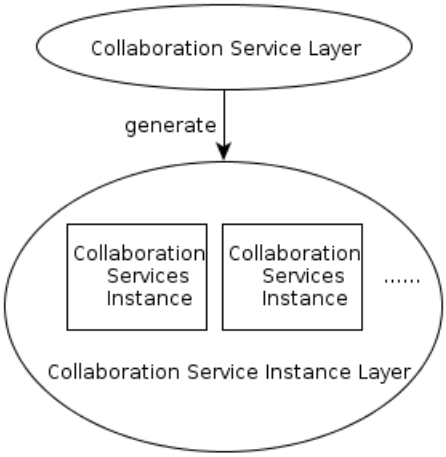

Figure 3.4: Collaboration Service Instance Layer

- **Process Definition File:** Each collaboration service instance is bonded with one process definition file, which defines how many participants roles should be involved and how they will interact together. The instances that generated from the same collaboration service will apply the same process definition file.
- **Participants:** The process definition file defines the roles for each task, which must be mapped to a concrete participant. Different participants can have the same role. For example, "submit a leave form" task must be done by

role “employee”, and “Mary” or “Chris” can both have the role “employee”. Therefore, despite that some instances apply the same process definition file, they might be done by different participants.

- **Collaboration Services:** Each collaboration service is an individual independent web service been defined by the framework that provides a specific functionality to support the collaboration.

3.7 Collaboration Lifecycle

The collaboration instance has five states in its lifecycle, as shown in Figure 3.5.

Figure 3.5: Collaboration Lifecycle

- **New Created**, the creator specifies a participant list and generates a collaboration instance from collaboration service. A registration requirement notification has been sent to each participant in the list.

- **Deployed**, all participants have done registration and the instance is ready to run.

- **Running**, process definition file has been initialized by the workflow engine, and the tasks have been assigned to participants.
- **Stop**, the instance is stopped by the instance management service.
- **Finish**, all tasks have been completed and the peer collaboration will be set to be finish.

Finally at the end of this chapter in Figure 3.6, we give a whole picture of how the framework works.

1. Creator invokes "Creation Service" to generate multiple collaboration services, each of which has its own workflow files.
2. Collaboration service generates multiple collaboration service instances, each of which is a concrete peer collaboration and has its own participants list.
3. Each participant must invoke "Registration Service" to explicitly join the peer collaboration.
4. After registered, the peer collaboration can be started and all the other collaboration services are running at the back end to support it.
5. "Instance Management Service" will also run at the back end to manage all instances like start or stop the instances.

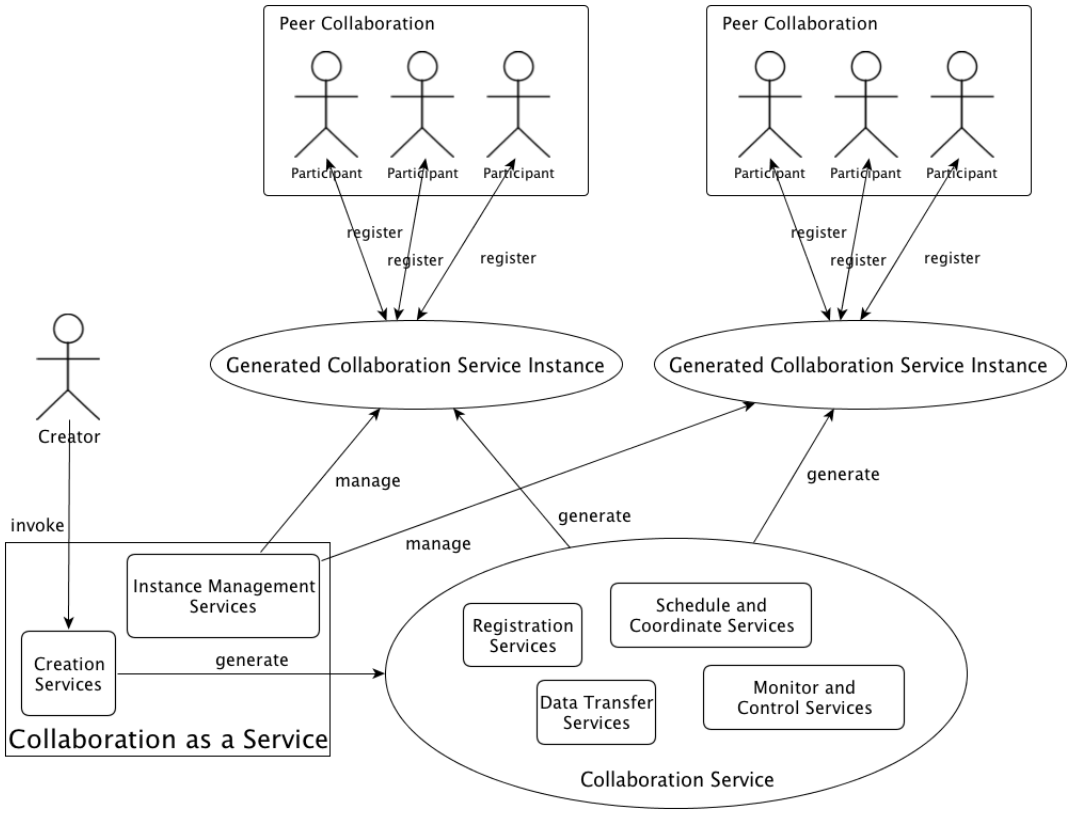

Figure 3.6: How PCSF works

Chapter 4

Framework Implementation

In this chapter, we will expand on more details about this system implementation. Section 4.1 will give an overview for the implementation showing the architecture of PCSF and what kind of services are needed in the framework to support peer collaborations. Section 4.2 will introduce the data model design and implementation. Section 4.3 follows with the design and implementation of collaboration as a service. Finally section 4.4 demonstrates the designs and implementations of each collaboration services.

4.1 Overview

4.1.1 PCSF Architecture

An overview of the PCSF architecture is presented in Figure 4.1. The framework is built on top of Amazon cloud services.

Collaboration as a service and generated collaboration service instances are hosted in Amazon EC2 [12]; the generated collaboration services, documents and other resources that are needed by the collaboration are stored in Amazon Simple Storage Service [45]; the collaboration and participant information are kept in Ama-

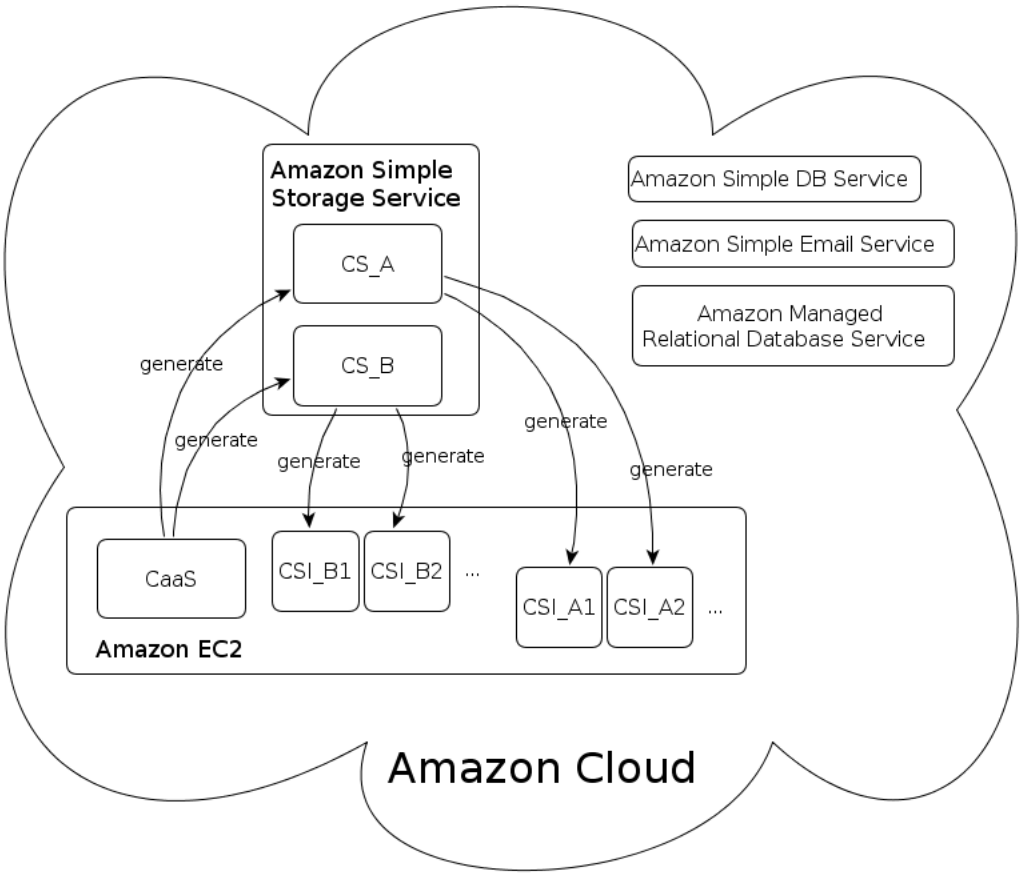

Figure 4.1: The PCSF Architecture

zon Simple DB [52]; task list and all the data that used to track the process of the collaboration are stored in Amazon Managed Relational Database [42]; Amazon Simple Email Service [49] is used to send system notification to participants and creators.

4.2 Data Model Design and Implementation

Two Amazon cloud services are applied by PCSF to host data: Amazon Simple DB and Amazon RDS. Amazon Simple DB is not a relational database and it provides a simple way to store data sets and retrieve them. Simple data like the status of the collaboration or the location of the documents are stored in Simple DB. Amazon RDS is used to set up a relational database in cloud and it provides database engine like MySQL, SQL Server and Oracle, therefore giving an easy way for those users who are familiar with them to migrate. PCSF builds up a MySQL database in RDS and stores data like task list or history records in it. These data are managed by the workflow engine itself, so the framework just establishes a database and connects it to the engine, then the engine will help create tables and all others.

In Amazon Simple DB, data are organized in "Domain", which is similar with the concept of "Table". PCSF builds up three domains in Simple DB: "Collaboration", "Creator", and "Participant". The relationship of these three domains is shown in Figure 4.2.

- One creator can create multiple collaborations
- One collaboration belongs to only one creator
- One collaboration can contains multiple participants
- One participant can belong to multiple collaborations

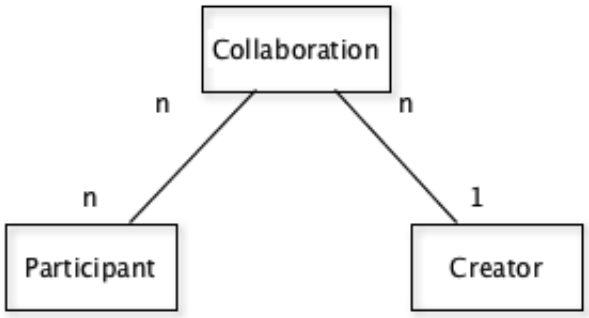

Figure 4.2: Domain Relationship

4.2.1 Domain Creator

As shown in Table 4.1, domain "Creator" contains creator information including:

Table 4.1: Domain Creator

Item	**Type**	**Explanation**	**Comment**
id	String	Unique identifier of the creator, been generated by the system, randomly between 0 and 1000000	Not Null, main key
name	String	Creator name	Not Null
password	String	Creator password	Not Null
email	String	Creator email	Not Null

After the creator creates a new collaboration, the system will send a notification email to the creator. In addition, when all participants have completed the registration for a collaboration, the creator will also receive a notification from the system saying the instance is ready to run.

4.2.2 Domain Participant

Table 4.2 shows the participant information that has been stored in domain "Participant". The process definition file defines **role** and **group**, indicating each task should be assigned to which role or group. A group can have multiple roles and a role can belong to multiple groups. When a task has been assigned to a role or a group, all participants with that role or in that group can take it. Each participant must be assigned a role and/or group to receive tasks.

Table 4.2: Domain Participant

Item	**Type**	**Explanation**	**Comment**
id	String	Unique identifier of the participant, been generated by the system, randomly between 0 and 1000000	Not Null, main key
name	String	Participant name	Not Null
email	String	Participant email	Not Null
collaborationId	String	Id of the collaboration that the participant in	Not Null
isReg	String	To identify if the participant has done the registration for the collaboration	Not Null, "yes" or "no"
role	String	The assigned role of the participant, this should be defined in process definition file	Not Null
group	String	The belonged group of the participant, this should be defined in process definition file, will be marked as "no group" if not join any group	Not Null

4.2.3 Domain Collaboration

Domain "Collaboration" contains collaboration information that is shown in Table 4.3. The **process id** is generated by the embeded workflow engine after the process definition file being deployed inside it.

Table 4.3: Domain Collaboration

Item	**Type**	**Explanation**	**Comment**
id	String	Unique identifier of the collaboration, been generated by the system, randomly between 0 and 1000000	Not Null, main key
processModel	String	Location of the process definition file	Not Null
name	String	Collaboration name	Not Null
creatorId	String	The id of the creator who creates the collaboration	Not Null
participants	String[]	Participant list of the collaboration	Not Null
currentState	String	Current state of the collaboration, including: • NEW CREATED • DEPLOYED • RUNNING • STOP • FINISH	Not Null
processId	String	The id of deployed process	Null

Figure 4.3 gives a full picture of the data model design:

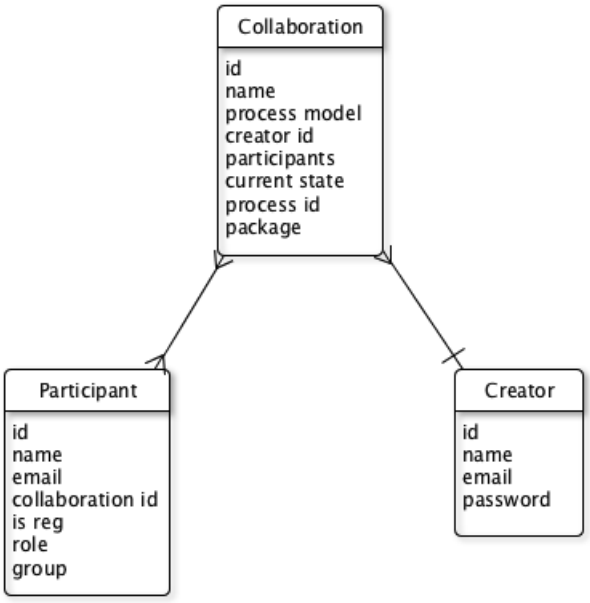

Figure 4.3: Data Model Design Detail

4.3 CaaS Implementation

Collaboration as a service is software service(s) that are used to create peer collaborations, and this layer contains two web services: Creation Service and Instance Management Service.

4.3.1 Creation Service

The collaboration creator creates a peer collaboration by invoking Creation Service, specifying a process definition file which is in the form of a workflow file used to direct the collaborative activities. The Creation Service will then generate and pack a group of collaboration services together with the peer collaboration and bind them. With this generated collaboration, creator can then deploy multiply collaboration instances, for each one, a participant list must be specified as well.

After building up the collaboration instance, the expected participants will then be notified to register into this collaboration.

4.3.1.1 Design

The "Creation Service" provides three operations for creator to manipulate: create collaboration, which is to generate collaboration service; deploy collaboration, which is to generate a collaboration instance and delete collaboration, which is to delete collaboration instances or collaboration service. The design is as shown in Figure 4.4 and the followed codes.

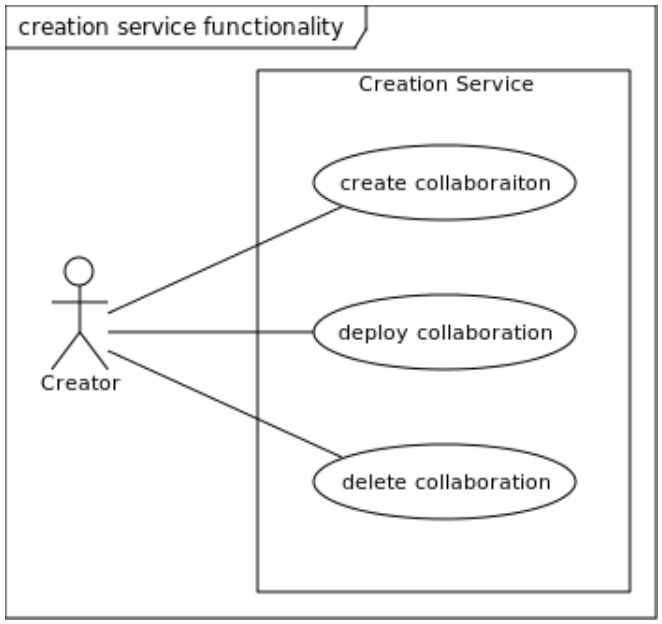

Figure 4.4: Creation Service Functionality

The design codes of the interfaces:

```
@WebService
public interface CreationService {
    /**
     * Create a new collaboration.
     *
     * @param collaborationName
     * @param participants
     * @param creatorId
     * @param workflow
     * @return a new collaboration
     */
    @WebMethod
    public boolean createCollaboration(String collaborationName,
        String creatorId, File workflowFile);

    /**
     * Deploy a collaboration and generate a set of services for it.
     *
     * @param collaborationId
     */
    @WebMethod
    public boolean deployCollaboration(String collaborationId);

    /**
     * Delete a collaboration.
     *
     * @param collaborationId
     */
    @WebMethod
    public boolean deleteCollaboration(String collaborationId);
}
```

4.3.1.2 Implementation

The user must register as a "Creator" first so as to log into the system, and all related information about this creator will be added to domain "Creator" stored in Amazon Simple DB. Only after that is the creator allowed to create collaborations. To successfully build up a peer collaboration, a collaboration name and a workflow definition file must be specified by the creator.

While adding a participant into the collaboration instance, a role and group should also be assigned to this peer. This role or group must be exactly the same with what are defined in workflow file, which is used to map the participants to tasks.

After finishing the creation and deploy instance, the state of the collaboration instance will be set to be "new created" and a random ID will be generated for it. The collaboration can not be operated before all participants have complete registration.

The created peer collaboration and relative participant information will be added to Amazon Simple DB; the workflow file will be uploaded to Amazon S3.

Figure 4.5 exhibits a sequence diagram showing in detail the process of creating a peer collaboration. Figure 4.6 shows the process of deploying a collaboration instance. After a peer collaboration instance has been deployed successfully, each participant will receive a notification being asked to register into that deployed collaboration instance.

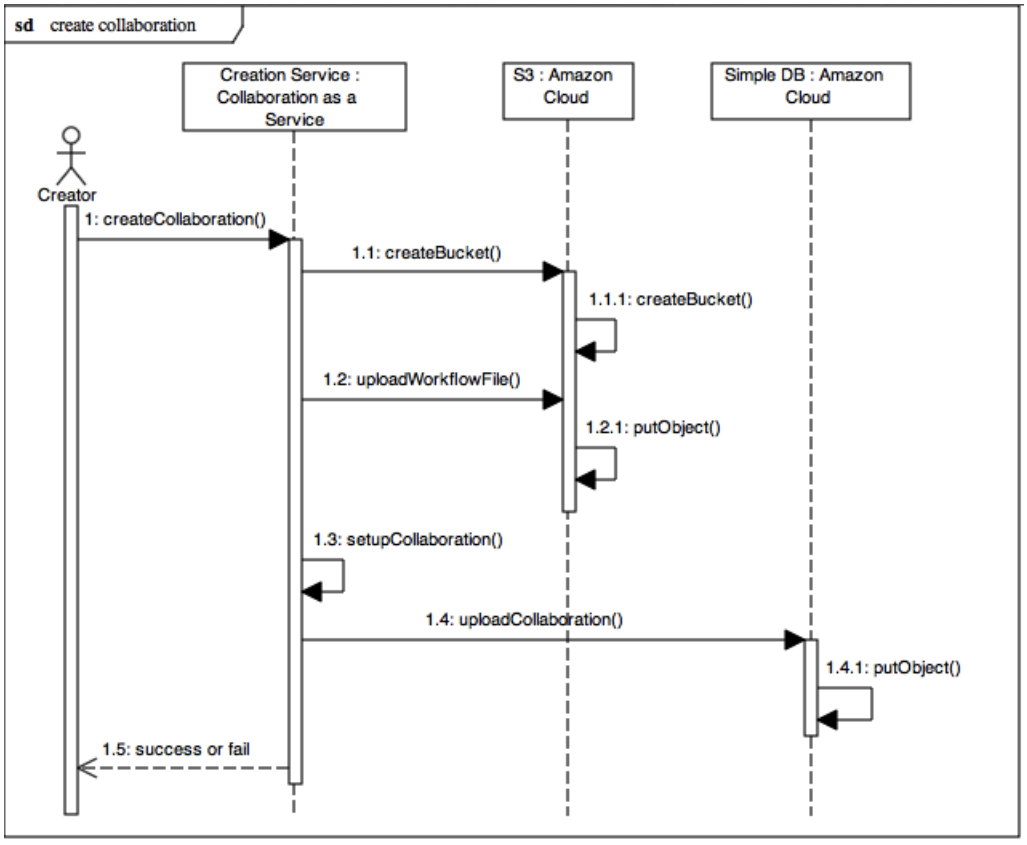

Figure 4.5: Sequence Diagram for Create Collaboration

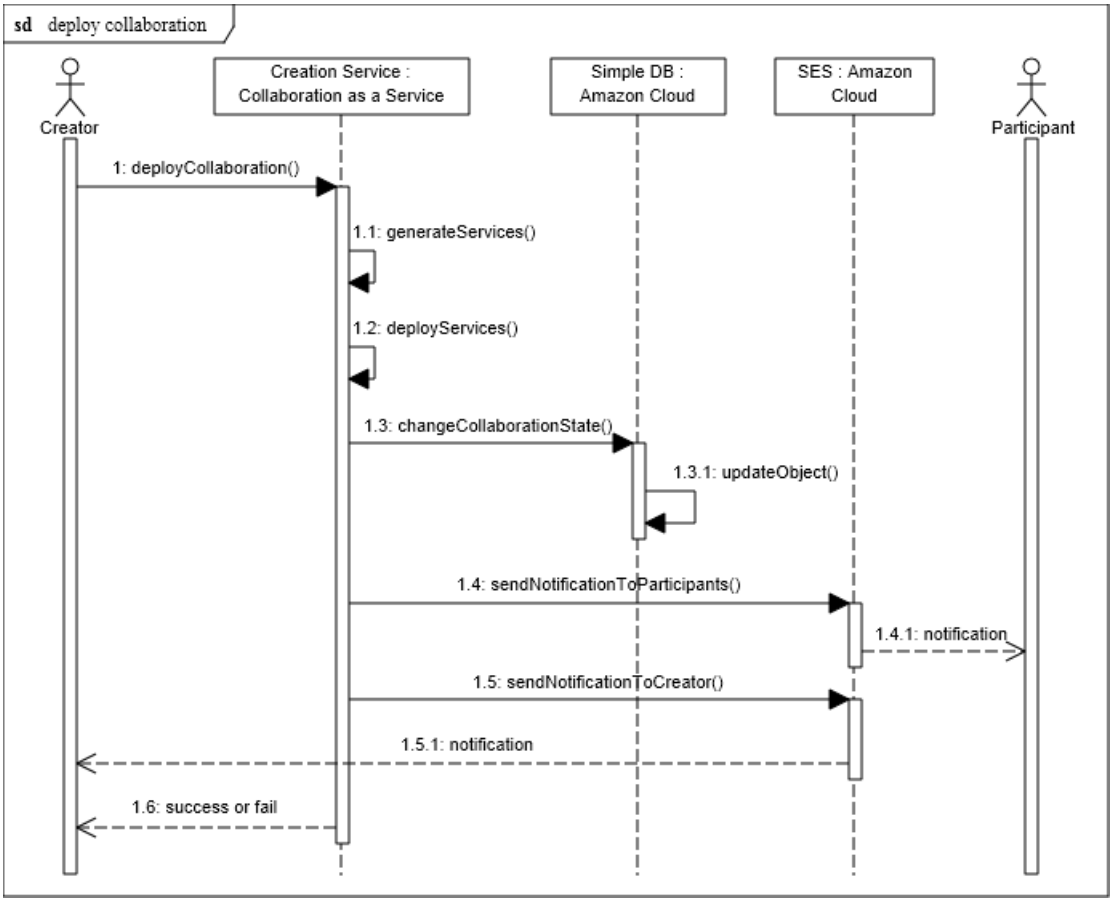

Figure 4.6: Sequence Diagram for Deploy Collaboration

The notification will be sent to an email address in the form of email. The following is an example email that the system sends to a participant after successfully deploying a collaboration instance.

Example email:

```
Dear medical doctor,

You are asked to participant in a collaboration!
Please click the following link to register into the collaboration
using your given id:

Your id: 134905
http://ec2-54-242-33-5.compute-1.amazonaws.com:8080/pcsf/index.jsp?
action=participantLogin
```

4.3.2 Instance Management Service

Instance management service is used to manage all collaboration instances that have been deployed and are running in the system.

4.3.2.1 Design

Instance Management Service provides two interfaces for creator to manipulate the collaboration instances as shown in Figure 4.7 and the followed codes.

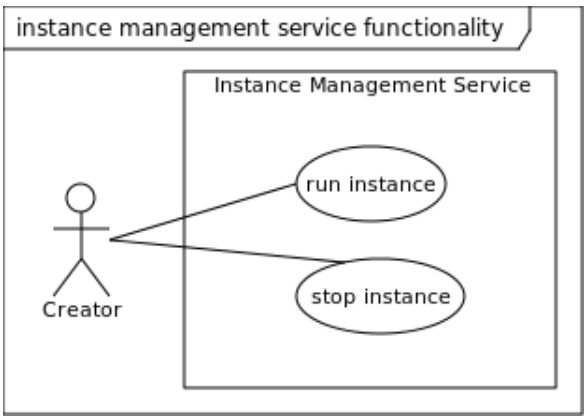

Figure 4.7: Instance Management Service Functionality

The design codes of the interfaces:

```
@WebService
public interface InstanceManagementService {
    /**
     * Run a collaboration instance.
     *
     * @param collaborationId
     */
    @WebMethod
    public void runInstance(String collaborationId);

    /**
     * Stop a collaboration instance.
     *
     * @param collaborationId
     */
    @WebMethod
    public void stopInstance(String collaborationId);
}
```

4.3.2.2 Implementation

Every collaboration instance has its own set of specific independent collaboration services, one of which is "schedule and coordinate service" that contains a workflow engine to execute the collaboration process. So when the creator commands the system to run a collaboration instance, the instance management service will try to communicate to schedule and coordinate service of that particular instance asking it to start the process. This is as shown in Figure 4.8. And the stop function will be similar.

4.4 Collaboration Services Implementation

Collaboration services will be generated together with the peer collaboration, each of which is an individual web service and will provide a certain aspect of functionality to support it.

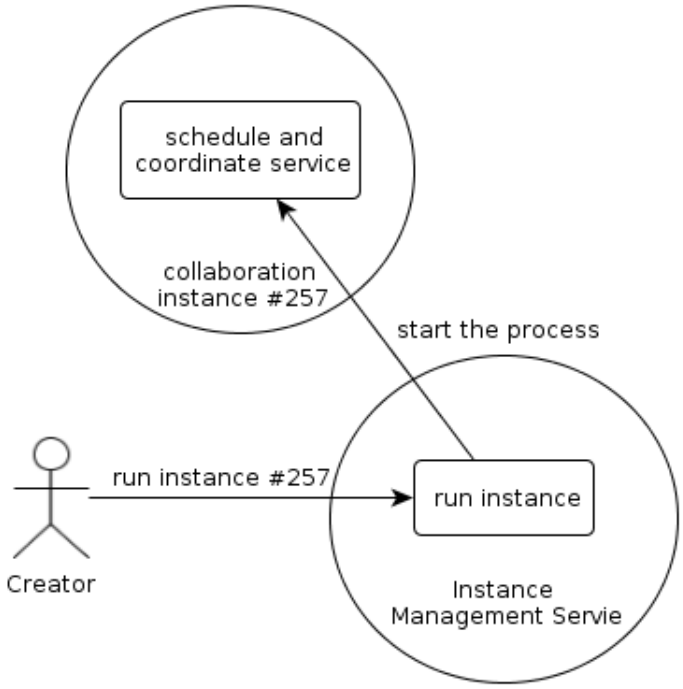

Figure 4.8: Run Collaboration Instance

4.4.1 Registration Service

Any peers that are asked to take part into the collaboration must first register as a participant after receiving the notification, and only after that can this collaboration occur between participants under the control of process definition file.

4.4.1.1 Design

Figure 4.9 shows that only one functionality "set as registered" has been provided by registration service. After that it is followed by a section of codes showing the interface definition. Notice that the web method requires one input parameter "participant id" which is included in the notification message that has been sent to each participant.

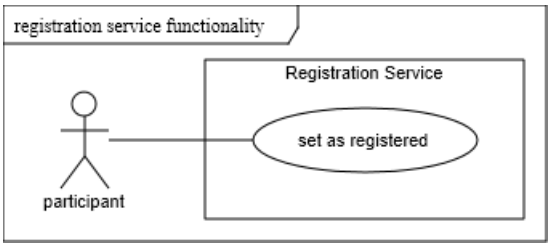

Figure 4.9: Registration Service Functionality

The design codes of the interfaces:

```
@WebService
public interface RegistrationService {
    /**
     * Register the participant into the collaboration.
     *
     * @param participantId
     */
    @WebMethod
    public void setAsReg(String participantId);
}
```

4.4.1.2 Implementation

Participants need to explicitly invoke "setAsReg" method provided by registration service to finish registration. The following Figure 4.10 depicts the detail registration process.

First, the service will query the participant with the given participant id from Amazon Simple DB and update its registration state to be "yes".

Then, the service will check if all the other participants in the same collaboration have done registration. If registration has been fully completed, a notification will be sent from Amazon Email Service to the creator and also a SOAP message will be sent to "Schedule and Coordinate Service" to initialize the process and the creator can manipulate the collaboration to start or stop it.

PCSF applies a third party framework named Apache CXF [18] to build and develop these collaboration services using frontend programming APIs, like JAX-WS [40] and JAX-RS [8]. With the help of CXF, it is much more convenient to make

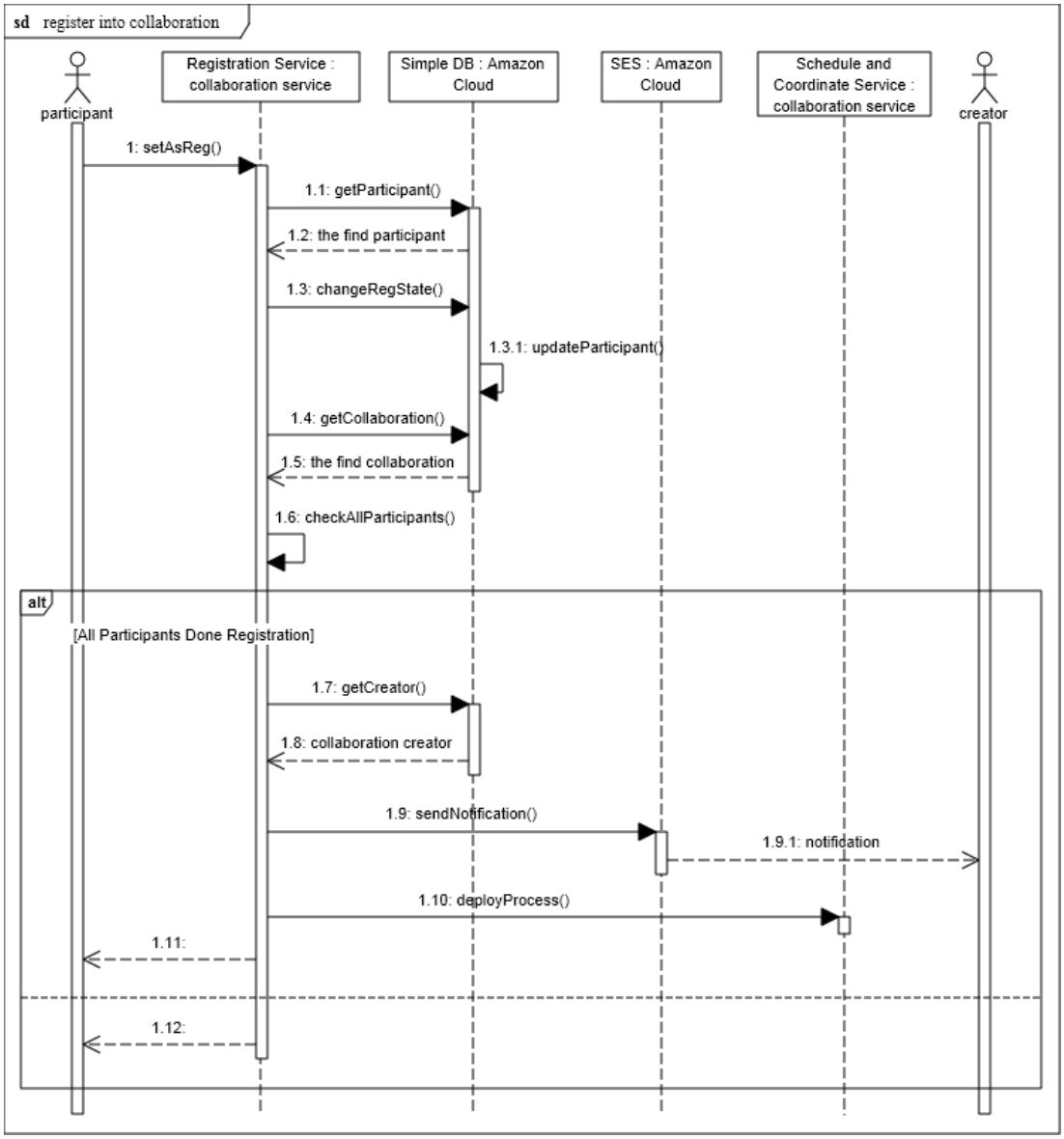

Figure 4.10: Sequence Diagram for Participant Registration

a SOAP call and get the results to and from a web service. The following is a method that build on top of CXF using which collaboration services can communicate like invoking local methods.

```
/**
 * A general method used to call web service.
 *
 * @param wsUrl
 *         web service url
 * @param method
 *         method name
 * @param arg
 *         method parameters
 * @return results
 */
private Object[] callService(String wsUrl, String method, Object... arg) {
    Object[] resutls = null;
    JaxWsDynamicClientFactory dcf = JaxWsDynamicClientFactory.newInstance();
    Client client = dcf.createClient(wsUrl);
    try {
        resutls = client.invoke(method, arg);
    } catch (Exception e) {
        e.printStackTrace();
    }

    return resutls;
}
```

4.4.2 Schedule and Coordinate Service

After creator has built up the collaboration and all the participants have registered into it, it is ready to launch. The collaboration will run under the rules of the workflow file been assigned before and "Schedule and Coordinate Service" is used to manage the process of the collaborative activities. It is also in charge of resource management to decide when to deliver and who should be submitted to.

A third party workflow engine "JBoss jBPM" [28] is embed into "Schedule and Coordinate Service" to help compile and execute the process definition file. jBPM supports the latest BPMN 2.0 specification. An Amazon RDS MySQL database is also launched being used to connect with the jBPM engine to store data including the task list, user list, and history records etc.

4.4.2.1 Design

The functionalities this service provides are given in Figure 4.11. It offers creator a few methods to manipulate the process, including run or stop the collaboration, get runtime details of the running process, etc. It also allows participants to get their assigned tasks from the center controller and submit them while done. The followed codes are design of these interfaces.

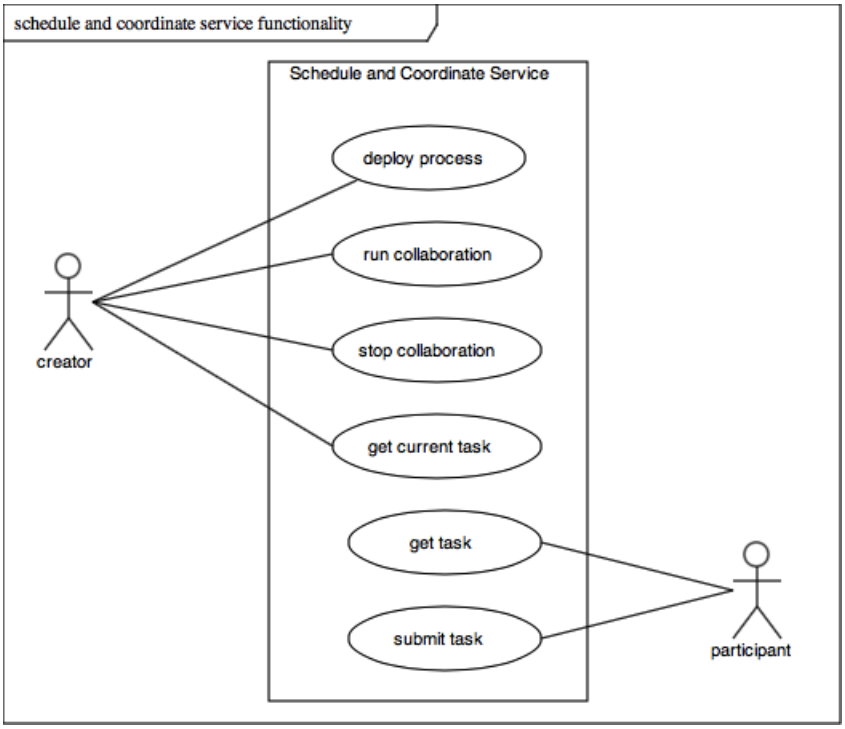

Figure 4.11: Schedule and Coordinate Service Functionality

The design codes of the interfaces:

```
@WebService
public interface ScheduleNCoordinateService {
    /**
     * Deploy a process
     */
    @WebMethod
     public void deployProcess(String collaborationId);

    /**
     * Run collaboration
     */
    @WebMethod
```

```
public void runCollaboration(String collaborationId, String processDefId);

/**
 * Stop collaboration
 */
@WebMethod
public void stopCollaboration(String collaborationId, String processDefId);

/**
 * Get task list of certain user
 */
@WebMethod
public String[] getTask(String uesername, String collaborationId);

/**
 * Get current task
 */
@WebMethod
public String[] getCurrentTask(String collaborationId);

/**
 * Submit a task
 */
@WebMethod
public void submitTask(String taskId, String collaborationId);
```

To start a process instance, schedule and coordinate service must first get the workflow file from data center and then deliver the file to jBPM engine to initialize the process. After a process has been created, relative tables will also have been built up in Amazon RDS database. jBPM applies Hibernate as its persistent layer solution and supports most of the popular database technologies by simply modifying the configuration file. Figure 4.12 is an overview of how it works.

The "EngineFactory" is in charge of creating new engines based on the types of workflow files, which can be identified by the extension, usually in the form of ".bpmn", ".xpdl", or ".bpel". Figure 4.13 gives a picture of engine factory class diagram, but so far, PCSF only implements BPMN engine and the others will be left as future extension.

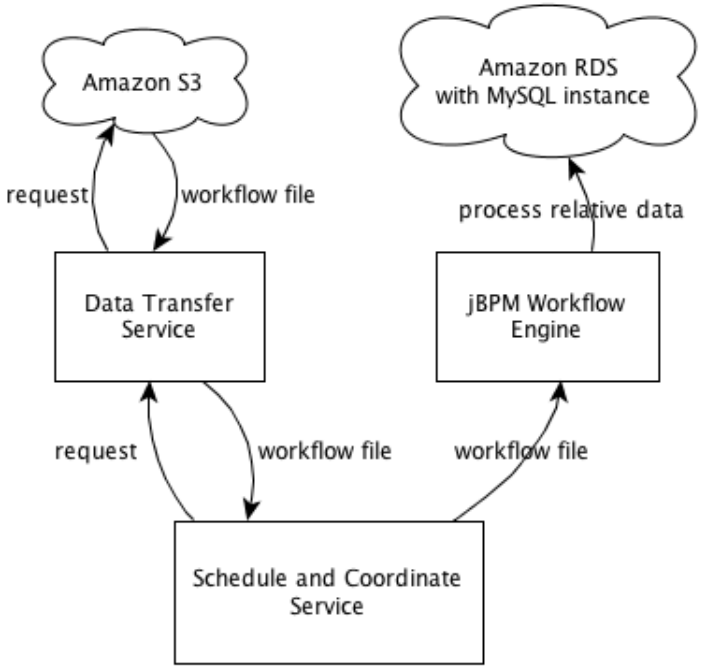

Figure 4.12: How Schedule and Coordinate Service Works

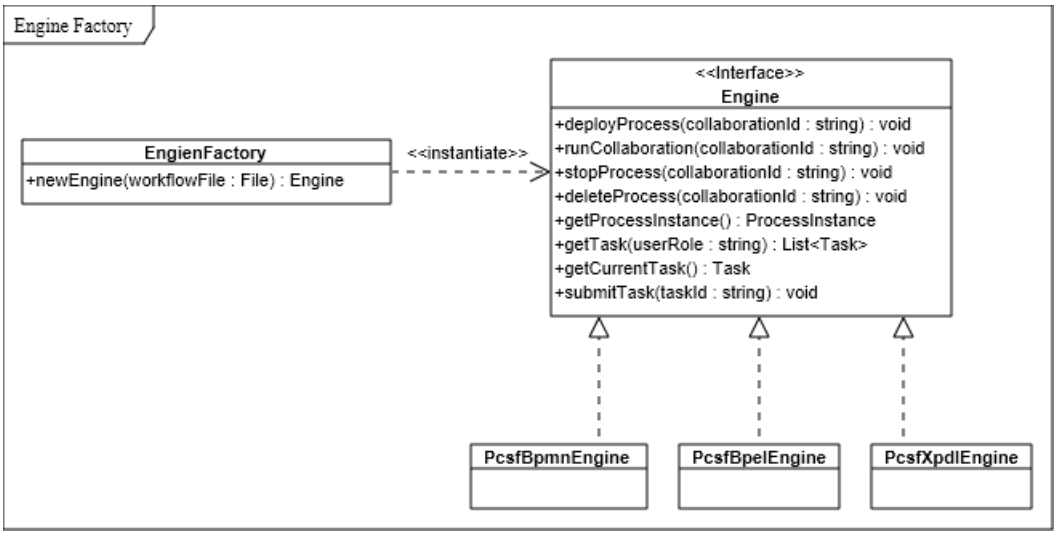

Figure 4.13: Class Diagram for Engine Factory

4.4.2.2 Implementation

The workflow file is an XML format file, and each elements in the workflow diagram is represented by an XML node as shown in the following simple BPMN example, which includes one start node, one end node, and the only one user task that will be assigned to group "managers". Both the diagram and the source codes are easy to be understood by workflow designer or business users.

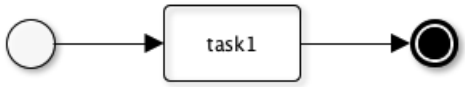

Figure 4.14: BPMN Example

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="simpleExample">
    <startEvent id="start" name="start">
    <sequenceFlow id="Flow1" sourceRef="startEvent" targetRef="task1"/>
    <userTask id="task1" name="task1">
        <potentialOwner resourceRef="manager">
            <resourceAssignmentExpression>
                <formalExpression>managers</formalExpression>
            </resourceAssignmentExpression>
        </potentialOwner>
    </userTask>
    <sequenceFlow id="Flow2" sourceRef="task1" targetRef="end"/>
    <endEvent id="end" name="end"/>
</process>
```

It is quite straightforward to deploy a process by simply initializing an engine and then passing the workflow file to it:

```
ProcessEngine processEngine = Configuration.getProcessEngine();
RepositoryService repositoryService = processEngine.getRepositoryService();
String processDeploymentId = repositoryService.createDeployment()
    .addResourceFromFile(processFile).deploy();
```

The workflow engine will generate a deployment id for each deployed process instance which later can be used to start or stop the process. Figure 4.15 is a sequence diagram showing the details of deploying a process.

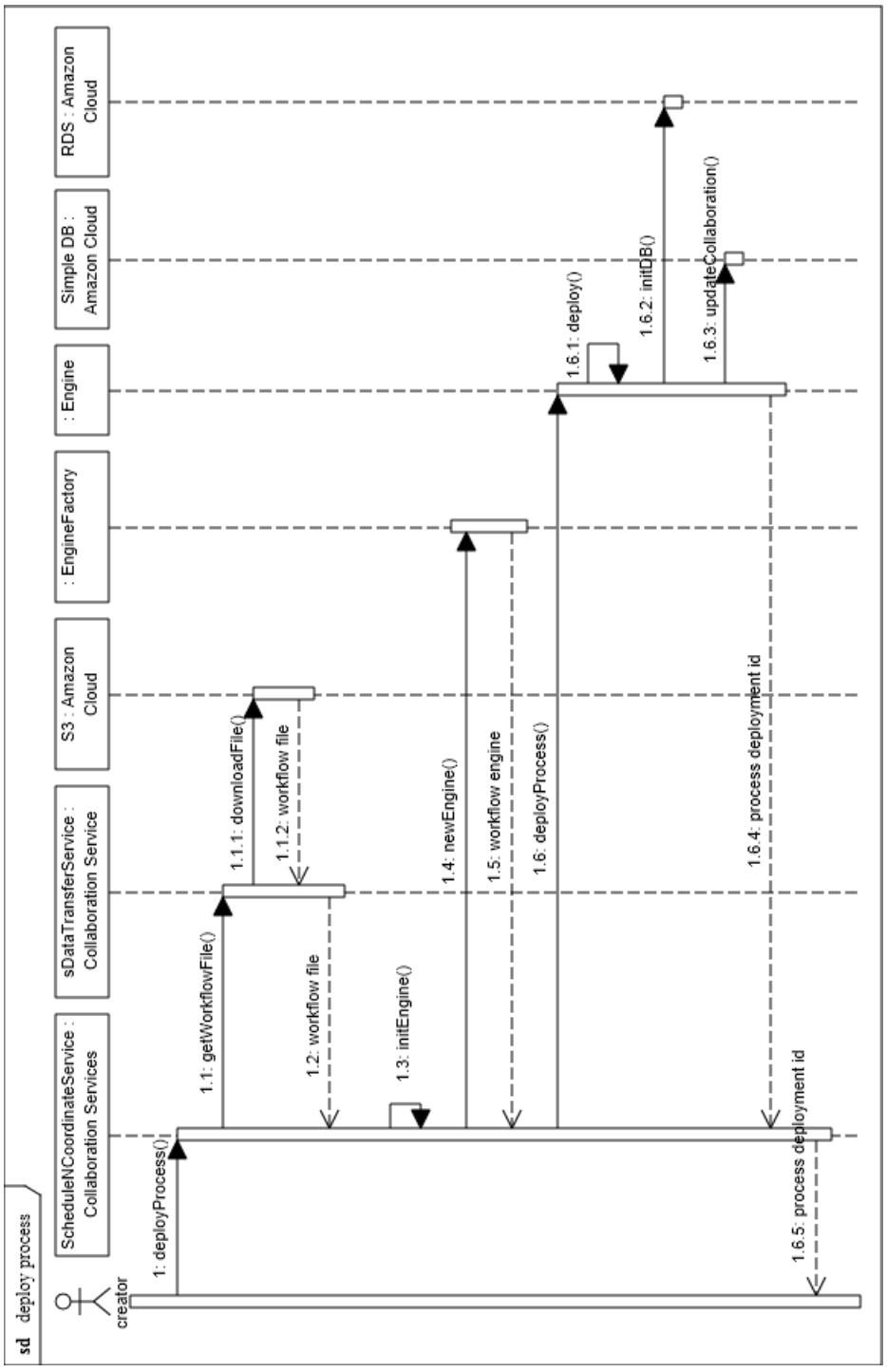

Figure 4.15: Sequence Diagram for Process Deployment

While finished deploying the process inside the engine, it is ready to run. jBPM provides a very convenient interface to do this as shown in the following piece of codes. Figure 4.16 shows the detail process of start a collaboration.

```
ProcessEngine processEngine = Configuration.getProcessEngine();
ExecutionService executionService = processEngine.getExecutionService();
executionService.startProcessInstanceById(processDeploymentId);
```

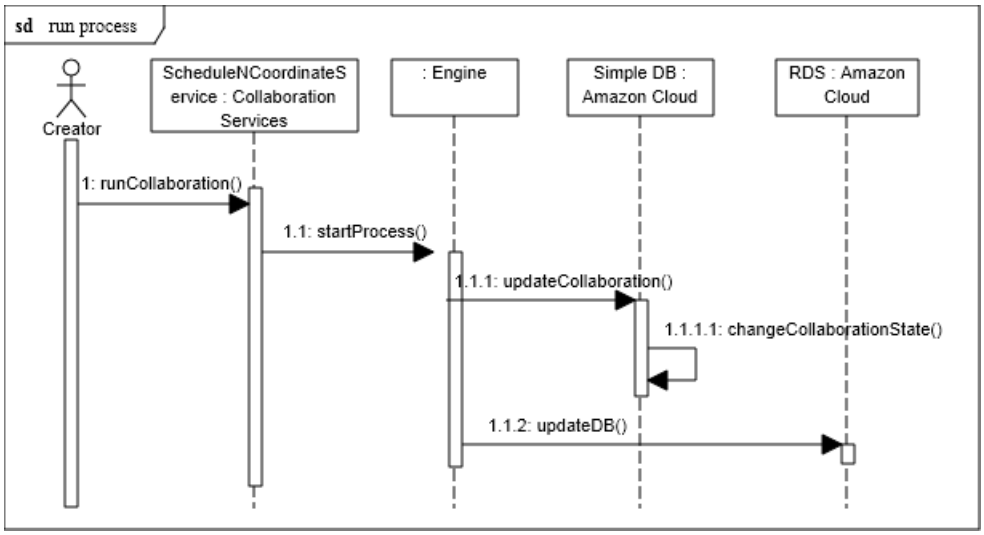

Figure 4.16: Sequence Diagram for Process Start

After successfully running the process, the state of the collaboration will be changed to be "running", and the current task will be displayed to the creator, as well as to the assigned participants. Participant can submit the task to signal the process keep going after done the task:

```
ProcessEngine processEngine = Configuration.getProcessEngine();
TaskService taskService = processEngine.getTaskService();
taskService.completeTask(taskId);
```

4.4.3 Data Transfer Service

During the collaboration, files will be transported from one participant to another and all these documents are stored in Amazon S3. So "Data Transfer Service" is to transport documents between peers and storage center in cloud.

4.4.3.1 Design

Data Transfer service provides two interfaces as shown in Figure 4.17 for users or other collaboration services to retrieve data from and upload data to Amazon S3.

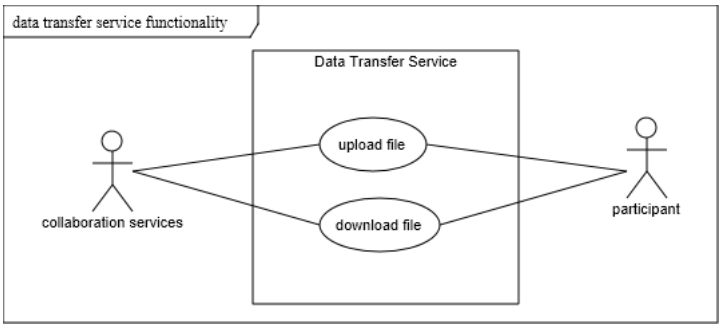

Figure 4.17: Data Transfer Service Functionality

The design codes of the interfaces:

```
@WebService
public interface DataTransferService {
    /**
     * upload a file to Amazon S3
     *
     * @param fileName
     * @param collaborationName
     * @param bytes
     * @return the key of the upload file and the bucket name
     */
    @WebMethod
    public String[] uploadFile(String fName, String cName, byte[] bytes);

    /**
     * download a file from Amazon S3
     *
     * @param bucketName
     * @param key
     * @return a file
     */
    @WebMethod
    public byte[] downloadFile(String bucketName, String key);
}
```

4.4.3.2 Implementation

The data is considered to be objects stored in bucket in Amazon S3 and each bucket is associated with an Amazon account and ownership can not be transferred. The framework will create a bucket for each peer collaboration with name in the form of "pcsf-bucket-<collaboration name>" which is uniquely used to identify a specific resource space for that certain collaboration. All documents that are needed for the collaborative activities are stored in that bucket. The bucket can not be deleted unless it is empty.

The objects in a bucket contain data along with an unique key. The key must be specified by users when uploading the data. And it will also be used to retrieve data from bucket in the future.

The following Java code example demonstrates how to upload data to Amazon S3. It provides users API to upload data that is from a file, or a stream:

```
AWSCredentials myCredentials = new PropertiesCredentials(DataTransferServiceImpl
    .class.getResourceAsStream(CREDENTIAL_FILE_PATH));
AmazonS3 s3Client = new AmazonS3Client(myCredentials);

if (!isBucketExist(bucketName)) {
    s3.createBucket(bucketName);
}

s3client.putObject(new PutObjectRequest(bucketName, keyName, file));
```

Similar like upload, Amazon also provides API to get data from the storage space. The data will be transported to users in the form of a stream and can be transferred to be needed format later on. The following Java code example demonstrates this process:

```
AWSCredentials myCredentials = new PropertiesCredentials(DataTransferServiceImpl
    .class.getResourceAsStream(CREDENTIAL_FILE_PATH));
AmazonS3 s3Client = new AmazonS3Client(myCredentials);

S3Object object = s3Client.getObject(new GetObjectRequest(bucketName, key));
InputStream objectData = object.getObjectContent();
// Process the objectData stream...
objectData.close();
```

4.4.4 Monitor and Control Service

The Monitor and Control Service is in charge of handling the exceptions. For example, when timeout exceptions occur during workflow activity, the service may determine what appropriate actions should be performed. This could be sending the task to the participant one more time after a certain time interval, or sending a message to all the participants informing that the task failed and has been terminated.

4.4.4.1 Design

The service will start automatically after being deployed as a collaboration service along with a peer collaboration. Also, it will stop running when the collaboration has been completed. Figure 4.18 shows the two interfaces that are provided by this service.

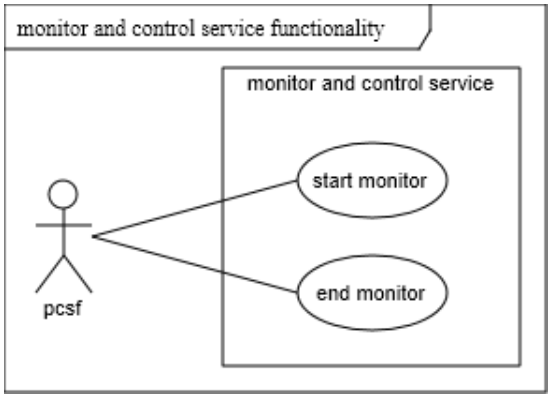

Figure 4.18: Monitor And Control Service Functionality

The design codes of the interfaces:

```
@WebService
public interface MonitorNControlService {
    /**
     * Start monitor and control service
     */
    @WebMethod
```

```
    public void startMonitor();

    /**
     * End monitor and control service
     */
    @WebMethod
    public void endMonitor();
}
```

4.4.4.2 Implementation

After monitor and control service is started, it will run recursively after a specified time interval. For example, every eight hours it will check the registration states of each participant and resend a notification email to those who have not complete registration. Or every two days it will check the state of current task and resend an email requiring task submission if it has not been done yet.

Each of these repeat activities are independent and are implemented as "Timer Task". As shown in the following example codes, all actions are included in "run()" method which will be executed recursively.

```
/**
 * Remind the participant to do registration after a certain time interval
 */
class RegistrationReminder extends TimerTask {
    @Override
    public void run() {
      // get the registration state of all participants
      // send an email to those who has not done registration
    }
}
```

To make the recursion happen, the service applies "Timer" which is part of Java API that provides a way to schedule "Timer Task" as shown in the following codes.

```
Timer timer = new Timer();
TimerTask registrationReminder = new RegistrationReminder();
TimerTask taskSubmitReminder = new TaskSubmitReminder();

// The registration reminder will occur 43200000 ms (0.5d) later, and after that
// will repeat every 86400000 ms (1d).
timer.schedule(registrationReminder, 43200000, 86400000);

// The task submit reminder will occur 86400000 ms (1d) later, and after that
```

```
// will repeat every 172800000 ms (2d).
timer.schedule(taskSubmitReminder, 86400000, 172800000);
```

Note that in above example, there are two timer tasks created and added to the timer schedule. The timer itself provides a multi-thread mechanism to schedule the tasks, so each task will run in an individual thread, focusing on their own work, and will not affect the others.

Furthermore, this also makes the monitor and control service extensible. When more functionalities are required, for instance, a new exception must be handled, a new "Timer Task" can be created and then added to the timer schedule as shown in Figure 4.19.

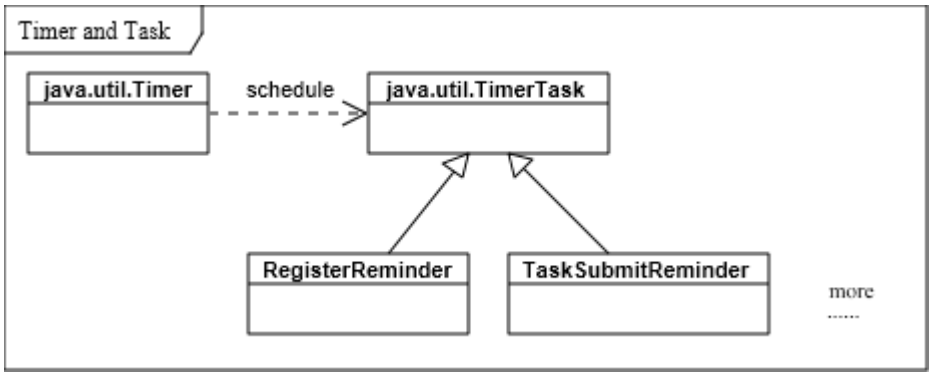

Figure 4.19: Timer And Task

Chapter 5

Case Study: Collaborative Testing

This chapter presents an example application about collaborative testing which is created and generated by our framework.

In this case study, a peer collaboration will be established for a test group, which will include a team manager, a test case designer, and test case programmer. Through this experiment, we will be able to find out how this framework can provide a systematic way to manage and facilitate network based dynamic peer collaborations.

In the rest part of this chapter, I will first give an introduction for collaborative testing and application in section 5.1, which might include how various roles in the team work together to finish a testing job. Next in section 5.2, the CaaS realization will be given to show the services that are used for this collaborative testing and their responsibilities, as well how these services work together to facilitate the collaboration. In section 5.3, Amazon Implementation will be presented which will include the required resources that will be provisioned from Amazon cloud. Finally in section 5.4, we will give several experiments on how the generated application works internally and its performance.

5.1 Collaborative Testing and Application

After a company receives a new testing project, it needs to form a group to accept this work. In this formed collaborative testing group, we have several participants with different roles and responsibilities working together to complete this task, which might include:

- Test Case Designer
- Page Library Designer
- Testing Developer
- Testing System Manager
- Group Manager

A test case is usually an exemplification scenario that will probably occur to the system in the future real world. It contains a set of operations or activities to determine if the software or application works correctly and to make sure the functionalities the system provides match the requirements from the clients. Therefore, the **Test Case Designer** must have a deep understanding of the requirements and the system and might not have any programming technique knowledge. A typical test case is as shown in Figure 5.1.

The page library is originally from the concept "Page Objects" by Selenium [51]. A page library will hide all the UI details of a page and only expose activities of that page to clients. The activities might consist of a set of operations on the web page. Page library can be considered to be a bridge between developers who deeply knows the structure of web pages and testers who are mainly focusing on test case implementation. For instance, the page library might provide a method "login" for testers to invoke and this activity includes three steps: type in username, type in password, and click login button. All these detail operations are transparent to the

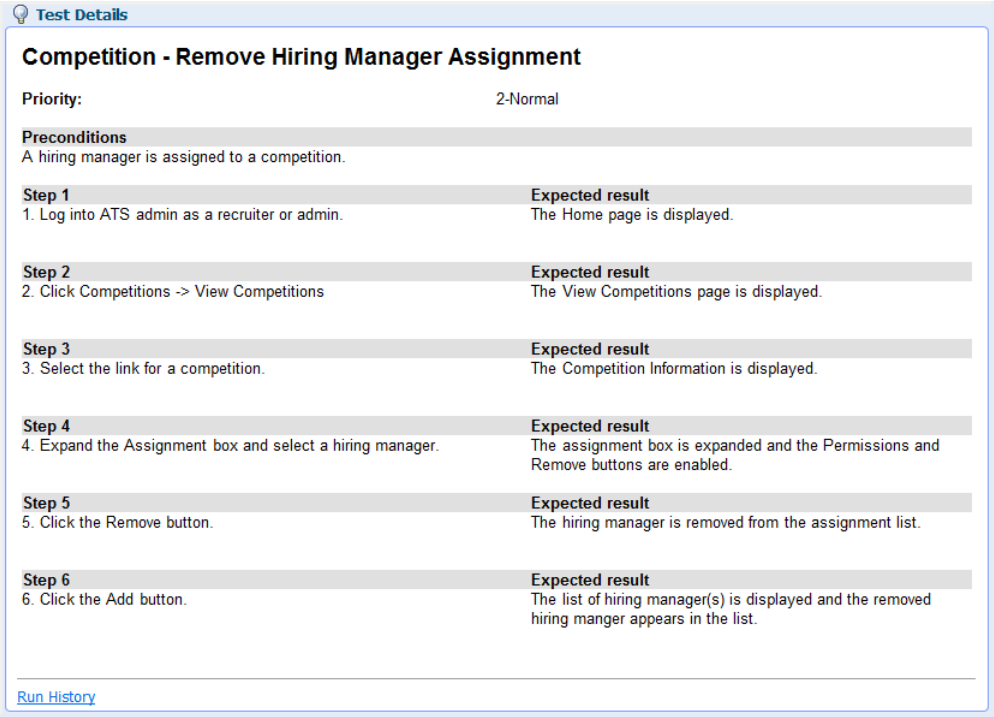

Figure 5.1: Example Of Test Case Design

clients. The tester does not need to know which text field on the page to type in or which button to click. If the UI layout of that page changes in the future, only the page library is needed to be modified and the tester will still use the same method "login" as before.

The page library is created by the **Page Library Designer**, who has a good HTML or web programming knowledge background. The **Testing Developer** is in charge of implementing test cases with the help of the interfaces provided by the page libraries. The completed test cases will then be submitted to the **Testing System Manager** to execute and the system finally will generate a testing report and send it to the **Group Manager**. By the end of this point, the collaborative testing is done, and if there are new requirements from the clients, the collaboration can run again among these participants.

5.2 CaaS Realization

We are going to generate the above collaborative testing application using our framework. As we introduced in Chapter 2, the CaaS is in charge of generating collaborations. In this layer, there are two services: creation service and instance management service. More accurately, the creation service is the one that will generate collaborations and the instance management service is used to manage the life cycle of the generated collaborations.

To generate this collaborative testing application, the collaboration specification must be provided to the creation service, which is a predefined execution model that is expressed in the form of a workflow file so as to direct the collaborative activities.

Firstly we give a list of roles and groups. Each of the tasks defined in the workflow file will be assigned to one of these roles or groups. The list is as shown in Table 5.1.

Table 5.1: The List of Roles and Groups

Role	Group
Test Case Designer	Designers
Page Library Designer	Designers
Testing Developer	Developers
Testing Developer	Developers
Testing System Manager	Managers
Group Manager	Managers

Next Figure 5.2 shows the overall process which is defined by a BPMN workflow file.

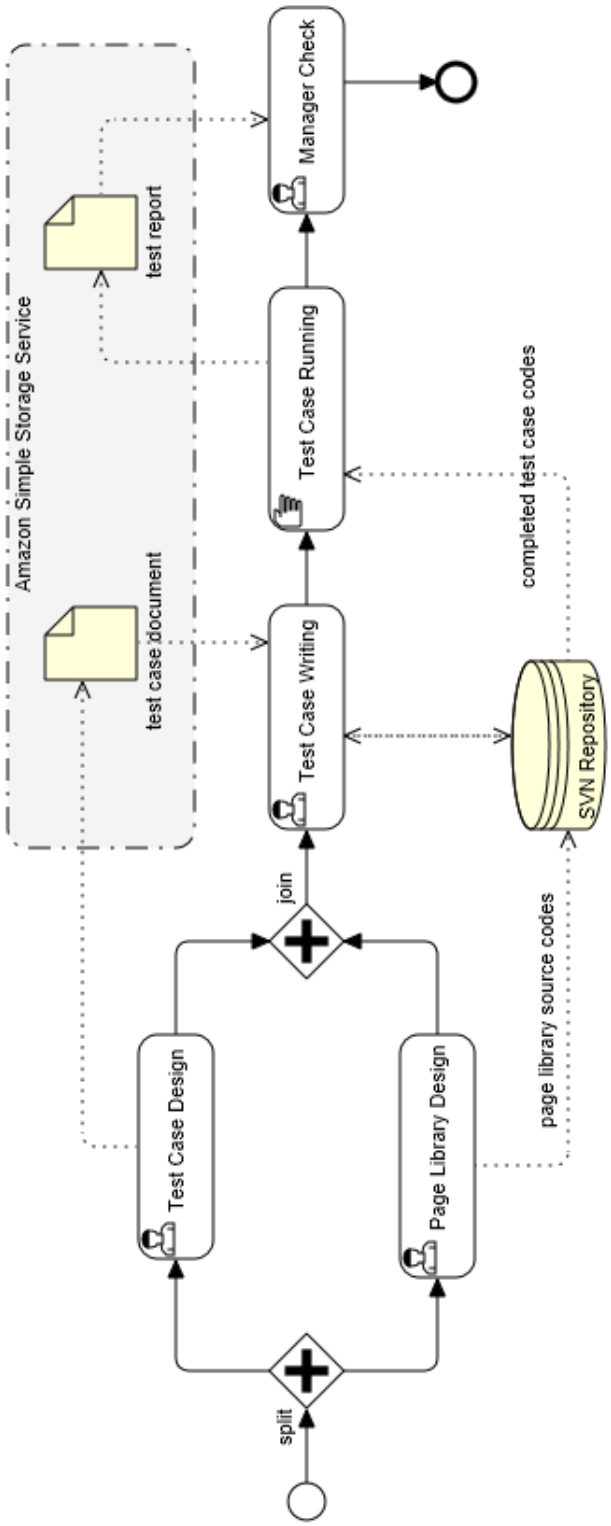

Figure 5.2: Workflow Diagram

The source code of the workflow file:

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions id="case_study"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://schema.omg.org/spec/BPMN/2.0../../../../../
        ../../../../../bpmn/src/main/resources/BPMN20.xsd"
    xmlns="http://schema.omg.org/spec/BPMN/2.0"
    typeLanguage="http://www.w3.org/2001/XMLSchema"
    expressionLanguage="http://www.w3.org/1999/XPath"
    targetNamespace="http://jbpm.org/example/bpmn2/case study"
    xmlns:jbpm="http://jbpm.org/bpmn2">

    <process id="collaborativeTesting" name="Collaborative Testing">
        <startEvent id="Start"/>
        <sequenceFlow id="flow1" name="fromStartToSplit" sourceRef="Start"
            targetRef="parallelGatewaySplit"/>

        <parallelGateway id="parallelGatewaySplit" name="Split"
            gatewayDirection="diverging"/>

        <sequenceFlow id="flow2a" name="toTestCaseDesign"
            sourceRef="parallelGatewaySplit" targetRef="Test Case Design"/>

        <userTask id="Test Case Design" name="Test Case Design">
            <humanPerformer>
                <resourceAssignmentExpression>
                    <formalExpression>test case designer</formalExpression>
                </resourceAssignmentExpression>
            </humanPerformer>
        </userTask>

        <sequenceFlow id="flow2b" name="fromTestCaseDesignToJoin"
            sourceRef="Test Case Design" targetRef="parallelGatewayJoin"/>

        <sequenceFlow id="flow3a" name="toPageLibraryDesign"
            sourceRef="parallelGatewaySplit" targetRef="Page Library Design"/>

        <userTask id="Page Library Design" name="Page Library Design">
            <humanPerformer>
                <resourceAssignmentExpression>
                    <formalExpression>page library designer
                    </formalExpression>
                </resourceAssignmentExpression>
            </humanPerformer>
        </userTask>

        <sequenceFlow id="flow3b" name="fromPageLibDesignToJoin"
            sourceRef="Page Library Design" targetRef="parallelGatewayJoin"/>

        <parallelGateway id="parallelGatewayJoin" name="Join"
            gatewayDirection="converging"/>
```

```
        <sequenceFlow id="flow4" sourceRef="parallelGatewayJoin"
            targetRef="Test Case Writing"/>

        <userTask id="Test Case Writing" name="Test Case Writing">
            <potentialOwner>
                <resourceAssignmentExpression>
                    <formalExpression>developers</formalExpression>
                </resourceAssignmentExpression>
            </potentialOwner>
        </userTask>

        <sequenceFlow id="flow5" sourceRef="Test Case Writing"
            targetRef="Test Case Running"/>

        <manualTask id="Test Case Running" name="Test Case Running">
            <humanPerformer>
                <resourceAssignmentExpression>
                    <formalExpression>testing system manager
                    </formalExpression>
                </resourceAssignmentExpression>
            </humanPerformer>
        </manualTask>

        <sequenceFlow id="flow6" sourceRef="Test Case Running"
            targetRef="Manager Check"/>

        <userTask id="Manager Check" name="Manager Check">
            <humanPerformer>
                <resourceAssignmentExpression>
                    <formalExpression>group manager</formalExpression>
                </resourceAssignmentExpression>
            </humanPerformer>
        </userTask>

        <sequenceFlow id="flow7" sourceRef="Manager Check" targetRef="End"/>

        <endEvent id="End" name="End">
            <terminateEventDefinition/>
        </endEvent>
    </process>
</definitions>
```

As depicted in the above workflow diagram and file, there are five tasks in this collaborative testing and each has been assigned to a certain role or group to complete. For each collaborative testing instances, the creator must also specify a list of concrete participants. An example of the participant list with assigned tasks is listed in Table 5.2.

With this information, the creation service will generate a collaboration together

Table 5.2: Task List

Participant	**Role**	**Group**	**Assigned Task**
Adam	Test Case Designer	Designers	Test Case Design
Jacob	Page Library Designer	Designers	Page Library Design
David	Testing Developer	Developers	Test Case Writing
Stephen	Testing Developers	Developers	Test Case Writing
Parker	Testing System Manager	Managers	Test Case Running
Chris	Group Manager	Managers	Manager Check

with a set of collaboration services running at the back end to support it. The **Registration Service** will inform each participant to register into this collaboration. The **Instance Management Service** will start or stop the collaboration after all participants done registration. The **Schedule and Coordinate Service** will distribute the tasks and documents to the right participants. The **Monitor and Control Service** will handle the exceptions occur in the middle of the collaboration. The **Data Transfer Service** will transport data between participants.

5.3 Amazon EC2 Implementation

The framework is deployed in Amazon cloud and applies a couple of cloud services that Amazon offers. The benefit of deploying in cloud is to dynamically provision compute resources when the load of tasks increase. For this case study, one Amazon EC2 instance is launched to host the framework and one Amazon RDS instance is created to act as database storage as described in Table 5.3.

Besides the above two launched instances, some other Amazon web services the framework employs are also listed in Table 5.4 together with the one-year free usage limitation for new customers. All these free limited resources are sufficient for our research purpose.

Table 5.3: Characteristics of Amazon EC2 and RDS Used in Case Study

	Amazon EC2	Amazon RDS
Platform	Ubuntu 12.04 Server LTS 64-bit	MySQL 5.5.27 64-bit
CPU(type)	1 EC2 Compute Unit*	1 EC2 Compute Unit
	Micro Instance(t1.micro)	Micro DB Instance(db.t1.micro)
Memory	613 Mb	630 Mb
Instance Storage	8 Gb EBS storage	20 Gb
Instance Location	US-east-1a	US-east-1d
Number of instances	1	1
Price per hour	free**	free

* 1 EC2 compute unit (one virtual core with one EC2 compute unit)
** new Amazon web service customers can get started with Amazon EC2 for free with limited compute resources for one year. For example, free 750 hours of EC2 running Linux/Unix Micro instance usage each month. Refer to [12] for more information.

Table 5.4: Other Amazon Web Services Used in Case Study

Service Name	Free Usage Limitation
Amazon Simple Storage Service	5 GB of Amazon S3 standard storage, 20,000 Get Requests, 2,000 Put Requests, and 15GB of data transfer out each month for one year.
Amazon Simple DB	25 SimpleDB Machine Hours and 1 GB of Storage for free each month
Amazon Simple Email Service	send 2,000 messages for free each day when calling Amazon SES from an Amazon EC2 instance

5.4 Experiments

In the following section, we will design six scenarios showing how to build a collaborative testing instance and manipulate it.

5.4.1 Scenario 1: Create A Collaboration

To create the example peer collaboration, a collaboration name and a workflow file must be given as shown in Figure 5.3.

The details of the successfully created collaborations will be listed in a table showing the creator as shown in Figure 5.4, along with a couple of command buttons and links to operate.

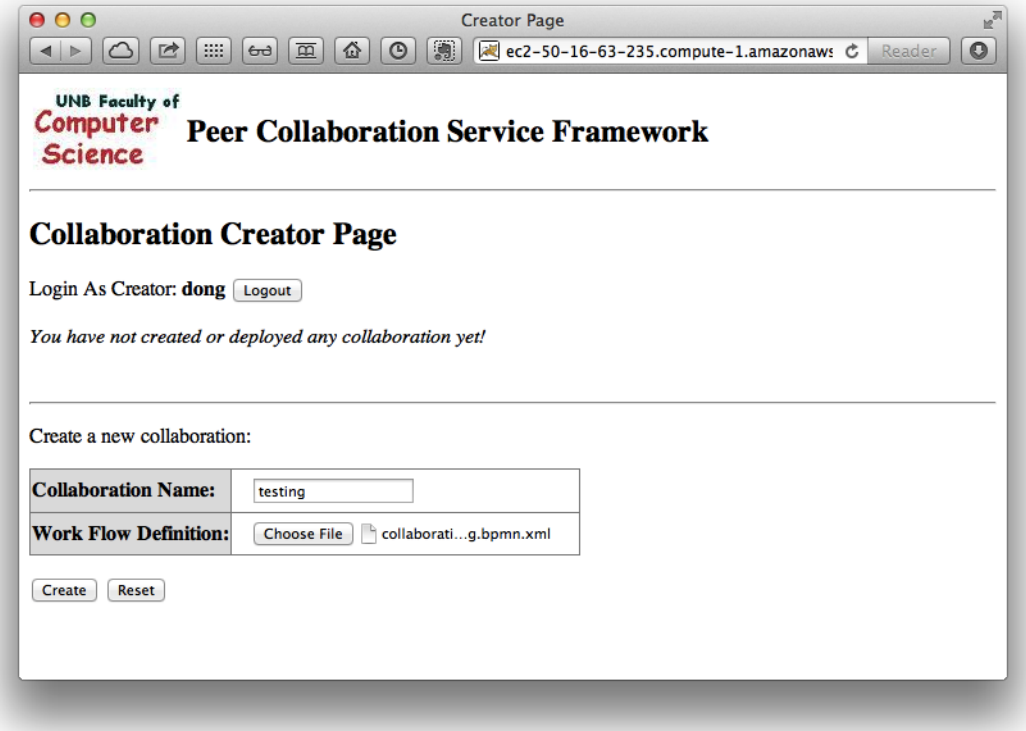

Figure 5.3: Screenshot for Creator Page: Ready to Create Collaboration

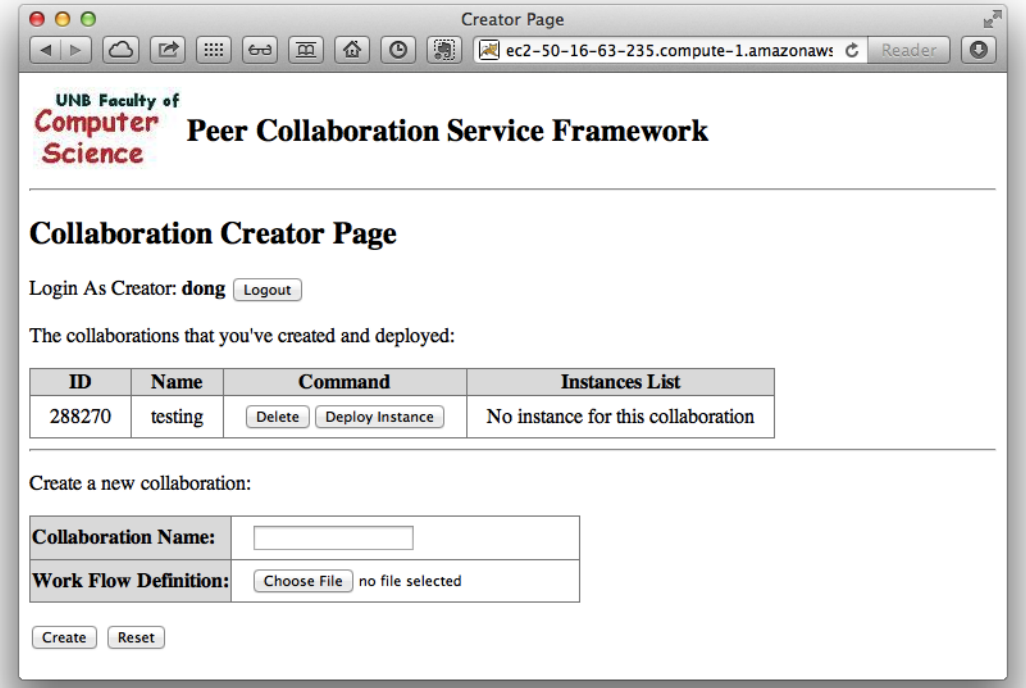

Figure 5.4: Screenshot for Creator Page: Collaboration Created

By this step, a collaboration has been created which is combined with a workflow file. The workflow file has been uploaded and stored in Amazon S3 when opening the view page for S3 service as shown in Figure 5.5. Using this collaboration, users can generate multiple collaboration instances, all of which share the same workflow model.

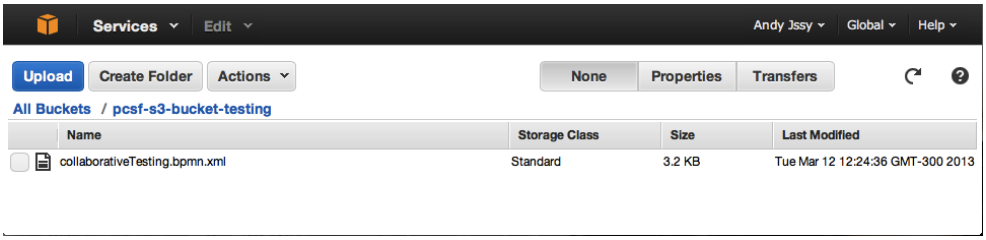

Figure 5.5: Screenshot for Amazon S3 View

5.4.2 Scenario 2: Generate A Collaboration Instance

By clicking the "Deploy Instance" button on the page, the framework will generate a collaboration service instance. To make the button work, an environment variable "COL_SVR_DEPLOY_LOC" must be added to the system and the value should be the location of the server so as to direct the system to deploy the instance to the right place. Figure 5.6 shows that three collaboration instances have been generated. After successfully deploying a collaboration instance, an email will be sent to the creator.

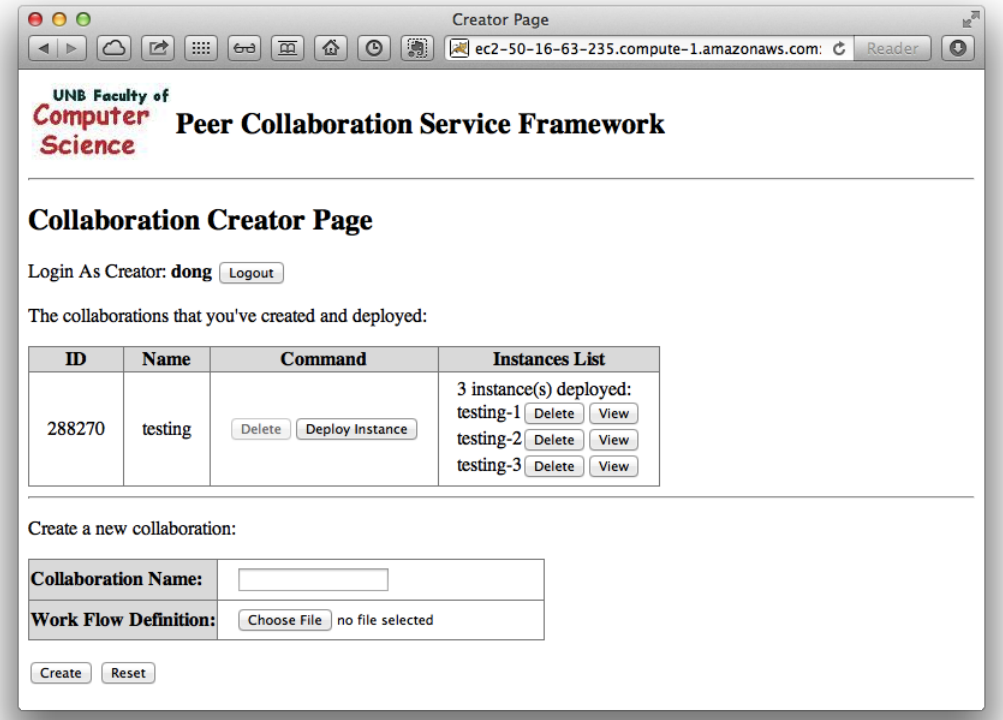

Figure 5.6: Screenshot for Creator Page: Instance Generated

Notification Email

```
Dear dong,

You have deployed a collaboration service instance, an email has been sent to
each participant and you can run the collaboration after all participants done
registration.

Click the following link to check the collaboration:
http://ec2-50-16-63-235.compute-1.amazonaws.com:8080/pcsf/index.jsp?
action=creatorLogin
```

The next step is to specify a participant list for the instance. Although all instances generated from one collaboration share the same workflow model, they may have different participants to finish the jobs. Figure 5.7 shows how to add participants to a collaboration instance. Some necessary information should be provided while adding a participant. Every participant has to be assigned a role but the group can be optional.

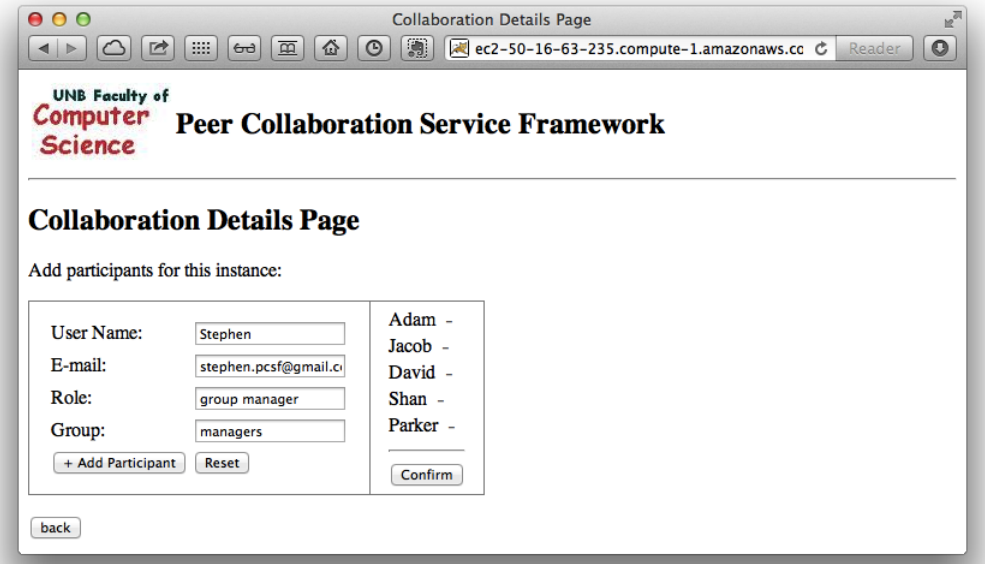

Figure 5.7: Screenshot for Creator Page: Add Participant

After confirming the list, all the participants information will be added to Amazon Simple DB as shown in Figure 5.8. The collaboration can not start until all participants complete registration as shown in Figure 5.9 after opening the view page of the collaboration instance.

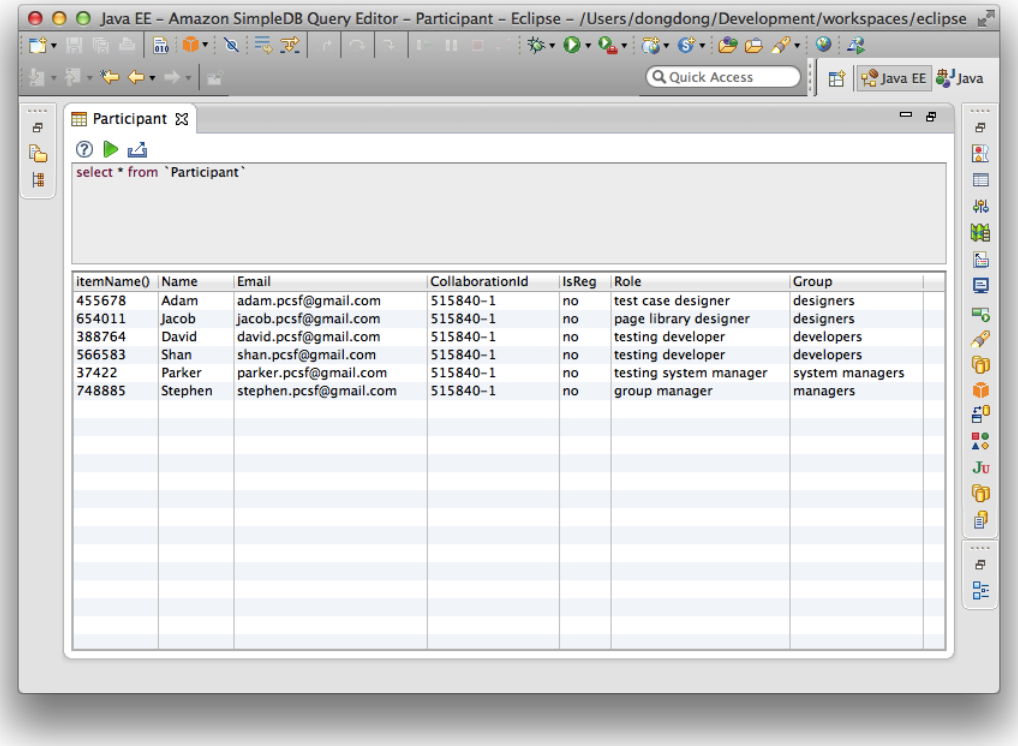

Figure 5.8: Screenshot for Amazon Simple DB View

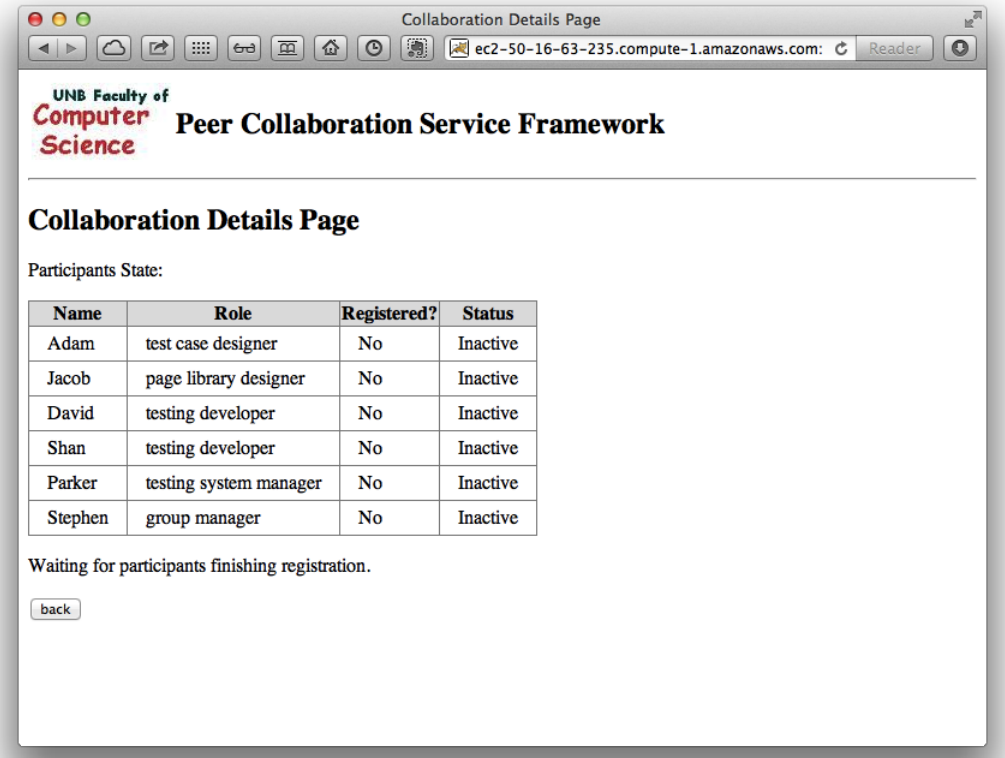

Figure 5.9: Screenshot for Creator Page: Waiting For Registration

5.4.3 Scenario 3: Participant Registration

At the same time the collaboration instance has been generated, each participant will also receive a notification requiring registration. An unique id will be assigned to each participant who will use it to login to the system to finish registration and view the assigned tasks.

Figure 5.10 shows the participant page and displays who has not completed registration, and Figure 5.11 shows that registration had been completed. The task list is empty because there are still participants who have not finished registration and the collaboration has not started yet.

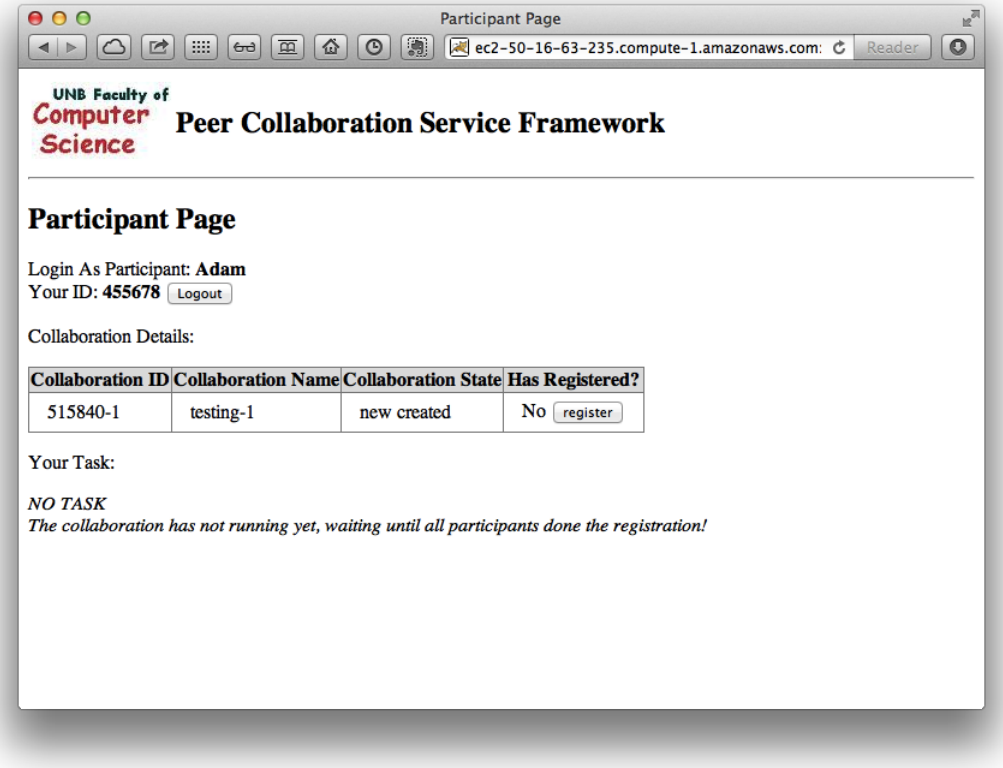

Figure 5.10: Screenshot for Participant Page: Not Registered

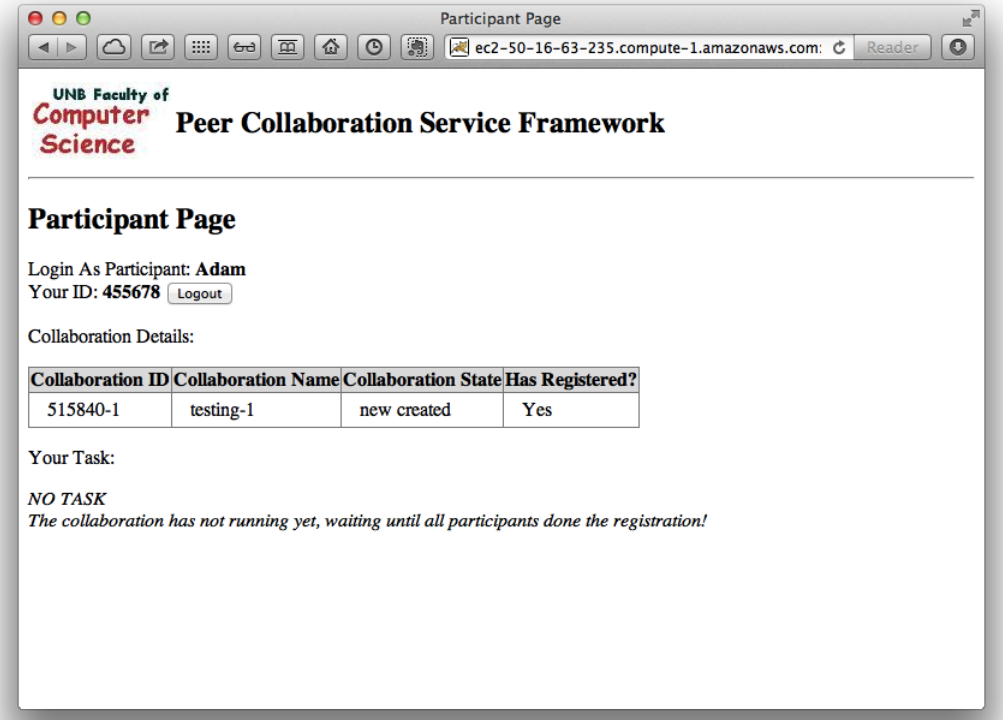

Figure 5.11: Screenshot for Participant Page: Done Registration

5.4.4 Scenario 4: Run Collaboration Instance

When all participants have completed the registration, the process will be initialized inside the workflow engine and the creator can run the collaboration from the view page as shown in Figure 5.12. Clicking the run button will start the collaboration. The process file will be executed by the workflow engine and the collaboration service will start to distribute tasks to participants.

Figure 5.13 shows two tasks are listed in the current task table after the collaboration starts. It matches the worlkflow diagram (Figure 5.2) that two parallel tasks will run at the beginning of the process. Note that the "Assignee" in the table represents the assigned "Role", because there might be more than one concrete participant with the same role.

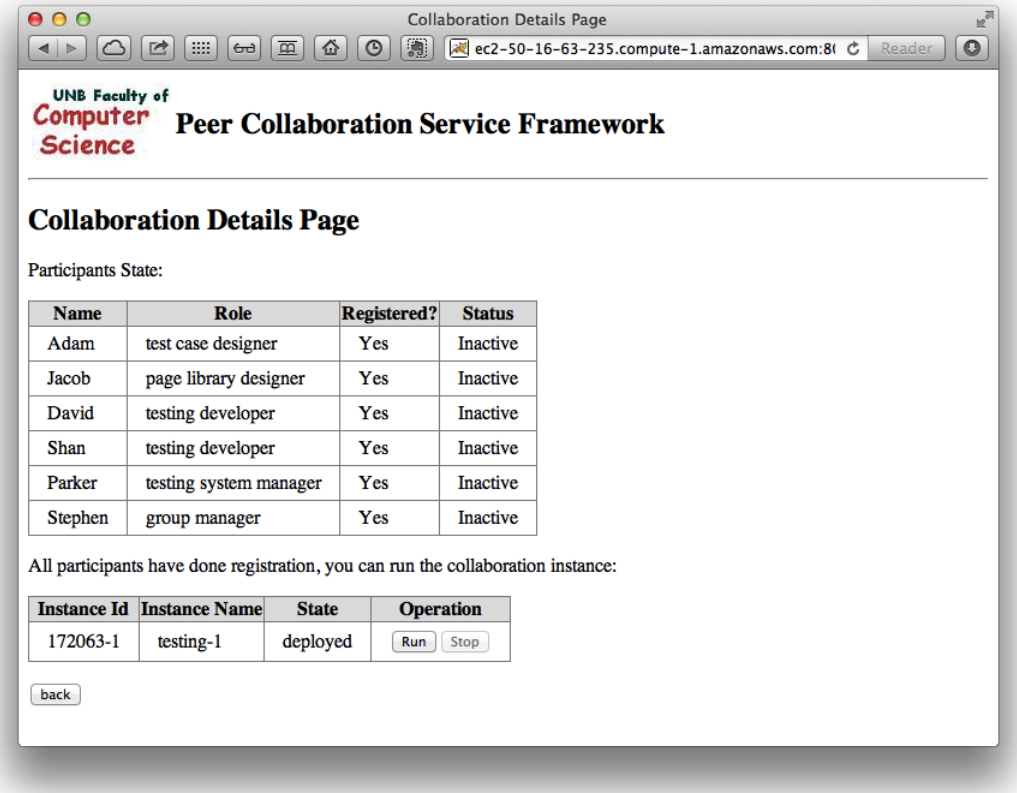

Figure 5.12: Screenshot for Creator Page: Ready To Run The Collaboration

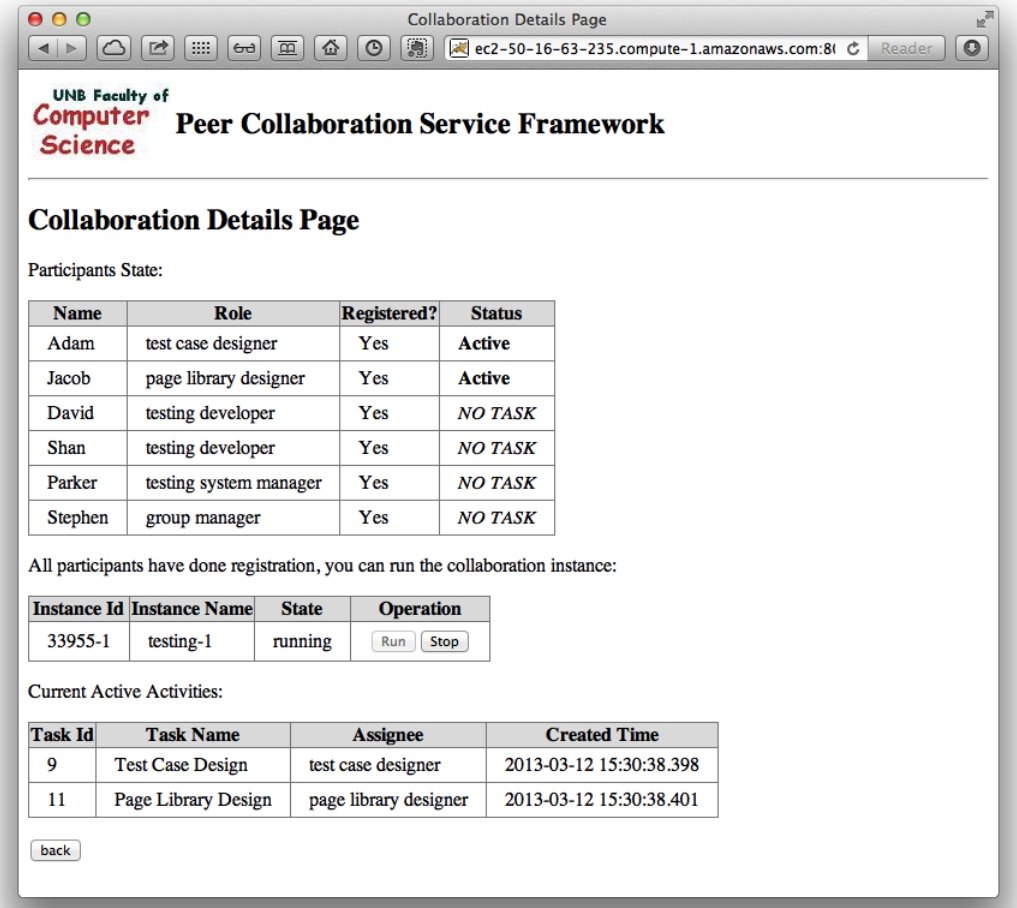

Figure 5.13: Screenshot for Creator Page: Running Collaboration

5.4.5 Scenario 5: Submit Task

When the collaboration has started, participants can see their assigned tasks after login to the system as shown in Figure 5.14. If a participant has a role "testing developer" as well as in group "managers", all tasks related to that role and group will be listed in the table.

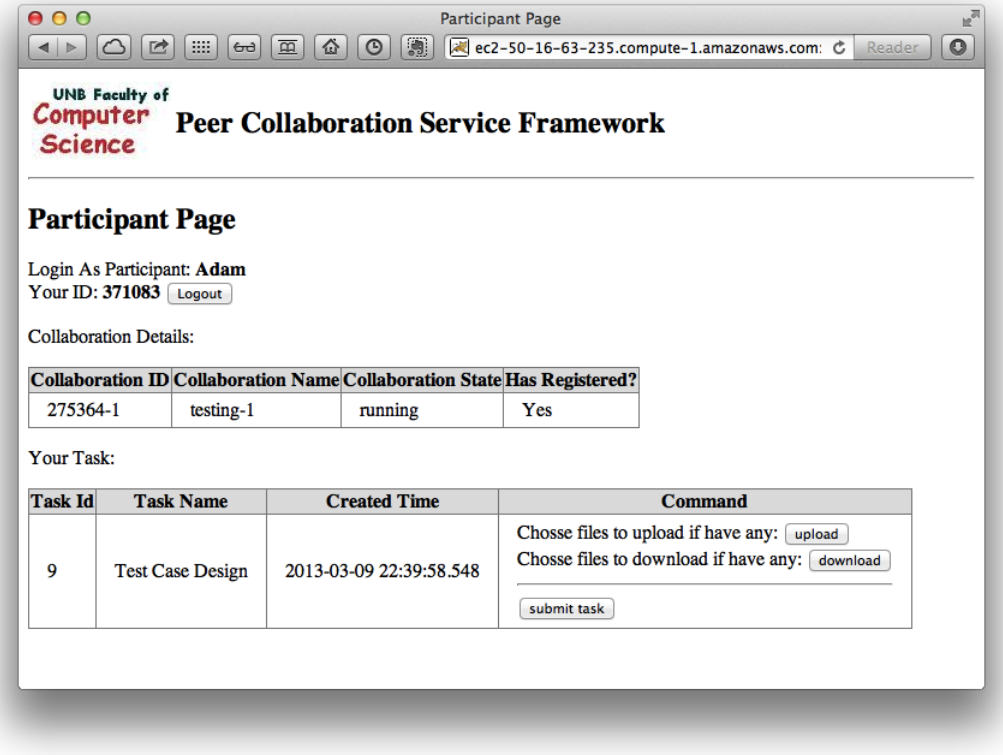

Figure 5.14: Screenshot for Participant Page: View Task

Users can also upload or download any documents that are needed to finish the task. Figure 5.15 shows the file upload page, and Figure 5.16 shows the file download page. The documents will be uploaded to or downloaded from Amazon S3. To download documents from S3, the user must specify the bucket name and the key, which is usually the same with the file names.

While uploading a document, the bucket name and the key related to that document will be sent by email to the next participant, so if any one needs to modify

the document, he or she can use this information to get the data.

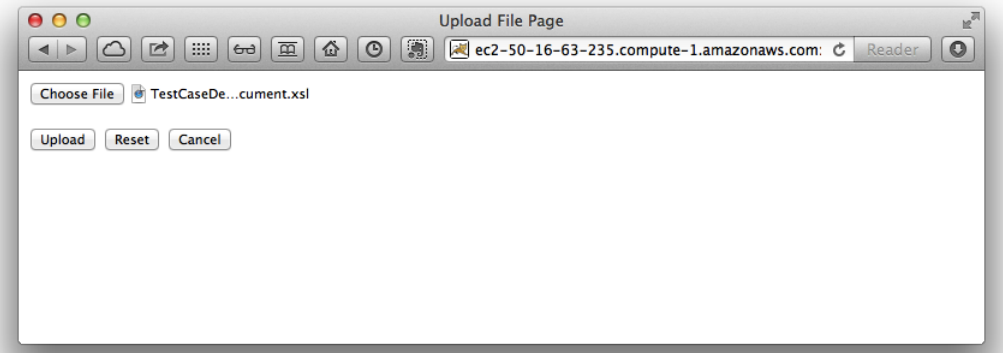

Figure 5.15: Screenshot for File Upload Page

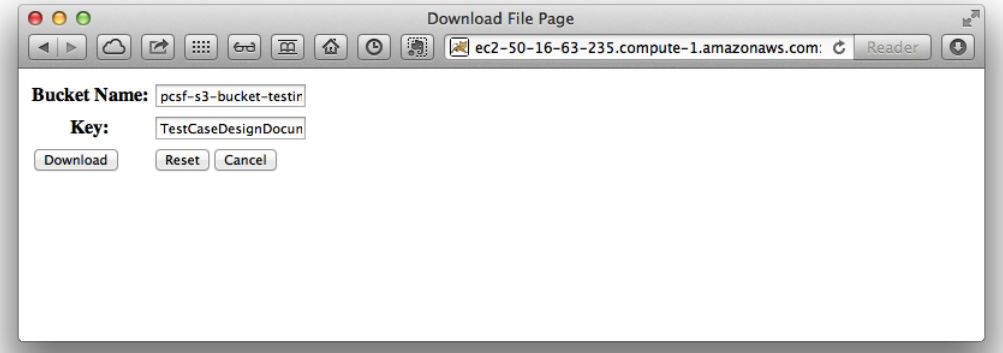

Figure 5.16: Screenshot for File Download Page

Clicking "submit task" button will end the current task and signal the process to move on to the next one until all tasks have completed and the collaboration will be set to be "finish".

5.4.6 Scenario 6: Multiple Collaborations and Instances

The last experiment is to verify that the framework can generate more than one peer collaborations, and each of the peer collaboration can generate more than one collaboration instances. In addition, these instances will run independently on cloud engine.

To set up this experiment, we create a peer collaboration "testing" using workflow file "collaborativeTesting.bpmn.xml" which specify a collaborative testing progress as what we did before, and another peer collaboration "leave" using workflow file "leave.bpmn.xml" which specify the progress of requesting a leave. With collaboration "testing", we generate two instances, and with collaboration "leave", only one instance has been generated as shown in Figure 5.17.

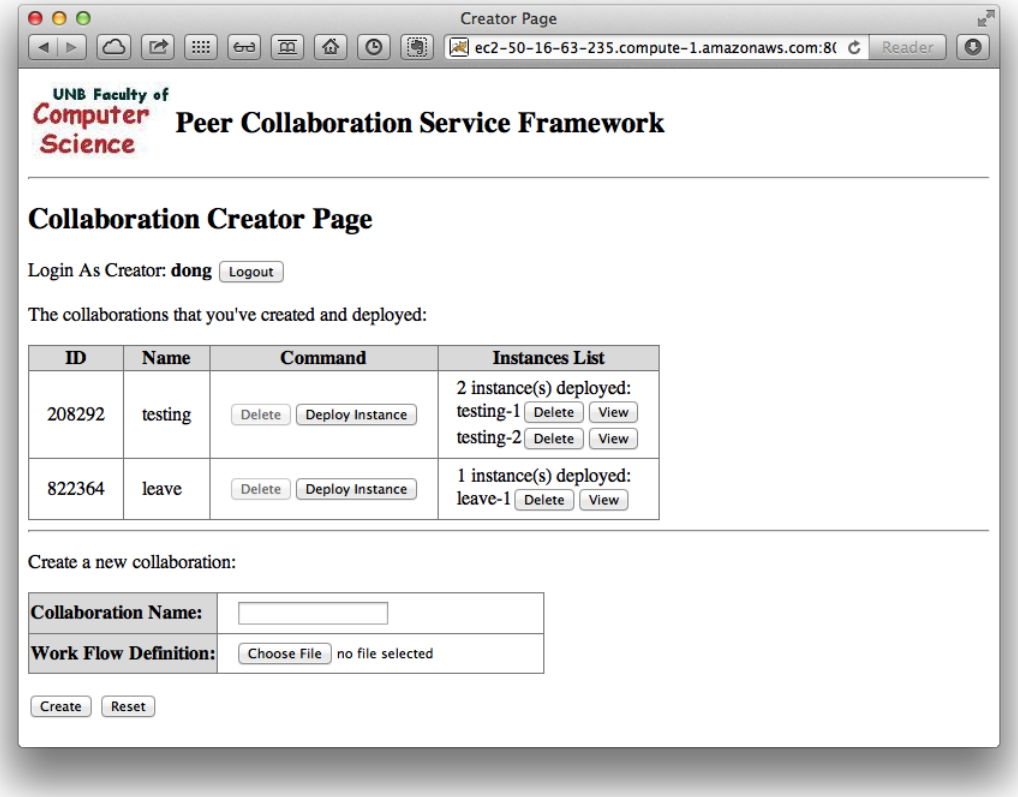

Figure 5.17: Screenshot for Creator Page: Multiple Collaboration and Instance

For instance “testing-1” and “testing-2”, we assign two different participant list to them. Similar with scenario 2, we step into each instances and add participants for them. Then every participant will receive a notification email requesting to register into corresponding collaboration instance like scenario 3. After completing the registration, we can run each instances. Figure 5.18, 5.19, and 5.20 show that these three instances are running independently and normally without mutual interruption.

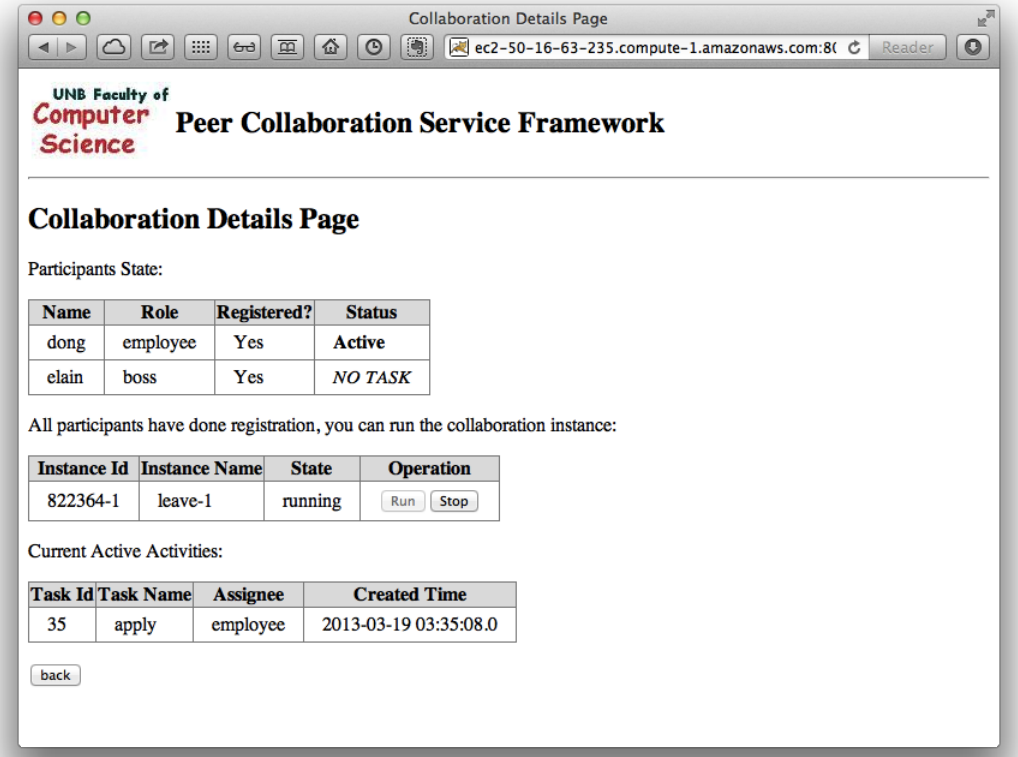

Figure 5.18: Screenshot for Creator Page: Running Instance “leave1”

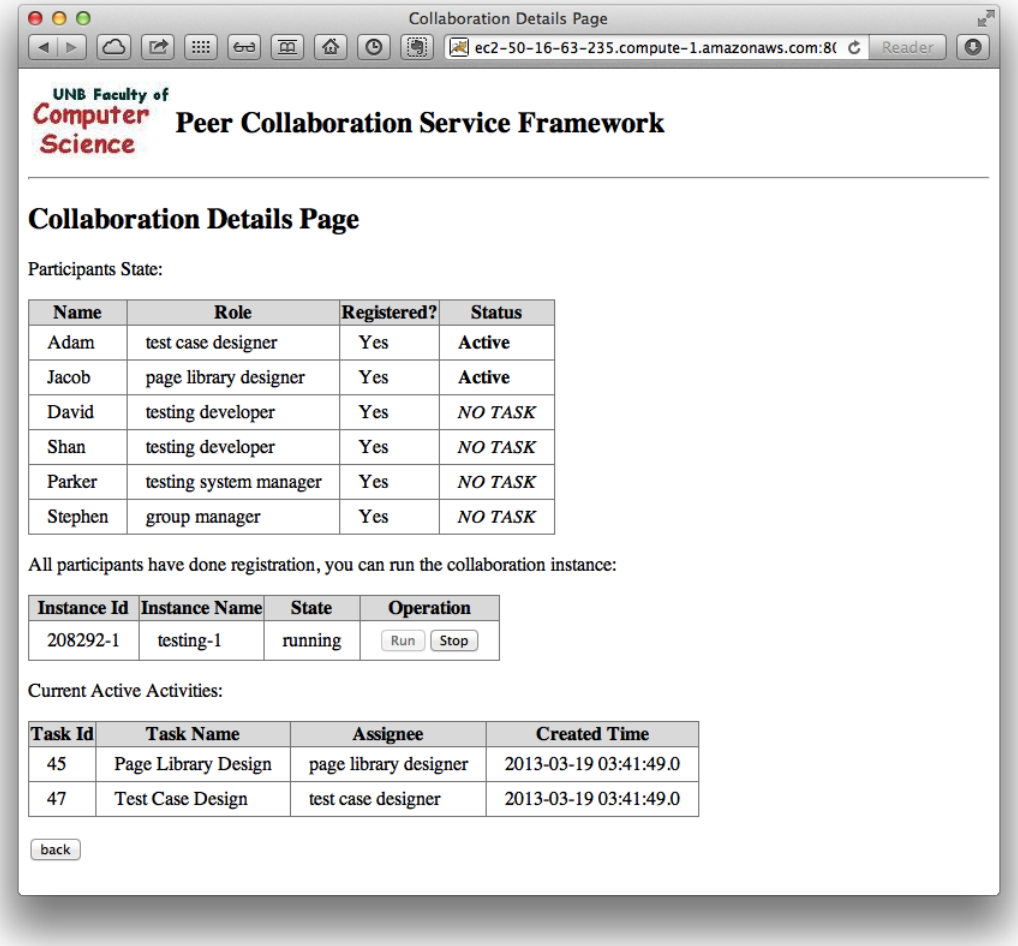

Figure 5.19: Screenshot for Creator Page: Running Instance “testing1”

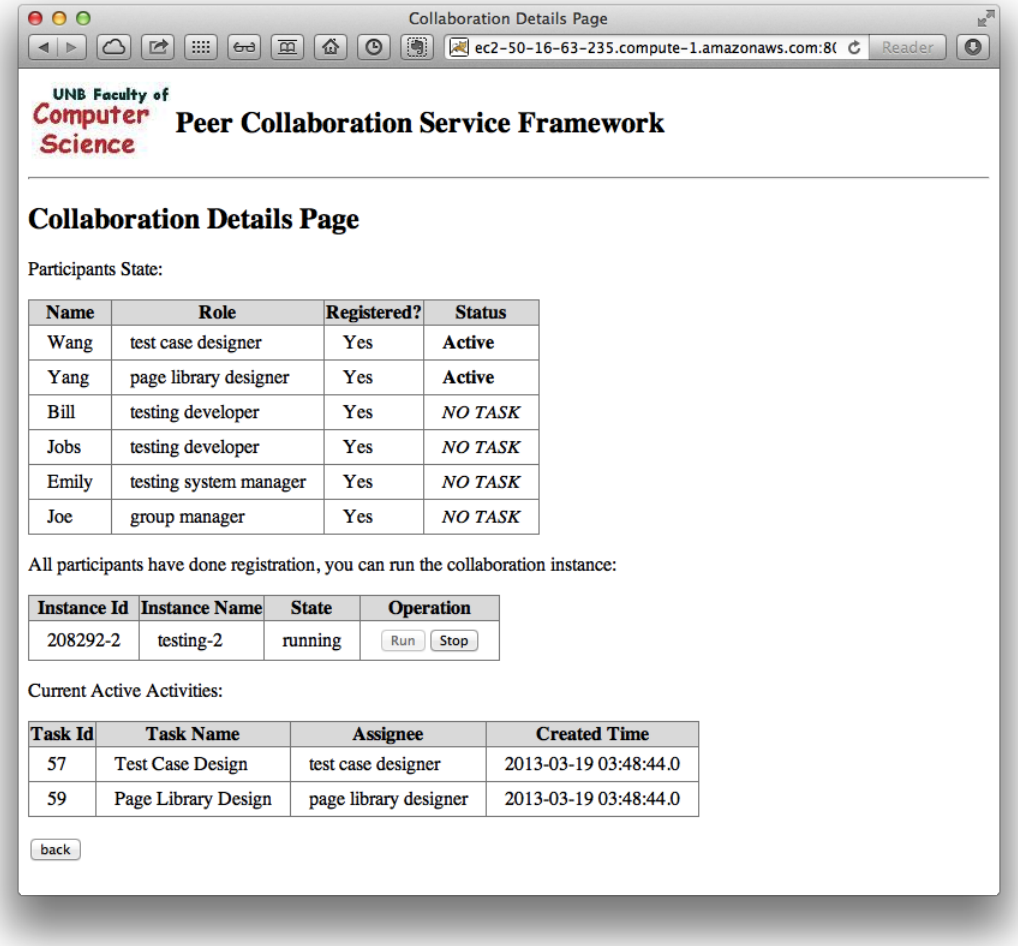

Figure 5.20: Screenshot for Creator Page: Running Instance “testing2”

Chapter 6

Evaluation

In this chapter, we are going to evaluate our framework. Firstly in section 6.1, we will discuss functionality, which includes what the collaboration system should support and what CaaS should support. Next in section 6.2, we will elaborate some required quality criteria, which might consist of reliability, security, privacy, performance, usability, and so on. Some of these aspects are realized in our framework but some are not covered.

6.1 Functionality

6.1.1 What Collaboration Systems Support

Collaboration systems are designed to help individuals working together to achieve a common task or goal. Accomplishing the goal is the main purpose to bring these participants together. The individuals might locate over geographic distances or work at different times, so the collaboration systems provide an integrated environment for them to cooperate together and also provides tools for accessing, creating, gathering, managing, storing, and exchanging information. Besides this, collaboration systems also support project management functionality, which might

include task assignment, task scheduling, shared calendars, and so on.

In general, the functionalities that collaboration systems support can be summarized as three different aspects:

- It supports communication, which refers to information exchange or message passing. Examples of this are email systems, and IM chatting applications.
- It supports collaboration, which refers to information sharing and interactive work towards a shared target. Examples of this are voting systems and video conferencing.
- It supports coordinate, which refers to business process automation. An example of this is workflow management systems.

6.1.2 What CaaS Supports

CaaS is designed to deliver collaboration as a service to users. It brings participants together without concern for their locations and creates a collaboration team for them to accomplish a common task. At the same time, it also generate a group of software services to help them finish the work. Simply saying, CaaS supports two functionalities: create peer collaborations and manage peer collaborations.

To create a peer collaboration, CaaS allows users to predefine a participant list and a workflow process to configure the collaboration. So collaborations can be specified differently by combining with different workflow models.

To manage a peer collaboration, CaaS generates a set of software services along with the collaboration to facilitate it. These services provide functionalities including:

- Register participant into collaboration.
- Assign tasks to participants and schedule the working process.

- Monitor collaborative activities and handle exceptions.
- Transfer data between participants.

6.2 Quality

To evaluate our framework, we first predefine some criteria on how to measure the quality. This will consist of the following several aspects: reliability, security, privacy, performance, modifiability, usability. So next, we are going to discuss our framework in terms of these creteria.

6.2.1 Reliability

Reliability measures the likelihood of potential system failure and to reduce the risk of system downtime. Our framework is deployed in Amazon cloud, and Amazon has servers running 24 times 7 hours every week in different regions all over the world, so even if one server is down in one place, the replica will be replaced immediately from other available regions.

The status of each collaboration is stored in a database in cloud, as well each process history record. So if the collaboration is interrupted in the middle of processing, the system will recover it from the database.

6.2.2 Security

Security determines how much the users can trust the system to put their secrets or safe data into it. For our framework, in order to create peer collaborations, a user must register himself as a creator and only after that can he or she create, view and manage the collaborations. Each participant will be assigned a random id by the framework, with which to view and submit their tasks. The data or documents that are needed to transfer between participants are stored in Amazon storage service.

The data or documents per se are not encrypted, but to access the storage space, a user must provide a valid access key and a valid secret access key. Besides that, each document has its own specified id and belonged bucket id. The user must also provide this information to locate the data and get it.

6.2.3 Performance

Performance is "the degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage." [1]

The experiments in previous chapters are running in resources from Amazon cloud and there are no subtle delays or waiting. The resources that have been applied are very limited with free tier agreement. Anyway, while the load increases, the resources will automatically scale up to fit the requirements, such as process speed, memory space, network I/O and so on.

6.2.4 Modifiability

Modifiability encompasses two aspects [1]:

- **Maintainability**: (1) The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment. (2) The ease with which a hardware system or component can be retained in, or restored to, a state in which it can perform its required functions.

- **Flexibility**: The ease with which a system or component can be modified for use in applications or environments other than those for which is was specifically designed.

As we explained in previous chapters, our framework is composed of collaboration as a service and collaboration services, all of which are independent software services and can be replaced or updated easily. If there are more functionalities that need to be added, say authentication service or data encrypt service, just plug them into the framework and the existing system will not be affected or interrupted.

6.2.5 Usability

Usability refers to "the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component." [1]

It is a measure of how well users can take advantage of system functionalities. From the creators' view, the framework provides interfaces for them to create peer collaborations by simply configuring a participant list and a work flow model file, as well for them to easily manage and manipulate the peer collaborations that they have created. From the participants' view, the framework provides functionalities for them to quickly take part in the peer collaborations by simply registering themselves and for them to easily share documents and manage their tasks.

Chapter 7

Conclusion

This chapter will give a conclusion for the thesis. First, we summarize Chapter 3 to Chapter 6. Then we will elaborate our contributions for this thesis, and end with the future work.

7.1 Summary

In this thesis, we propose and implement a framework implemented on cloud providing a systematic approach to generate and manage web based peer collaborations, namely Peer Collaboration Service Framework, which is short for "PCSF".

We first explain the three-layer design of this framework. On the top is collaboration as a service layer, which contains service(s) to dynamically generate peer collaborations based on the specification provided by the creators and service(s) to manage the generated peer collaborations. The specification is a workflow file describing how the collaboration works, and every peer collaboration must be combined with a workflow file. In the middle is collaboration service layer, generated from collaboration as a service layer, and contains a group of predefined services, each of which provide a certain aspect of functionalities to facilitate peer collaborations, such as registration, task management, exceptions handling, resource management,

etc. At the bottom is collaboration service instance layer, generated from collaboration service layer. One peer collaboration can generate multiple collaboration instances, and all of these instances share the same workflow model but might be completed by different participants.

Then we give the implementations of the six services inside the framework. Collaboration as a service layer consists of two services: creation service, which is responsible for creating peer collaborations; instance management service, which is in charge of managing the lifecycle of generated collaboration instances. Collaboration service layer consists of four services: registration service, to register participants into collaboration instances; schedule and coordinate service, to manage the tasks and the workflow process; data transfer service, to transfer data between participants and storage center; monitor and control service, to handle exceptions.

A collaborative testing application has been generated by the framework as a case study. We have also designed several experiments to evaluate and exam the functionalities and performances, covering create a peer collaboration; create multiple peer collaborations; generate a collaboration instance; generate multiple collaboration instances from one peer collaboration; run an instance; run multiple instances. In addition, we list serval criteria to evaluate our framework and also give a short explain on how well it matches them.

7.2 Contributions

First, we have introduced a new concept of collaboration as a service, which is services that are used to dynamically generate peer collaborations based on the information specified by users. Then under the notion of this concept, we have proposed a framework called "Peer Collaboration Service Framework", which consists of collaboration as a service, as well a group of four predefined collaboration services

that will run at the back end to support the generated peer collaborations. These collaboration services are: registration service, schedule and coordinate service, monitor and control service, data transfer service, each of which is independent services and provides a specific functionality to facilitate the collaborations. Therefore our framework gives an easy way for creators to create and manage collaborations and for participants to use them.

Next, we have implemented our framework based on cloud computing technology. The whole framework is deployed in Amazon EC2 and the generated peer collaborations are also running in cloud. We have applied services offered by Amazon which include compute power and storage spaces so as to take the advantage of scalability that allows real-time provisioning of resources to meet changed requirements.

In order to aid our framework, we have tested it by generating a collaborative testing application in cloud. We have created a workflow model to simulate the working process in a testing company when they receive a task. We have also created some artificial participants to take part in this collaboration to complete it. Each participant can run in different locations. The workflow is well executed and controlled by the framework and all the documents and data that need to be delivered are well managed as well.

7.3 Future Work

Collaboration has always been a popular area for researchers. CaaS is a new concept and idea to provide a systematic way to supervise network based peer collaborations.

Currently, our framework has been deployed on Amazon EC2, but there are other cloud vendors to choose like Google App Engine. Amazon provides Infrastructure as a Service, which allows users to do the whole setup things required for their

applications. Google App Engine, on the other hand, gives a complete platform for users to develop and deploy their applications, therefore, users might not have too much fine grain control on OS and hardware. Besides, GAE only supports Java and Python so far.

To support collaboration, there are many aspects need to be considered such as user management, credential management, task management, resource management and so on. PCSF includes four collaboration services covering parts of these aspects. In the future, more services can be added into the framework to provide more features like security, which might include data encryption and decryption during transporting, user authentications, and which users are authorized with which resources.

In our implementation, all participants are humans and non-human participants are not support yet, so this could be a further extension. Further more, each participants must provide their email addresses because the system will send notifications in the form of email and this is the only way currently support. So a further work could be to make the system applying more approach like send to an URI.

Bibliography

[1] *Ieee standard glossary of software engineering terminology*, IEEE Std 610.12-1990 (1990), 1.

[2] O. Alfarraj and L. Vlacic, *Towards using web-based collaborative technologies within e-government practice: A case study of quickplace technology*, 2010 International Conference on Information Society (i-Society), 2010, pp. 605–611.

[3] R.M. Baecker, *Readings in human-computer interaction: toward the year 2000*, Morgan Kaufmann Publishers, 1995.

[4] R. Buyya, S. Pandey, and C. Vecchiola, *Cloudbus toolkit for market-oriented cloud computing*, Proceedings of the 1st International Conference on Cloud Computing (CloudCom 2009, Springer, Germany), December 2009.

[5] Rajkumar Buyya, James Broberg, and Andrzej Goscinski, *Cloud computing: Principles and paradigms*, Wiley, 2012.

[6] Peter H. Carstensen and Kjeld Schmidt, *Computer supported cooperative work: New challenges to systems design*, In K. Itoh (Ed.), Handbook of Human Factors, 1999, pp. 619–636.

[7] Workflow Management Coalition, *Xml process definition language (xpdl) 2.2 - a standard for wfmc*, http://www.xpdl.org/, 30 August 2012.

[8] Java Community, *Jsr 339: Jax-rs 2.0: The java api for restful web services*, http://jcp.org/en/jsr/detail?id=339, July 2012.

[9] B. Dahlbom and F. Ljungberg, *Mobile informatics*, Scandinavian Journal of Information Systems **10** (1998), 227–234.

[10] S. Dustdar, H. Gall, and R. Schmidt, *Web services for groupware in distributed and mobile collaboration*, Parallel, Distributed and Network-Based Processing, 2004. Proceedings. 12th Euromicro Conference on, feb. 2004, pp. 241 – 247.

[11] A. Dwivedi, R. K. Bali, A. E. James, and R. N. G. Naguib, *Workflow management systems: the healthcare technology of the future?*, 2001 Proceedings of the 23rd Annual Engineering in Medicine and Biology Society International Conference, Istanbul, Turkey, 2001, pp. 3887–3890.

[12] Amazon EC2, *Amazon elastic compute cloud (amazon ec2)*, http://aws.amazon.com/ec2, October 2012.

[13] C. Eikemeier and U. Lechner, *Introducing domain specific ad-hoc collaboration: the peer-to-peer tool iknow*, Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICEí03), 2003, pp. 107–112.

[14] P. Eklund, M. Johansson, J. Karlsson, and R. Astrom, *Bpmn and its semantics for information management in emergency care*, Computer Sciences and Convergence Information Technology, 2009. ICCIT '09. Fourth International Conference on, Nov. 2009, pp. 273–278.

[15] T. Erl, *Service-oriented architecture: Concept, technology, and design*, Hall PTR, 2005.

[16] M. Esbjrnsson and M. Ostergren, *Hocman: Supporting mobile group collaboration*, New York: ACM Press, 2002.

[17] Organization for the Advancement of Structured Information Standards (OASIS), *Web services business process execution language version 2.0 - oasis standard*, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, 11 April 2007.

[18] Apache Software Foundation, *Apache cxf, an opne source services framework*, http://cxf.apache.org/, October 2012.

[19] Q. Fu, J. Liang, H. Chen, and F. Bu, *The research on formal specification and review of workflows*, 2010 Fifth International Conference on Internet and Web Applications and Services, 2010, pp. 19–25.

[20] DeSanctis G. and Gallupe R. B, *A foundation for the study of group decision support systems*, Management Science **33** (1987), no. 5, 589.

[21] J.J. Garcia-Luna-Aceves, E.J. Craighill, and R. Lang, *An open-systems model for computer-supported collaboration*, Computer Workstations, 1988., Proceedings of the 2nd IEEE Conference on, mar 1988, pp. 40 –51.

[22] C. Gong, J. Liu, Q. Zhang, H. Chen, and Z. Gong, *The characteristics of cloud computing*, 2010 International Conference on Parallel Processing Workshops (ICPPW), 2010, pp. 275–279.

[23] Google, *Google app engine*, http://code.google.com/appengine, October 2012.

[24] M. Hauswirth, I. Podnar, and S. Decker, *On p2p collaboration infrastructures*, 2005 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise, 2005, pp. 66–71.

[25] Yonggang Huang, Chunyang Hu, Yongwang Zhao, and Dianfu Ma, *Web-based remote collaboration over medical image using web services*, Information Infrastructure Symposium, 2009. GIIS '09. Global, June 2009, pp. 1 –8.

[26] Wikipedia IBM Lotus Notes, *Ibm lotus notes*, http://en.wikipedia.org/wiki/IBM_Lotus_Notes, December 2012.

[27] Google Inc., *Google office productivity suite*, https://docs.google.com/, October 2012.

[28] JBoss, *Java business process management*, http://www.jboss.org/jbpm, August 2012.

[29] J. Lipnack and J. Stamps, *Virtual teams: People working across boundaries with technology*, New York: Wiley, 2000.

[30] F. Liu, G. Wang, L. Li, and W. Chou, *Web service for distributed communication systems*, 2006 IEEE International Conference on Service Operations and Logistics, and Informatics, Jun. 2006, pp. 1030–1035.

[31] S. Liu, B. Spencer, W. Du, and C. Chi, *Towards an open collaboration service framework*, 2011 International Conference on Collaboration Technologies and Systems (CTS), 2011, pp. 77–85.

[32] M. I. Mabrouk, *Soa fundamentals in a nutshell*, https://www.ibm.com/developerworks/webservices/tutorials/ws-soa-ibmcertified/, September 2008.

[33] Microsoft, *Windows azure*, http://www.microsoft.com/windowsazure/, October 2012.

[34] K. Nakanishi, T. Aonuma, J. Ma, and J. Lee, *Peer collaboration modes and private room services in tomscop*, 2005. Proceedings. 11th International Conference on Parallel and Distributed Systems, 2005, pp. 196–200.

[35] L. Navarro, W. Prinz, and T. Rodden, *Towards open cscw systems*, Distributed Computing Systems, 1992., Proceedings of the Third Workshop on Future Trends of, apr 1992, pp. 4 –10.

[36] NetSuite, *Cloud erp, business accounting software, crm, ecommercen - netsuite*, http://www.netsuite.com/portal/home.shtml, October 2012.

[37] Object Management Group (OMG), *Business process modeling notation (bpmn), version 2.0*, http://www.omg.org/spec/BPMN/2.0/, January 2011.

[38] Y. Osais, S. Abdala, and A. Matrawy, *A multilayer peer-to-peer framework for distributed synchronous collaboration*, Internet Computing, IEEE **10, no. 6** (Nov. 2006), 33–41.

[39] M. P. Papazoglou, *Web services: Principles and technology*, Pearson Prentice Hall, 2008.

[40] Java Community Process, *Jsr-000224 javatm api for xml-based web services 2.2 rev a*, http://jcp.org/aboutJava/communityprocess/mrel/jsr224/index4.html, December 2011.

[41] Inc. Questetra, *Bpel*, http://en.q-bpm.org/mediawiki/index.php/BPEL, October 2012.

[42] Amazon RDS, *Amazon relational database service (amazon rds)*, http://aws.amazon.com/rds/, October 2012.

[43] B.A. Reinig and D.L. Amoroso, *Measuring the effectiveness of collaboration technology*, System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on, jan. 2004, p. 1 pp.

[44] T. Rodden, *A technological framework for cscw*, CSCW: Some Fundamental Issues, IEE Colloquium on, mar 1991, pp. 7/1 –7/5.

[45] Amazon S3, *Amazon simple storage service (amazon s3)*, http://aws.amazon.com/s3/, October 2012.

[46] SalesForce, *Crm and cloud computing - salesforce.com*, http://www.salesforce.com/, October 2012.

[47] Sarker, Saonee, Damon E. Campbell, Jan Ondrus, and Joseph S. Valacich, *Mapping the need for mobile collaboration technologies: A fit perspective*, International Journal of e-Collaboration (IJeC) **6** (2010), 32–53.

[48] D. Schumm, D. Karastoyanova, F. Leymann, and J. Nitzsche, *On visualizing and modelling bpel with bpmn*, Grid and Pervasive Computing Conference, 2009. GPC '09. Workshops at the, may 2009, pp. 80–87.

[49] Amazon SES, *Amazon simple email service (amazon ses)*, http://aws.amazon.com/ses/, October 2012.

[50] G. Shroff, *Enterprise cloud computing-technology, architecture, applications 1st edtion*, Cambridge University Press, 2010.

[51] simon.m.stewart, *Page objects - selenium: Browser automation framework*, http://code.google.com/p/selenium/wiki/PageObjects, October 8 2011.

[52] Amazon SimpleDB, *Amazon simpledb*, http://aws.amazon.com/simpledb/, October 2012.

[53] Wikipedia SOAP, *Soap*, http://en.wikipedia.org/wiki/SOAP, 29 October 2012.

[54] C Steinfield, *Realizing the benefits of virtual teams*, IEEE Computer, March 2002, pp. 104–106.

[55] S. Tilley and P. Bhallamudi, *Soa migration case studies and lessons learned*, Systems Conference (SysCon), 2011 IEEE International, 2011, pp. 123–128.

[56] Wikipedia Peer to Peer, *Peer-to-peer*, http://en.wikipedia.org/wiki/Peer-to-peer, August 2011.

[57] Christos Tranoris, *Tutorial 2: Orchestrating web services and eclipse - part 1*, http://www.softwareagility.gr/index.php?q=node/22, November 2009.

[58] A. T. Velte, T. J. Velte, and R. Elsenpeter, *Cloud computing-a practical approach*, McGraw Hill, 2010.

[59] L. Wang and X. Huang, *A research on workflow technology application in publishing industry*, 2011 International Conference on Computer Science and Service System (CSSS), 2011, pp. 1968–1970.

[60] X. Xu, *From cloud computing to cloud manufacturing*, Robotics and Computer-Integrated Manufacturing **28** (2012), 75–86.

[61] Shenjun Xue and Yi Liu, *A cscw systems classification*, Computer Science and Software Engineering, 2008 International Conference on, vol. 4, dec. 2008, pp. 996 –999.

[62] J. Yang and Z. Chen, *Cloud computing research and security issues*, 2010 International Conference on Computational Intelligence and Software Engineering (CiSE 2010), 2010, pp. 1–3.

[63] X. Ying, C. Deren, and C. Min, *Research of web services workflow and its key technology based on xpdl*, 2004 IEEE International Conference on Systems, Man and Cybernetics, 2004, pp. 2137–2142.

Appendix A

CaaS Implementation Codes

A.1 Creation Service

```
/**
 * Faculty of Computer Science, University of New Brunswick
 */
package ca.unb.cs.pcsf.services.crt;

import static ca.unb.cs.pcsf.services.crt.CreationServiceConstants.*;

import java.io.*;
import java.util.*;
import javax.jws.WebService;
import javax.mail.*;

import org.apache.log4j.Logger;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.*;
import com.amazonaws.services.simpledb.*;
import com.amazonaws.services.simpleemail.*;

/**
 * @author dongdong
 * @version 0.0.1
 * @date Jul 8, 2012
 */
@WebService(endpointInterface = "ca.unb.cs.pcsf.services.crt.CreationService")
public class CreationServiceImpl implements CreationService {
  private Logger logger = Logger.getLogger(CreationServiceImpl.class.getName());
  private AmazonS3 s3;
  private AmazonSimpleDB sdb;
  private AmazonSimpleEmailService ses;
  private PropertiesCredentials credentials;
```

```
/** Constructor */
public CreationServiceImpl() {
  try {
    credentials = new PropertiesCredentials(CreationServiceImpl.class
              .getResourceAsStream(CREDENTIAL_FILE_PATH));
    s3 = new AmazonS3Client(credentials);
    sdb = new AmazonSimpleDBClient(credentials);
    ses = new AmazonSimpleEmailServiceClient(credentials);

    createDomain(DOMAIN_CREATOR); // create domain "Creator"
    createDomain(DOMAIN_COLLABORATION);// create domain "Collaboration"
    createDomain(DOMAIN_PARTICIPANT); // create domain "Participant"
  } catch (IOException e) {
    e.printStackTrace();
  }

  logger.info("Creating temporary workspace folders for pcsf...");
  File savePath = new File(SAVE_PATH);
  if (!savePath.exists() || !savePath.isDirectory()) {
    logger.info(" - creating collaboration home folder <pcsf>...");
    savePath.mkdir();
  }
}

@Override
public boolean createCollaboration(String collaborationName
              , String creatorId, File workflowFile) {
  logger.debug(LOGPRE + "createCollaboration() start" + LOGPRE);

  // create bucket if not exist
  logger.info("checking Amazon S3 bucket...");
  String bucketName = "pcsf-s3-bucket-" + collaborationName;

  boolean isBucketExist = false;
  List<Bucket> bucketList = s3.listBuckets();
  for (Bucket bucket : bucketList) {
    if (bucket.getName().equals(bucketName))
      isBucketExist = true;
  }

  if (!isBucketExist) {
    logger.info("bucket <" + bucketName
              + "> not exist, creating the bucket...");
    s3.createBucket(bucketName);
  }

  // upload file to bucket
  logger.info("uploading file into S3 bucket...");
  String key = workflowFile.getName();
  s3.putObject(new PutObjectRequest(bucketName, key, workflowFile));

  logger.info("file <" + workflowFile.getName()
              + "> has been uploaded to bucket <"
```

```
            + bucketName + "> with key <"
            + key + ">");

  // create collaboration
  String collaborationId = this.idGenerator(DOMAIN_COLLABORATION);

  List<ReplaceableItem> items = new ArrayList<ReplaceableItem>();
  ReplaceableItem item = new ReplaceableItem(collaborationId);
  item.withAttributes(
      new ReplaceableAttribute(COLLABORATION_ATTRIBUTE_NAME
              , collaborationName, true),
      new ReplaceableAttribute(COLLABORATION_ATTRIBUTE_CREATOR_ID
              , creatorId, true),
      new ReplaceableAttribute(COLLABORATION_ATTRIBUTE_CURRENT_STATE
              , "0", true),
      new ReplaceableAttribute(COLLABORATION_ATTRIBUTE_WORKFLOW_MODEL
              , workflowFile.getAbsolutePath(), true),
      new ReplaceableAttribute(COLLABORATION_ATTRIBUTE_PARTICIPANT
              , "NO DATA", true));

  items.add(item);
  logger.info("Putting collaboration <" + collaborationName
              + "> into domain...");
  sdb.batchPutAttributes(new BatchPutAttributesRequest
              (DOMAIN_COLLABORATION, items));
  logger.info("Collaboration <" + collaborationName
              + "> has been created successfully!");

  logger.debug(LOGPRE + "createCollaboration() end" + LOGPRE);
  return true;
}

@Override
public boolean deployCollaboration(String collaborationId) {
  // query collaboration from db
  String collaborationName = "";
  String creatorId = "";
  String noOfInstances = "";
  String workflowFilePath = "";

  Item findItem = new Item();
  String getCollaborationRequest = "select * from `"
              + DOMAIN_COLLABORATION + "`";
  findItem = findItem(collaborationId, getCollaborationRequest);

  if (findItem != null) {
    for (Attribute attribute : findItem.getAttributes()) {
      if (attribute.getName().equals(COLLABORATION_ATTRIBUTE_NAME))
        collaborationName = attribute.getValue();
      if (attribute.getName().equals(COLLABORATION_ATTRIBUTE_CREATOR_ID))
        creatorId = attribute.getValue();
      if (attribute.getName().equals(COLLABORATION_ATTRIBUTE_CURRENT_STATE))
        noOfInstances = attribute.getValue();
      if (attribute.getName().equals(COLLABORATION_ATTRIBUTE_WORKFLOW_MODEL))
```

```
        workflowFilePath = attribute.getValue();
    }

    logger.info("Deploying collaboration instance...");
    // generate collaboration services
    logger.info("Generating collaboration services...");
    generateServices(collaborationId, collaborationName, noOfInstances
              , workflowFilePath);

    // increase the number of instance.
    List<ReplaceableAttribute> replaceableAttributes
              = new ArrayList<ReplaceableAttribute>();
    replaceableAttributes.add(new ReplaceableAttribute
              (COLLABORATION_ATTRIBUTE_CURRENT_STATE
              , String.valueOf(Integer.parseInt(noOfInstances) + 1), true));
    sdb.putAttributes(new PutAttributesRequest(DOMAIN_COLLABORATION
              , collaborationId, replaceableAttributes));

    // send notification email to creator
    logger.info("Sending an email to creator...");

    String creatorEmail = "";
    String creatorName = "";

    String getCreatorRequest = "select * from `" + DOMAIN_CREATOR + "`";
    findItem = findItem(creatorId, getCreatorRequest);
    if (findItem != null) {
      for (Attribute attribute : findItem.getAttributes()) {
        if (attribute.getName().equals(CREATOR_ATTRIBUTE_NAME))
          creatorName = attribute.getValue();
        if (attribute.getName().equals(CREATOR_ATTRIBUTE_EMAIL))
          creatorEmail = attribute.getValue();
      }
    }
    sendCreatorNotificationMail(creatorEmail, creatorName);
  }
  return true;
}

@Override
public boolean deleteCollaboration(String collaborationId) {
  logger.debug(LOGPRE + "deleteCollaboration() start" + LOGPRE);

  // find collaboration name and work flow file path
  String collaborationName = "";
  String workflowFilePath = "";

  Item findItem = new Item();

  String getCollaborationRequest = "select * from `"
              + DOMAIN_COLLABORATION + "`";
  findItem = findItem(collaborationId, getCollaborationRequest);
  if (findItem != null) {
    for (Attribute attribute : findItem.getAttributes()) {
```

```
      if (attribute.getName().equals(COLLABORATION_ATTRIBUTE_NAME))
        collaborationName = attribute.getValue();
      if (attribute.getName().equals(COLLABORATION_ATTRIBUTE_WORKFLOW_MODEL))
        workflowFilePath = attribute.getValue();
    }
  }

  logger.info("Deleting collaboration <" + collaborationName + ">...");
  File collaborationFolder = new File(workflowFilePath).getParentFile();

  // delete collaboration services
  logger.info("Deleting collaboration services...");
  File[] services = new File(SERVICES_DEPLOY_LOCATION).listFiles();
  if (null != services) {
    for (File service : services) {
      if (service.isFile()
            && service.getName().startsWith(collaborationName)) {
        logger.info(" - deleting service <"
            + service.getName().substring(0,service.getName().length()-4)
            + ">");
        delFile(service);
      }
      if (service.isDirectory()
            && service.getName().startsWith(collaborationName)) {
        delDirectory(service);
      }
    }
  }

  // delete work flow file
  logger.info("Deleting workflow file...");
  delDirectory(collaborationFolder);

  // delete collaboration from db
  if (isCollaborationExist(collaborationName)) {
    sdb.deleteAttributes(new DeleteAttributesRequest(DOMAIN_COLLABORATION
            , collaborationId));
    logger.debug(LOGPRE + "deleteCollaboration() end" + LOGPRE);
  }

  // delete bucket from s3
  logger.info("Deleting bucket from s3...");
  String bucketName = "pcsf-s3-bucket-" + collaborationName;
  s3.deleteObject(bucketName, collaborationName + ".zip");
  s3.deleteObject(bucketName, new File(workflowFilePath).getName());
  s3.deleteBucket(bucketName);

  logger.info("Delete Done!");
  logger.info("Collaboration <" + collaborationName
            + "> has been deleted successfully!");
  logger.debug(LOGPRE + "deleteCollaboration() end" + LOGPRE);
  return true;
}
```

```
@Override
public boolean deleteInstance(String collaborationId) {
  logger.debug(LOGPRE + "deleteInstance() start" + LOGPRE);

  // find collaboration name and work flow file path
  String collaborationName = "";

  Item findItem = new Item();

  String getCollaborationRequest = "select * from `"
              + DOMAIN_COLLABORATION + "`";
  findItem = findItem(collaborationId, getCollaborationRequest);
  if (findItem != null) {
    for (Attribute attribute : findItem.getAttributes()) {
      if (attribute.getName().equals(COLLABORATION_ATTRIBUTE_NAME))
        collaborationName = attribute.getValue();
    }
  }

  logger.info("Deleting collaboration <" + collaborationName + ">...");

  // delete collaboration services
  logger.info("Deleting collaboration services...");
  File[] services = new File(SERVICES_DEPLOY_LOCATION).listFiles();
  if (null != services) {
    for (File service : services) {
      if (service.isFile()
              && service.getName().startsWith(collaborationName)) {
        logger.info(" - deleting service <"
              + service.getName().substring(0,service.getName().length()-4)
              + ">");
        delFile(service);
      }
      if (service.isDirectory()
              && service.getName().startsWith(collaborationName)) {
        delDirectory(service);
      }
    }
  }

  // delete collaboration from db
  if (isCollaborationExist(collaborationName)) {
    logger.info("Deleting participants...");
    String[] participantNames = null;

    String selectRequest = "select * from `" + DOMAIN_COLLABORATION + "`";
    findItem = findItem(collaborationId, selectRequest);

    if (findItem != null) {
      for (Attribute attribute : findItem.getAttributes()) {
        if(attribute.getName().equals(COLLABORATION_ATTRIBUTE_PARTICIPANT))
          participantNames = attribute.getValue().split(":");
      }
    }
```

```
    for (String p : participantNames) {
      if (p != null && !p.equals("NO DATA")) {
        logger.info(" - deleting participant <" + p + ">...");
        sdb.deleteAttributes(new DeleteAttributesRequest(DOMAIN_PARTICIPANT
              , p));
      }
    }
    sdb.deleteAttributes(new DeleteAttributesRequest(DOMAIN_COLLABORATION
              , collaborationId));
  }

  String collaborationServiceName = collaborationName.split("-")[0];
  String selectExp = "select * from `" + DOMAIN_COLLABORATION
              + "` where " + COLLABORATION_ATTRIBUTE_NAME
              + "='" + collaborationServiceName + "'";
  Item collaborationServiceItem = sdb.select(new SelectRequest(selectExp))
              .getItems().get(0);
  String instanceCount = "";
  for (Attribute attribute : collaborationServiceItem.getAttributes()) {
    if (attribute.getName().equals(COLLABORATION_ATTRIBUTE_CURRENT_STATE))
      instanceCount = attribute.getValue();
  }

  // change the number of instance.
  List<ReplaceableAttribute> replaceableAttributes
              = new ArrayList<ReplaceableAttribute>();
  replaceableAttributes.add(new ReplaceableAttribute
              (COLLABORATION_ATTRIBUTE_CURRENT_STATE
              , String.valueOf(Integer.parseInt(instanceCount)-1), true));
  sdb.putAttributes(new PutAttributesRequest(DOMAIN_COLLABORATION
              , collaborationServiceItem.getName()
              , replaceableAttributes));

  logger.info("Delete Done!");
  logger.info("Instance <" + collaborationName
              + "> has been deleted successfully!");
  logger.debug(LOGPRE + "deleteInstance() end" + LOGPRE);
  return true;
}

/**
 * Generate collaboration services.
 *
 * @param collaborationName
 */
private void generateServices(String collaborationId
      , String collaborationName, String noOfInstance, String workflowFile) {
  logger.debug(LOGPRE + "generateServices() start" + LOGPRE);

  // add a new instance into simple db
  String instanceIndex = String.valueOf(Integer.parseInt(noOfInstance)+1);
  List<ReplaceableItem> items = new ArrayList<ReplaceableItem>();
  ReplaceableItem item = new ReplaceableItem(collaborationId + "-"
```

```
            + instanceIndex);
item.withAttributes(
    new ReplaceableAttribute(COLLABORATION_ATTRIBUTE_NAME,
             collaborationName + "-" + instanceIndex, true),
    new ReplaceableAttribute(COLLABORATION_ATTRIBUTE_CURRENT_STATE
            , COLLABORATION_STATE_NEW_CREATED, true),
    new ReplaceableAttribute(COLLABORATION_ATTRIBUTE_WORKFLOW_MODEL
            , workflowFile, true),
    new ReplaceableAttribute(COLLABORATION_ATTRIBUTE_PARTICIPANT
            , "NO DATA", true));

items.add(item);
sdb.batchPutAttributes(new BatchPutAttributesRequest
            (DOMAIN_COLLABORATION, items));

String serviceLocation = "";
if (SERVICES_DEPLOY_LOCATION.endsWith(File.separator)) {
  serviceLocation = SERVICES_DEPLOY_LOCATION + SERVICES_SOURCE_LOCATION;
} else {
  serviceLocation = SERVICES_DEPLOY_LOCATION
            + File.separator + SERVICES_SOURCE_LOCATION;
}
// get all files and folders in source folder
File[] files = (new File(serviceLocation)).listFiles();

if (files != null && files.length > 0) {
  for (File file : files) {
    if (file.isFile() && !(file.getName().startsWith("."))
            && !(file.getName().contains("pcsfWeb"))) {
      logger.info(" - generating collaboration service <"
            + file.getName() + ">...");
      try {
        String deployedName = SERVICES_DEPLOY_LOCATION + File.separator
            + collaborationName + "-" + instanceIndex + "-" + file.getName();
        copyFile(file, new File(deployedName));
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
    if (file.isDirectory() && !(file.getName().startsWith("."))
            && !(file.getName().contains("pcsfWeb"))) {
      logger.info(" - generating collaboration service <"
            + file.getName() + ">...");
      String sourceDir = serviceLocation + File.separator + file.getName();
      String targetDir = SERVICES_DEPLOY_LOCATION + File.separator
            + collaborationName + "-" + instanceIndex + "-" + file.getName();
      try {
        copyDirectory(sourceDir, targetDir);
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
  }
}
```

```
  logger.debug(LOGPRE + "generateServices() end" + LOGPRE);
}

/**
 * Copy file
 *
 * @param sourceFile
 * @param targetFile
 * @throws IOException
 */
private void copyFile(File sourceFile, File targetFile)
              throws IOException {
  logger.debug(LOGPRE + "copyFile() start" + LOGPRE);
  logger.debug("generating file: " + sourceFile);

  FileInputStream input = new FileInputStream(sourceFile);
  BufferedInputStream inBuff = new BufferedInputStream(input);

  FileOutputStream output = new FileOutputStream(targetFile);
  BufferedOutputStream outBuff = new BufferedOutputStream(output);

  byte[] b = new byte[1024 * 5];
  int len;
  while ((len = inBuff.read(b)) != -1) {
    outBuff.write(b, 0, len);
  }
  outBuff.flush();

  inBuff.close();
  outBuff.close();
  output.close();
  input.close();

  logger.debug(LOGPRE + "copyFile() end" + LOGPRE);
}

/**
 * Copy directory
 *
 * @param sourceDir
 * @param targetDir
 * @throws IOException
 */
private void copyDirectory(String sourceDir, String targetDir)
              throws IOException {
  logger.debug(LOGPRE + "copyDirectory() start" + LOGPRE);
  logger.debug("generating folder " + targetDir);

  (new File(targetDir)).mkdirs();
  File[] file = (new File(sourceDir)).listFiles();
  for (int i = 0; i < file.length; i++) {
    if (file[i].isFile()) {
      File sourceFile = file[i];
      File targetFile = new File(new File(targetDir).getAbsolutePath()
```

```
              + File.separator + file[i].getName());
      copyFile(sourceFile, targetFile);
    }
    if (file[i].isDirectory()) {
      String dir1 = sourceDir + File.separator + file[i].getName();
      String dir2 = targetDir + File.separator + file[i].getName();
      copyDirectory(dir1, dir2);
    }
  }

  logger.debug(LOGPRE + "copyDirectory() start" + LOGPRE);
}

/**
 * Delete file
 *
 * @param file
 */
private void delFile(File file) {
  logger.debug(LOGPRE + "delFile() start" + LOGPRE);

  if (file.exists() && file.isFile())
    file.delete();

  logger.debug(LOGPRE + "delFile() start" + LOGPRE);
}

/**
 * Delete directory
 *
 * @param directory
 */
private void delDirectory(File directory) {
  logger.debug(LOGPRE + "delDirectory() start" + LOGPRE);

  if (directory.exists()) {
    File[] children = directory.listFiles();
    if (children == null || children.length <= 0)
      directory.delete();
    else {
      for (File child : children) {
        if (child.exists() && child.isFile())
          child.delete();
        if (child.exists() && child.isDirectory())
          delDirectory(child);
      }
      directory.delete();
    }
  }

  logger.debug(LOGPRE + "delDirectory() end" + LOGPRE);
}
```

```
/**
 * Send a notification to creator notifying new collaboration creation.
 *
 * @param creatorEmail
 * @param creatorName
 */
private void sendCreatorNotificationMail(String creatorEmail
              , String creatorName) {
  logger.debug(LOGPRE + "sendCreatorNotificationMail() start" + LOGPRE);

  this.verifyEmailAddress(ses);
  Properties props = new Properties();
  props.setProperty("mail.transport.protocol", "aws");
  props.setProperty("mail.aws.user", credentials.getAWSAccessKeyId());
  props.setProperty("mail.aws.password", credentials.getAWSSecretKey());

  Session session = Session.getInstance(props);
  try {
    // Create a new Message
    Message msg = new MimeMessage(session);
    msg.setFrom(new InternetAddress(MAIL_FROM));
    msg.addRecipient(Message.RecipientType.TO,
              new InternetAddress(creatorEmail));
    msg.setSubject(MAIL_SUBJECT);
    msg.setText("Dear " + creatorName + ",\n\n"
              + MAIL_COMMON_CONTENT_FOR_CREATOR
              + LINK_CREATOR);
    msg.saveChanges();

    // Reuse one Transport object for sending all your messages
    // for better performance
    Transport t = new AWSJavaMailTransport(session, null);
    t.connect();
    t.sendMessage(msg, null);
    logger.info("one mail sent to creator notifying the new"
              + "created collaboration!");
    t.close();
  } catch (AddressException e) {
    e.printStackTrace();
    logger.info("Caught an AddressException, which means one or more of your"
        + " addresses are improperly formatted.");
  } catch (MessagingException e) {
    e.printStackTrace();
    logger.info("Caught a MessagingException, which means that there was a "
      + "problem sending your message to Amazon's E-mail Service check the "
      + "stack trace for more information.");
  }

  logger.debug(LOGPRE + "sendCreatorNotificationMail() end" + LOGPRE);
}

/**
 * Sends a request to Amazon Simple Email Service to verify the specified
 * email address. This triggers a verification email, which will contain a link
```

```
 * that you can click on to complete the verification process.
 *
 * @param ses
 *          The Amazon Simple Email Service client to use when making
 *          requests to Amazon SES.
 * @param address
 *          The email address to verify.
 */
private void verifyEmailAddress(AmazonSimpleEmailService ses) {
  logger.debug(LOGPRE + "verifyEmailAddress() start" + LOGPRE);

  ListVerifiedEmailAddressesResult verifiedEmails
              = ses.listVerifiedEmailAddresses();
  if (verifiedEmails.getVerifiedEmailAddresses().contains(MAIL_FROM)) {
    logger.debug(LOGPRE + "verifyEmailAddress() end" + LOGPRE);
    return;
  }

  ses.verifyEmailAddress(new VerifyEmailAddressRequest()
              .withEmailAddress(MAIL_FROM));
  logger.info("Please check the email address " + MAIL_FROM + " to verify it");
  logger.debug(LOGPRE + "verifyEmailAddress() end" + LOGPRE);
}

/**
 * Generate a random id for new user or collaboration.
 *
 * @param domainName
 * @return random id
 */
private String idGenerator(String domainName) {
  logger.debug(LOGPRE + "idGenerator() start" + LOGPRE);

  Random random = new Random(System.currentTimeMillis());
  String id = String.valueOf(((int) (random.nextDouble() * MAX)) + 1);
  boolean isDuplicated = true;

  String selectExp = "";
  if (domainName.equals(DOMAIN_PARTICIPANT)) {
    logger.info("generating id for participant ...");
    selectExp = "select * from `" + DOMAIN_PARTICIPANT + "`";
  }
  if (domainName.equals(DOMAIN_COLLABORATION)) {
    logger.info("generating id for collaboration ...");
    selectExp = "select * from `" + DOMAIN_COLLABORATION + "`";
  }
  if (domainName.equals(DOMAIN_CREATOR)) {
    logger.info("generating id for creator ...");
    selectExp = "select * from `" + DOMAIN_CREATOR + "`";
  }

  List<String> ids = new ArrayList<String>();
  List<Item> items = sdb.select(new SelectRequest(selectExp)).getItems();
  if (!items.isEmpty()) {
```

```
    for (Item item : items) {
      ids.add(item.getName());
    }
  }
  while (isDuplicated) {
    isDuplicated = false;
    for (String s : ids) {
      if (s.equals(id)) {
        isDuplicated = true;
        id = String.valueOf(((int) (random.nextDouble() * MAX)) + 1);
        break;
      }
    }
  }

  logger.debug(LOGPRE + "idGenerator() end" + LOGPRE);
  return id;
}

/**
 * create a domain
 *
 * @param domainName
 * @return if the domain been created successfully
 */
private void createDomain(String domainName) {
  logger.debug(LOGPRE + "createDomain() start" + LOGPRE);

  boolean isDomainExist = false;
  for (String dn : sdb.listDomains().getDomainNames()) {
    if (dn.equals(domainName)) {
      logger.debug("Domain <" + domainName + "> exists");
      isDomainExist = true;
    }
  }

  if (!isDomainExist) {
    logger.info("Creating domain called <" + domainName + ">...");
    sdb.createDomain(new CreateDomainRequest(domainName));
    logger.info("Domain <" + domainName + "> has been created");
  }

  logger.debug(LOGPRE + "createDomain() end" + LOGPRE);
}

/**
 * find a record by id from db
 *
 * @param id
 * @param selectRequest
 * @return the find record
 */
private Item findItem(String id, String selectReq) {
  logger.debug(LOGPRE + "findItem() start" + LOGPRE);
```

```
    logger.debug("Selecting: " + selectRequest);
    logger.debug("Getting data...");
    List<Item> items = sdb.select(new SelectRequest(selectReq)).getItems();
    Item findItem = new Item();
    if (!items.isEmpty()) {
      for (Item item : items) {
        if (item.getName().equals(id)) {
          findItem = item;
          break;
        }
      }
    }

    logger.debug(LOGPRE + "findItem() end" + LOGPRE);
    return findItem;
  }

  /**
   * check if collaboration exist
   *
   * @param collaborationName
   * @return
   */
  private boolean isCollaborationExist(String collaborationName) {
    logger.debug(LOGPRE + "isCollaborationExist() start" + LOGPRE);

    String selectReq = "select * from `" + DOMAIN_COLLABORATION
                + "` where " + COLLABORATION_ATTRIBUTE_NAME
                + " = '" + collaborationName + "'";
    List<Item> items = sdb.select(new SelectRequest(selectReq)).getItems();
    if (items.isEmpty()) {
      logger.debug("Collaboration <" + collaborationName + "> doesn't exist");
      logger.debug(LOGPRE + "isCollaborationExist() end" + LOGPRE);
      return false;
    } else {
      logger.debug("Collaboration <" + collaborationName + "> exists");
      logger.debug(LOGPRE + "isCollaborationExist() end" + LOGPRE);
      return true;
    }
  }
}
```

A.2 Instance Management Service

```
/**
 * Faculty of Computer Science, University of New Brunswick
 */
package ca.unb.cs.pcsf.services.img;
```

```
import static ca.unb.cs.pcsf.services.img.InstanceManagementServiceConstants.*;

import java.io.IOException;
import java.util.*;
import javax.jws.WebService;
import org.apache.cxf.*;
import org.apache.log4j.Logger;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.simpledb.*;

/**
 * @author ddong
 * @version 1.0
 * @createDate Dec. 21, 2012
 * @email dong.dong@unb.ca
 */
@WebService(endpointInterface
                = "ca.unb.cs.pcsf.services.img.InstanceManagementService")
public class InstanceManagementServiceImpl
                implements InstanceManagementService {
  private Logger logger
        = Logger.getLogger(InstanceManagementServiceImpl.class.getName());

  private AmazonSimpleDB sdb;
  private PropertiesCredentials credentials;

  public InstanceManagementServiceImpl() {
    // get credentials
    try {
      credentials
        = new PropertiesCredentials(InstanceManagementServiceImpl.class
                .getResourceAsStream(CREDENTIAL_FILE_PATH));
      sdb = new AmazonSimpleDBClient(credentials);
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  @Override
  public void runInstance(String collaborationId) {
    // get collaboration
    String collaborationName = "";
    String collaborationState = "";
    String processDeploymentId = "";

    String request = "select * from `" + DOMAIN_COLLABORATION + "`";
    List<Item> items = sdb.select(new SelectRequest(request)).getItems();
    Item findItem = new Item();

    if (!items.isEmpty()) {
      for (Item item : items) {
        if (item.getName().equals(collaborationId)) {
          findItem = item;
          break;
```

```
      }
    }
  }

  if (findItem != null) {
    for (Attribute attribute : findItem.getAttributes()) {
      if (attribute.getName().equals(COLLABORATION_ATTRIBUTE_NAME))
        collaborationName = attribute.getValue();
      if (attribute.getName().equals(COLLABORATION_ATTRIBUTE_CURRENT_STATE))
        collaborationState = attribute.getValue();
    }
  }

  if (!collaborationState.equals(COLLABORATION_STATE_STOP)) {
    logger.info("Ready to run collaboration <" + collaborationName + ">...");

    String url = WSURL_PRE + collaborationName
      + "-scheduleNCoordinateService/ScheduleNCoordinateService?wsdl";
    String method = "runCollaboration";
    callService(url, method, collaborationId, processDeploymentId);
  }

  // change collaboration state
  List<ReplaceableItem> changeStateitems = new ArrayList<ReplaceableItem>();
  ReplaceableItem changeStateItem = new ReplaceableItem(collaborationId);
  changeStateItem.withAttributes(new ReplaceableAttribute
      (COLLABORATION_ATTRIBUTE_CURRENT_STATE
      , COLLABORATION_STATE_RUNNING, true));
  changeStateitems.add(changeStateItem);
  sdb.batchPutAttributes(new BatchPutAttributesRequest
      (DOMAIN_COLLABORATION, changeStateitems));
}

@Override
public void stopInstance(String collaborationId) {
  // change collaboration state
  List<ReplaceableItem> changeStateitems = new ArrayList<ReplaceableItem>();
  ReplaceableItem changeStateItem = new ReplaceableItem(collaborationId);
  changeStateItem.withAttributes(new ReplaceableAttribute
      (COLLABORATION_ATTRIBUTE_CURRENT_STATE
      , COLLABORATION_STATE_STOP, true));
  changeStateitems.add(changeStateItem);
  sdb.batchPutAttributes(new BatchPutAttributesRequest
      (DOMAIN_COLLABORATION, changeStateitems));
}

/**
 * A general method used to call web service.
 *
 * @param wsUrl
 *          web service url
 * @param method
 *          method name
 * @param arg
```

```
     *          method parameters
     * @return results
     */
    private Object[] callService(String wsUrl, String method, Object... arg) {
      logger.debug(LOGPRE + "callService() start" + LOGPRE);

      Object[] resutls = null;
      JaxWsDynamicClientFactory dcf = JaxWsDynamicClientFactory.newInstance();
      Client client = dcf.createClient(wsUrl);
      try {
        resutls = client.invoke(method, arg);
      } catch (Exception e) {
        e.printStackTrace();
      }

      logger.debug(LOGPRE + "callService() end" + LOGPRE);
      return resutls;
    }
}
```

Appendix B

CS Implementation Codes

B.1 Registration Service

```
/**
 * Faculty of Computer Science, University of New Brunswick
 */
package ca.unb.cs.pcsf.services.reg;

import static ca.unb.cs.pcsf.services.reg.RegistrationServiceConstants.*;
import java.io.IOException;
import java.util.*;
import javax.jws.WebService;
import javax.mail.*;
import org.apache.cxf.*;
import org.apache.log4j.Logger;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.simpledb.*;
import com.amazonaws.services.simpleemail.*;
/**
 * @author ddong
 * @version 0.1
 * @createDate Jul 10, 2012
 * @email dong.dong@unb.ca
 */
@WebService(endpointInterface="ca.unb.cs.pcsf.services.reg.RegistrationService")
public class RegistrationServiceImpl implements RegistrationService {
  private Logger logger = Logger.getLogger(RegistrationServiceImpl.class
        .getName());

  private AmazonSimpleDB sdb;
  private AmazonSimpleEmailService ses;
  private PropertiesCredentials credentials;
```

```
public RegistrationServiceImpl() {
  // get credentials
  try {
    credentials = new PropertiesCredentials(RegistrationServiceImpl.class
      .getResourceAsStream(CREDENTIAL_FILE_PATH));
    sdb = new AmazonSimpleDBClient(credentials);
    ses = new AmazonSimpleEmailServiceClient(credentials);
  } catch (IOException e) {
    e.printStackTrace();
  }
}

@Override
public void setAsReg(String participantId) {
  logger.debug(LOGPRE + "setAsReg() start" + LOGPRE);

  // get participant
  String isReg = "";
  String participantName = null;
  String collaborationId = null;

  String getParticipantRequest = "select * from `" + DOMAIN_PARTICIPANT + "`";
  List<Item> items = sdb.select(new SelectRequest(getParticipantRequest))
      .getItems();
  Item findItem = new Item();

  if (!items.isEmpty()) {
    for (Item item : items) {
      if (item.getName().equals(participantId)) {
        findItem = item;
        break;
      }
    }
  }

  if (findItem != null) {
    for (Attribute attribute : findItem.getAttributes()) {
      if (attribute.getName().equals(PARTICIPANT_ATTRIBUTE_IS_REG)) {
        isReg = attribute.getValue();
      }
      if (attribute.getName().equals(PARTICIPANT_ATTRIBUTE_COLLABORATION_ID)){
        if (null == collaborationId)
          collaborationId = attribute.getValue();
      }
      if (attribute.getName().equals(PARTICIPANT_ATTRIBUTE_NAME)) {
        participantName = attribute.getValue();
      }
    }
  }

  // set participant as registered if not.
  if (isReg.equals(PARTICIPANT_IS_REG_NO)) {
    logger.info("registering participant <" + participantName + ">");
    List<ReplaceableAttribute> replaceableAttributes
```

```
  = new ArrayList<ReplaceableAttribute>();
replaceableAttributes.add(new ReplaceableAttribute
  (PARTICIPANT_ATTRIBUTE_IS_REG, PARTICIPANT_IS_REG_YES, true));

sdb.putAttributes(new PutAttributesRequest(DOMAIN_PARTICIPANT
  , participantId, replaceableAttributes));
try {
  Thread.sleep(1000);
} catch (InterruptedException e) {
  e.printStackTrace();
}
logger.info("done registeration!");

// get collaboration
String collaborationName = null;
String participantNames = "";
String creatorId = null;

String getCollaborationRequest = "select * from `"
  + DOMAIN_COLLABORATION + "`";
items = sdb.select(new SelectRequest(getCollaborationRequest)).getItems();

if (!items.isEmpty()) {
  for (Item item : items) {
    if (item.getName().equals(collaborationId)) {
      findItem = item;
      break;
    }
  }
}

if (findItem != null) {
  for (Attribute attribute : findItem.getAttributes()) {
    if (attribute.getName().equals(COLLABORATION_ATTRIBUTE_NAME)) {
      if (collaborationName == null)
        collaborationName = attribute.getValue();
    }
    if (attribute.getName().equals(COLLABORATION_ATTRIBUTE_CREATOR_ID)) {
      if (creatorId == null)
        creatorId = attribute.getValue();
    }
    if (attribute.getName().equals(COLLABORATION_ATTRIBUTE_PARTICIPANT))
      participantNames = attribute.getValue();
  }
}

// check if all participants have done registration.
boolean isAllReg = true;
String[] participants = participantNames.split(":");
for (String participant : participants) {
  String selectRequest = "select * from `" + DOMAIN_PARTICIPANT + "`";
  items = sdb.select(new SelectRequest(selectRequest)).getItems();

  if (!items.isEmpty()) {
```

```
    for (Item item : items) {
      if (item.getName().equals(participant)) {
        findItem = item;
        break;
      }
    }
  }

  if (findItem != null) {
    for (Attribute attribute : findItem.getAttributes()) {
      if (attribute.getName().equals(PARTICIPANT_ATTRIBUTE_IS_REG)) {
        if (!attribute.getValue().equals(PARTICIPANT_IS_REG_YES)) {
          isAllReg = false;
          break;
        }
      }
    }
  }
}

if (isAllReg) {
  logger.info("all participants have done registration");

  // get creator
  String creatorEmail = null;
  String creatorName = null;
  String getCreatorRequest = "select * from `" + DOMAIN_CREATOR + "`";
  items = sdb.select(new SelectRequest(getCreatorRequest)).getItems();

  if (!items.isEmpty()) {
    for (Item item : items) {
      if (item.getName().equals(creatorId)) {
        findItem = item;
        break;
      }
    }
  }

  if (findItem != null) {
    for (Attribute attribute : findItem.getAttributes()) {
      if (attribute.getName().equals(CREATOR_ATTRIBUTE_NAME)) {
        if (creatorName == null)
          creatorName = attribute.getValue();
      }
      if (attribute.getName().equals(CREATOR_ATTRIBUTE_EMAIL)) {
        if (creatorEmail == null)
          creatorEmail = attribute.getValue();
      }
    }
  }

  // deploy a process.
  logger.info("preparing to start the process...");
  String url = WSURL_PRE + collaborationName
```

```
      + "-scheduleNCoordinateService/ScheduleNCoordinateService?wsdl";
      String method = "deployProcess";
      callService(url, method, collaborationId);

      // set collaboration instance state to be deployed
      List<ReplaceableItem> changeStateitems
      = new ArrayList<ReplaceableItem>();
      ReplaceableItem changeStateItem = new ReplaceableItem(collaborationId);
      changeStateItem.withAttributes(new ReplaceableAttribute
      (COLLABORATION_ATTRIBUTE_CURRENT_STATE,
          COLLABORATION_STATE_DEPLOYED, true));
      changeStateitems.add(changeStateItem);
      sdb.batchPutAttributes(new BatchPutAttributesRequest
      (DOMAIN_COLLABORATION, changeStateitems));

      logger.info("sending an email to notify creator...");
      sendNotificationMail(creatorEmail, creatorName);
    }
  }
  logger.debug(LOGPRE + "setAsReg() end" + LOGPRE);
}

/**
 * A general method used to call web service.
 *
 * @param wsUrl
 *          web service url
 * @param method
 *          method name
 * @param arg
 *          method parameters
 * @return results
 */
private Object[] callService(String wsUrl, String method, Object... arg) {
  logger.debug(LOGPRE + "callService() start" + LOGPRE);

  Object[] resutls = null;
  JaxWsDynamicClientFactory dcf = JaxWsDynamicClientFactory.newInstance();
  Client client = dcf.createClient(wsUrl);
  try {
    resutls = client.invoke(method, arg);
  } catch (Exception e) {
    e.printStackTrace();
  }

  logger.debug(LOGPRE + "callService() end" + LOGPRE);
  return resutls;
}

/**
 * Send a notification to creator when all participants have done
 * the registration.
 *
 * @param role
```

```
 * @param link
 */
private void sendNotificationMail(String creatorEmail, String creatorName) {
  logger.debug(LOGPRE + " sendNotificationMail() start " + LOGPRE);

  this.verifyEmailAddress(ses);
  Properties props = new Properties();
  props.setProperty("mail.transport.protocol", "aws");
  props.setProperty("mail.aws.user", credentials.getAWSAccessKeyId());
  props.setProperty("mail.aws.password", credentials.getAWSSecretKey());
  Session session = Session.getInstance(props);
  try {
    // Create a new Message
    Message msg = new MimeMessage(session);
    msg.setFrom(new InternetAddress(MAIL_FROM));
    msg.addRecipient(Message.RecipientType.TO,
               new InternetAddress(creatorEmail));
    msg.setSubject(MAIL_SUBJECT);
    msg.setText("Dear " + creatorName + ",\n\n"
              + MAIL_COMMON_CONTENT_FOR_CREATOR + LINK_CREATOR);
    msg.saveChanges();

    // Reuse one Transport object for sending all your messages
    // for better performance
    Transport t = new AWSJavaMailTransport(session, null);
    t.connect();
    t.sendMessage(msg, null);
    logger.info("one mail sent to creator!");
    t.close();
  } catch (AddressException e) {
    e.printStackTrace();
    logger.info("Caught an AddressException, which means one or more of your"
        + " addresses are improperly formatted.");
  } catch (MessagingException e) {
    e.printStackTrace();
    logger.info("Caught a MessagingException, which means that there was a "
      + "problem sending your message to Amazon's E-mail Service check the "
      + "stack trace for more information.");
  }
  logger.debug(LOGPRE + " sendNotificationMail() end " + LOGPRE);
}

/**
 * Sends a request to Amazon Simple Email Service to verify the specified
 * email address. This triggers a verification email, which will contain
 * a link that you can click on to complete the verification process.
 *
 * @param ses
 *          The Amazon Simple Email Service client to use when
 *          making requests to Amazon SES.
 * @param address
 *          The email address to verify.
 */
private void verifyEmailAddress(AmazonSimpleEmailService ses) {
```

```
    logger.debug(LOGPRE + "verifyEmailAddress() start" + LOGPRE);

    ListVerifiedEmailAddressesResult verifiedEmails
                = ses.listVerifiedEmailAddresses();
    if (verifiedEmails.getVerifiedEmailAddresses().contains(MAIL_FROM)) {
      logger.debug(LOGPRE + "verifyEmailAddress() end" + LOGPRE);
      return;
    }
    ses.verifyEmailAddress(new VerifyEmailAddressRequest()
                .withEmailAddress(MAIL_FROM));
    logger.info("Please check the email address "+MAIL_FROM+" to verify it");
    logger.debug(LOGPRE + "verifyEmailAddress() end" + LOGPRE);
  }
}
```

B.2 Schedule and Coordination Service

```
/**
 * Faculty of Computer Science, University of New Brunswick
 */
package ca.unb.cs.pcsf.services.snc;

import static ca.unb.cs.pcsf.services.snc.ScheduleNCoordinateServiceConstants.*;
import java.io.*;
import java.util.*;
import javax.jws.WebService;
import org.apache.log4j.Logger;
import org.jbpm.api.*;
import ca.unb.cs.pcsf.services.snc.engine.*;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.simpledb.*;

/**
 * @author dongdong
 * @version 1.0
 * @createDate Aug 8, 2012
 * @email dong.dong@unb.ca
 */
@WebService(endpointInterface
        = "ca.unb.cs.pcsf.services.snc.ScheduleNCoordinateService")
public class ScheduleNCoordinateServiceImpl
        implements ScheduleNCoordinateService {
  private Logger logger = Logger.getLogger(
                ScheduleNCoordinateServiceImpl.class.getName());
  private AmazonSimpleDB sdb;
  private Engine engine;

  /** Constructor */
  public ScheduleNCoordinateServiceImpl() {
    try {
```

```
    sdb = new AmazonSimpleDBClient(new PropertiesCredentials(
        ScheduleNCoordinateServiceImpl.class
      .getResourceAsStream(CREDENTIAL_FILE_PATH)));
  } catch (IOException e) {
    e.printStackTrace();
  }
}

@Override
public void deployProcess(String collaborationId) {
  logger.debug(LOGPRE + "deployProcess() start" + LOGPRE);

  getEngine(collaborationId);
  engine.deployProcess(collaborationId);

  logger.debug(LOGPRE + "deployProcess() end" + LOGPRE);
}

@Override
public void runCollaboration(String collaborationId, String processDefId) {
  logger.debug(LOGPRE + "runCollaboration() start" + LOGPRE);

  getEngine(collaborationId);
  engine.startProcess(collaborationId, processDefId);

  logger.debug(LOGPRE + "runCollaboration() end" + LOGPRE);
}

@Override
public String[] getCurrentTask(String collaborationId) {
  logger.debug(LOGPRE + "getCurrentTask() start" + LOGPRE);
  getEngine(collaborationId);
  List<Task> taskList = engine.getCurrentTask();

  List<String> resultList = new ArrayList<String>();
  for (Task task : taskList) {
    String taskInfo = "";
    taskInfo += task.getId() + ",";
    taskInfo += task.getName() + ",";
    taskInfo += task.getAssignee() + ",";
    taskInfo += task.getCreateTime();

    resultList.add(taskInfo);
  }

  String[] strings = new String[resultList.size()];
  for (int i = 0; i < strings.length; i++) {
    strings[i] = resultList.get(i);
  }

  logger.debug(LOGPRE + "getCurrentTask() end" + LOGPRE);
  return strings;
}
```

```
@Override
public String[] getTask(String username, String collaborationId) {
  logger.debug(LOGPRE + "getTask() start" + LOGPRE);
  getEngine(collaborationId);
  List<Task> taskList = engine.getTasks(username);

  List<String> taskStrings = new ArrayList<String>();
  for (Task task : taskList) {
    String taskInfo = "";
    taskInfo += task.getId() + ",";
    taskInfo += task.getName() + ",";
    taskInfo += task.getCreateTime();

    taskStrings.add(taskInfo);
  }

  String[] strings = new String[taskStrings.size()];
  for (int i = 0; i < strings.length; i++) {
    strings[i] = taskStrings.get(i);
  }

  logger.debug(LOGPRE + "getTask() end" + LOGPRE);
  return strings;
}

@Override
public void submitTask(String taskId, String collaborationId) {
  logger.debug(LOGPRE + "submitTask() start" + LOGPRE);

  getEngine(collaborationId);
  engine.submitTask(taskId);

  logger.debug(LOGPRE + "submitTask() end" + LOGPRE);
}

/**
 * Check if the engine is initialized or not
 *
 * @param collaborationId
 */
private void getEngine(String collaborationId) {
  if (this.engine == null) {
    // String bucket = "";
    // String key = "";
    String workflowModel = "";

    String getReq = "select * from `"
              + DOMAIN_COLLABORATION + "`";
    List<Item> items = sdb.select(new SelectRequest(getReq)).getItems();
    Item findItem = new Item();

    if (!items.isEmpty()) {
      for (Item item : items) {
```

```
          if (item.getName().equals(collaborationId)) {
            findItem = item;
            break;
          }
        }
      }

      if (findItem != null) {
        for (Attribute attribute : findItem.getAttributes()) {
          if (attribute.getName().equals(
                COLLABORATION_ATTRIBUTE_WORKFLOW_MODEL)) {
            workflowModel = attribute.getValue();
            break;
          }
        }
      }

      this.engine = EngineFactory.newEngine(new File(workflowModel));
    }
  }
}
```

<PcsfBpmnEngine.java>

```
/**
 * Faculty of Computer Science, University of New Brunswick
 */
package ca.unb.cs.pcsf.services.snc.engine;

import java.io.*;
import java.util.*;
import org.apache.log4j.Logger;
import org.jbpm.api.*;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.simpledb.*;
/**
 * @author ddong
 * @version 0.1
 * @createDate Sep 6, 2012
 * @email dong.dong@unb.ca
 */
public class PcsfBpmnEngine implements Engine {
  private Logger logger = Logger.getLogger(PcsfBpmnEngine.class.getName());

  private AmazonSimpleDB sdb;
  private File processFile;
  private ProcessEngine processEngine;
  private RepositoryService repositoryService;
  private ExecutionService executionService;
  private TaskService taskService;


  /** Constructor */
```

```
public PcsfBpmnEngine(File processFile) {
  this.processFile = processFile;

  try {
    sdb = new AmazonSimpleDBClient(new PropertiesCredentials(
        PcsfBpmnEngine.class.getResourceAsStream(CREDENTIAL_FILE_PATH)));
  } catch (IOException e) {
    e.printStackTrace();
  }

  processEngine = Configuration.getProcessEngine();
  repositoryService = processEngine.getRepositoryService();
  executionService = processEngine.getExecutionService();
  taskService = processEngine.getTaskService();
}

@Override
public void deployProcess(String collaborationId) {
  logger.debug(LOGPRE + "deployProcess() start" + LOGPRE);

  // deploy a process definition and put it into database
  String processDeploymentId = repositoryService.createDeployment()
              .addResourceFromFile(processFile).deploy();
  List<ReplaceableItem> items = new ArrayList<ReplaceableItem>();
  ReplaceableItem item = new ReplaceableItem(collaborationId);
  item.withAttributes(new ReplaceableAttribute(
      COLLABORATION_ATTRIBUTE_PROCESS_DEFINITION_ID
      , processDeploymentId, true));
  items.add(item);
  sdb.batchPutAttributes(new BatchPutAttributesRequest(
      DOMAIN_COLLABORATION, items));

  logger.debug(LOGPRE + "deployProcess() end" + LOGPRE);
}

@Override
public void startProcess(String collaborationId, String processDepId) {
  logger.debug(LOGPRE + "startProcess() start" + LOGPRE);

  ProcessDefinition pd = repositoryService.createProcessDefinitionQuery()
      .deploymentId(processDepId).list().get(0);

  // start a process instance
  logger.info("starting a process instance...");
  executionService.startProcessInstanceById(pd.getId());

  // change collaboration state to be running
  List<ReplaceableItem> items = new ArrayList<ReplaceableItem>();
  ReplaceableItem item = new ReplaceableItem(collaborationId);
  item.withAttributes(new ReplaceableAttribute(
      COLLABORATION_ATTRIBUTE_CURRENT_STATE
      , COLLABORATION_STATE_RUNNING
      , true));
  items.add(item);
```

```
    sdb.batchPutAttributes(
        new BatchPutAttributesRequest(DOMAIN_COLLABORATION, items));

    logger.debug(LOGPRE + "startProcess() end" + LOGPRE);
  }

  @Override
  public void deleteProcess(String deploymentId) {
    logger.debug(LOGPRE + "deleteProcess() start" + LOGPRE);

    repositoryService.deleteDeploymentCascade(deploymentId);

    logger.debug(LOGPRE + "deleteProcess() end" + LOGPRE);
  }

  @Override
  public List<Task> getTasks(String username) {
    logger.debug(LOGPRE + "getTasks() start" + LOGPRE);
    List<Task> taskList = new LinkedList<Task>();
    List<Task> userTaskList = taskService.findPersonalTasks(username);
    List<Task> groupTaskList = taskService.findGroupTasks(username);

    for (Task task : userTaskList)
      taskList.add(task);
    for (Task task : groupTaskList)
      taskList.add(task);

    logger.debug(LOGPRE + "getTasks() end" + LOGPRE);
    return taskList;
  }

  @Override
  public List<Task> getCurrentTask() {
    logger.debug(LOGPRE + "getCurrentTask() start" + LOGPRE);
    logger.debug(LOGPRE + "getCurrentTask() end" + LOGPRE);
    return taskService.createTaskQuery().list();
  }

  @Override
  public void submitTask(String taskId) {
    logger.debug(LOGPRE + "submitTask() start" + LOGPRE);
    taskService.completeTask(taskId);
    logger.debug(LOGPRE + "submitTask() end" + LOGPRE);
  }

  @Override
  public void takeTask(String taskId, String username) {
    logger.debug(LOGPRE + "takeTask() start" + LOGPRE);
    taskService.takeTask(taskId, username);
    logger.debug(LOGPRE + "takeTask() end" + LOGPRE);
  }
}
```

B.3 Data Transfer Service

```
/**
 * Faculty of Computer Science, University of New Brunswick
 */
package ca.unb.cs.pcsf.services.dtr;

import static ca.unb.cs.pcsf.services.DataTransferServiceConstants.*;
import java.io.*;
import java.util.List;
import javax.jws.WebService;
import org.apache.log4j.Logger;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.*;

/**
 * @author ddong
 * @version 0.1
 * @createDate Sep 21, 2012
 * @email dong.dong@unb.ca
 */
@WebService(endpointInterface="ca.unb.cs.pcsf.services.dtr.DataTransferService")
public class DataTransferServiceImpl implements DataTransferService {
  private Logger logger = Logger.getLogger(DataTransferService.class.getName());
  private AmazonS3 s3;

  /** Constructor */
  public DataTransferServiceImpl() {
    try {
      s3 = new AmazonS3Client(new PropertiesCredentials(
          DataTransferServiceImpl.class
        .getResourceAsStream(CREDENTIAL_FILE_PATH)));
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  @Override
  public String[] uploadFile(String fileName, String collaborationName
                , byte[] bytes) {
    logger.debug(LOGPRE + "uploadFile() start" + LOGPRE);

    String[] result = new String[2];

    // create a temporary file with text data
    logger.info("loading the file...");
    FileOutputStream outputStream = null;
    File newFile = null;

    try {
      newFile = File.createTempFile(fileName, null);
      newFile.deleteOnExit();
```

```
      outputStream = new FileOutputStream(newFile);
      outputStream.write(bytes);
    } catch (IOException e) {
      e.printStackTrace();
    } finally {
      if (outputStream != null) {
        try {
          outputStream.close();
        } catch (IOException e) {
          e.printStackTrace();
        }
      }
    }

    // create bucket if not exist
    logger.info("checking Amazon S3 bucket...");
    String bucketName = "pcsf-s3-bucket-" + collaborationName;

    boolean isBucketExist = false;
    List<Bucket> bucketList = s3.listBuckets();
    for (Bucket bucket : bucketList) {
      if (bucket.getName().equals(bucketName))
        isBucketExist = true;
    }

    if (!isBucketExist) {
      logger.info("bucket <" + bucketName
                + "> not exist, creating the bucket...");
      s3.createBucket(bucketName);
    }

    // upload file to bucket
    logger.info("uploading file into S3 bucket...");
    String key = fileName;
    s3.putObject(new PutObjectRequest(bucketName, key, newFile));

    logger.info("file <" + fileName + "> has been uploaded to bucket <"
                + bucketName + "> with key <" + key + ">");

    // return the key and bucket name back to user
    result[0] = key;
    result[1] = bucketName;
    logger.debug(LOGPRE + "uploadFile() end" + LOGPRE);
    return result;
  }

  @Override
  public byte[] downloadFile(String bucketName, String key) {
    logger.debug(LOGPRE + "downloadFile() start" + LOGPRE);

    // create input stream from the object.
    S3Object object = s3.getObject(new GetObjectRequest(bucketName, key));
    InputStream inputStream = object.getObjectContent();
```

```
    // convert input stream into byte array.
    ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
    byte[] data = new byte[100];
    int count = -1;

    try {
      while ((count = inputStream.read(data, 0, 100)) != -1)
        outputStream.write(data, 0, count);
    } catch (IOException e) {
      e.printStackTrace();
    } finally {
      data = null;
    }

    byte[] result = outputStream.toByteArray();
    try {
      outputStream.close();
    } catch (IOException e) {
      e.printStackTrace();
    }

    logger.info("file <" + key + "> has been downloaded!");
    logger.debug(LOGPRE + "downloadFile() end" + LOGPRE);
    return result;
  }
}
```

B.4 Monitor and Control Service

```
/**
 * Faculty of Computer Science, University of New Brunswick
 */
package ca.unb.cs.pcsf.services.mnc;

import static ca.unb.cs.pcsf.services.mnc.MonitorNControlServiceConstants.*;

import java.io.IOException;
import java.util.*;

import javax.jws.WebService;
import javax.mail.*;
import org.apache.cxf.*;
import org.apache.log4j.Logger;

import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.simpledb.*;
import com.amazonaws.services.simpleemail.*;
/**
 * @author ddong
 * @version 0.1
```

```
 * @createDate Sep 21, 2012
 * @email dong.dong@unb.ca
 */
@WebService(endpointInterface
        = "ca.unb.cs.pcsf.services.mnc.MonitorNControlService")
public class MonitorNControlServiceImpl implements MonitorNControlService {
  Logger logger = Logger.getLogger(MonitorNControlServiceImpl.class.getName());
  private Timer timer = new Timer();
  private AmazonSimpleDB sdb;
  private AmazonSimpleEmailService ses;
  private PropertiesCredentials credentials;

  private String collaborationName;
  private String collaborationId;
  private String participantNames;

  /** Constructor */
  public MonitorNControlServiceImpl() {
    // get credentials
    try {
      credentials = new PropertiesCredentials(MonitorNControlServiceImpl
                .class.getResourceAsStream(CREDENTIAL_FILE_PATH));
      sdb = new AmazonSimpleDBClient(credentials);
      ses = new AmazonSimpleEmailServiceClient(credentials);
    } catch (IOException e) {
      e.printStackTrace();
    }

    getCollaboration();
    startMonitor();
  }

  @Override
  public void startMonitor() {
    TimerTask registrationRequest = new RegisterRequest();
    TimerTask taskSubmitReques = new TaskSubmitRequest();

    timer.schedule(registrationRequest, 86400000, 86400000);
    timer.schedule(taskSubmitReques, 86400000, 86400000);
  }

  @Override
  public void endMonitor() {
    timer.cancel();
  }

  /**
   * Get collaboration details from the database
   */
  private void getCollaboration() {
    String curPath = this.getClass().getResource("/").getPath();
    String[] curPaths = curPath.split("/");

    for (String s : curPaths) {
```

```
    if (s.contains("monitorNControlService"))
      this.collaborationName = s.split("-")[0];
  }

  String selectExp = "select * from `" + DOMAIN_COLLABORATION
      + "` where " + COLLABORATION_ATTRIBUTE_NAME + "='"
      + collaborationName + "'";
  Item findItem = sdb.select(new SelectRequest(selectExp)).getItems().get(0);

  if (findItem != null) {
    collaborationId = findItem.getName();
    for (Attribute attribute : findItem.getAttributes()) {
      if (attribute.getName().equals(COLLABORATION_ATTRIBUTE_PARTICIPANT))
        participantNames = attribute.getValue();
    }
  }
}

/**
 * A general method used to call web service.
 *
 * @param wsUrl
 *          web service url
 * @param method
 *          method name
 * @param arg
 *          method parameters
 * @return results
 */
private Object[] callService(String wsUrl, String method, Object... arg) {
  logger.debug(LOGPRE + "callService() start" + LOGPRE);

  Object[] resutls = null;
  JaxWsDynamicClientFactory dcf = JaxWsDynamicClientFactory.newInstance();
  Client client = dcf.createClient(wsUrl);
  try {
    resutls = client.invoke(method, arg);
  } catch (Exception e) {
    e.printStackTrace();
  }

  logger.debug(LOGPRE + "callService() end" + LOGPRE);
  return resutls;
}

/**
 * Send a notification to participant notifying new collaboration creation.
 *
 * @param pEmail
 * @param pName
 */
private void sendParticipantNotificationMail(String pEmail, String pName
      , String pId, String mailContent) {
  logger.debug(LOGPRE + "sendParticipantNotificationMail() start" + LOGPRE);
```

```
  this.verifyEmailAddress(ses);
  Properties props = new Properties();
  props.setProperty("mail.transport.protocol", "aws");
  props.setProperty("mail.aws.user", credentials.getAWSAccessKeyId());
  props.setProperty("mail.aws.password", credentials.getAWSSecretKey());

  Session session = Session.getInstance(props);
  try {
    // Create a new Message
    Message msg = new MimeMessage(session);
    msg.setFrom(new InternetAddress(MAIL_FROM));
    msg.addRecipient(Message.RecipientType.TO, new InternetAddress(pEmail));
    msg.setSubject(MAIL_SUBJECT);
    msg.setText("Dear " + pName + ",\n\n" + mailContent + "Your id: " + pId
      + "\n" + LINK_PARTICIPANT);
    msg.saveChanges();

    // Reuse one Transport object for sending all your messages
    // for better performance
    Transport t = new AWSJavaMailTransport(session, null);
    t.connect();
    t.sendMessage(msg, null);
    logger.info("one mail sent to participant notifying the new created"
      + " collaboration!");

    t.close();
  } catch (AddressException e) {
    e.printStackTrace();
    logger.info("Caught an AddressException, which means one or more of your "
        + "addresses are improperly formatted.");
  } catch (MessagingException e) {
    e.printStackTrace();
    logger.info("Caught a MessagingException, which means that there was a "
      + "problem sending your message to Amazon's E-mail Service check the "
      + "stack trace for more information.");
  }

  logger.debug(LOGPRE + "sendParticipantNotificationMail() end" + LOGPRE);
}

/**
 * Sends a request to Amazon Simple Email Service to verify the specified
 * email address. This triggers a verification email, which will contain a
 * link that you can click on to complete the verification process.
 *
 * @param ses
 *          The Amazon Simple Email Service client to use when making
 *          requests to Amazon SES.
 * @param address
 *          The email address to verify.
 */
private void verifyEmailAddress(AmazonSimpleEmailService ses) {
  logger.debug(LOGPRE + "verifyEmailAddress() start" + LOGPRE);
```

```
  ListVerifiedEmailAddressesResult verifiedEmails
      = ses.listVerifiedEmailAddresses();
  if (verifiedEmails.getVerifiedEmailAddresses().contains(MAIL_FROM)) {
    logger.debug(LOGPRE + "verifyEmailAddress() end" + LOGPRE);
    return;
  }

  ses.verifyEmailAddress(new VerifyEmailAddressRequest()
      .withEmailAddress(MAIL_FROM));
  logger.info("Please check the email address " + MAIL_FROM + " to verify it");

  logger.debug(LOGPRE + "verifyEmailAddress() end" + LOGPRE);
}

/**
 * Request the participant to do registration after a long time interval
 */
class RegisterRequest extends TimerTask {
  @Override
  public void run() {
    // get the registration state of all participants
    String[] participants = participantNames.split(":");
    for (String participant : participants) {
      String selectReq = "select * from `" + DOMAIN_PARTICIPANT + "`";
      List<Item> items = sdb.select(new SelectRequest(selectReq)).getItems();
      Item findItem = new Item();
      String participantIsReg = "";
      String participantEmail = "";
      String participantName = "";
      String participantId = "";

      if (!items.isEmpty()) {
        for (Item item : items) {
          if (item.getName().equals(participant)) {
            findItem = item;
            break;
          }
        }
      }

      if (findItem != null) {
        participantId = findItem.getName();
        for (Attribute attribute : findItem.getAttributes()) {
          if (attribute.getValue().equals(PARTICIPANT_ATTRIBUTE_IS_REG))
            participantIsReg = attribute.getValue();
          if (attribute.getValue().equals(PARTICIPANT_ATTRIBUTE_EMAIL))
            participantEmail = attribute.getValue();
          if (attribute.getValue().equals(PARTICIPANT_ATTRIBUTE_NAME))
            participantName = attribute.getValue();
        }
      }

      // send an email to those who has not done registration
```

```
        if (participantIsReg.equals(PARTICIPANT_IS_REG_NO))
          sendParticipantNotificationMail(participantEmail, participantName
        , participantId, MAIL_CONTENT_REMIND_PARTICIPANT_TO_REG);
      }
    }
  }

  /**
   * Request the user to submit task after a long time interval
   */
  class TaskSubmitRequest extends TimerTask {
    @Override
    public void run() {
      // get the assignee of the current task
      logger.info("get current task...");
      String url = WSURL_PRE + collaborationName
        + "-scheduleNCoordinateService/ScheduleNCoordinateService?wsdl";
      String method = "getCurrentTask";
      Object[] tskResults = callService(url, method, collaborationId);
      List<?> tskResultList = (ArrayList<?>) tskResults[0];
      for (Object o : tskResultList) {
        String s = (String) o;
        String[] infos = s.split(",");
        String assignee = infos[2];

        // send an email to the assignee if the task is running over time
        String participantEmail = "";
        String participantName = "";
        String participantId = "";
        String selectReq = "select * from `" + DOMAIN_PARTICIPANT + "` where "
        + PARTICIPANT_ATTRIBUTE_ROLE + " = '" + assignee + "'";
        List<Item> items = sdb.select(new SelectRequest(selectReq)).getItems();
        if (items != null) {
          for (Item findItem : items) {
            participantId = findItem.getName();
            for (Attribute attribute : findItem.getAttributes()) {
              if (attribute.getValue().equals(PARTICIPANT_ATTRIBUTE_EMAIL))
                participantEmail = attribute.getValue();
              if (attribute.getValue().equals(PARTICIPANT_ATTRIBUTE_NAME))
                participantName = attribute.getValue();
            }
            sendParticipantNotificationMail(participantEmail, participantName
                , participantId, MAIL_CONTENT_REMIND_PARTICIPANT_SUBMIT_TASK);
          }
        }
      }
    }
  }
}
```

Appendix C

Constants Definition

```
// domain names
public static final String DOMAIN_CREATOR = "Creator";
public static final String DOMAIN_PARTICIPANT = "Participant";
public static final String DOMAIN_COLLABORATION = "Collaboration";

// domain creator attributes
public static final String CREATOR_ATTRIBUTE_NAME = "Name";
public static final String CREATOR_ATTRIBUTE_PASSWORD = "Password";
public static final String CREATOR_ATTRIBUTE_EMAIL = "Email";

// domain collaboration attributes
public static final String COLLABORATION_ATTRIBUTE_NAME = "Name";
public static final String COLLABORATION_ATTRIBUTE_CREATOR_ID = "CreatorId";
public static final String COLLABORATION_ATTRIBUTE_CURRENT_STATE
         = "CurrentState";
public static final String COLLABORATION_ATTRIBUTE_WORKFLOW_MODEL
         = "WorkflowModel";
public static final String COLLABORATION_ATTRIBUTE_PARTICIPANT = "Participant";
public static final String COLLABORATION_ATTRIBUTE_PROCESS_DEFINITION_ID
         = "ProcessDefId";

// domain participant attributes
public static final String PARTICIPANT_ATTRIBUTE_NAME = "Name";
public static final String PARTICIPANT_ATTRIBUTE_EMAIL = "Email";
public static final String PARTICIPANT_ATTRIBUTE_COLLABORATION_ID
         = "CollaborationId";
public static final String PARTICIPANT_ATTRIBUTE_IS_REG = "IsReg";
public static final String PARTICIPANT_ATTRIBUTE_ROLE = "Role";
public static final String PARTICIPANT_ATTRIBUTE_GROUP = "Group";

public static final String PARTICIPANT_IS_REG_NO = "no";
public static final String PARTICIPANT_IS_REG_YES = "yes";
public static final String PARTICIPANT_NO_GROUP = "No Group";
```

```
// collaboration state
public static final String COLLABORATION_STATE_NEW_CREATED = "new created";
public static final String COLLABORATION_STATE_DEPLOYED = "deployed";
public static final String COLLABORATION_STATE_RUNNING = "running";
public static final String COLLABORATION_STATE_STOP = "stop";
public static final String COLLABORATION_STATE_FINISH = "finish";

// email
public static final String MAIL_FROM = "pcsf.notification@gmail.com";
public static final String MAIL_SUBJECT = "Collaboration Notification From PCSF";
public static final String MAIL_CONTENT_CREATOR_ALL_REG_DONE
         = "All participants have registered into the collaboration!\n"
        + "Please click the following link to run it:\n\n";
public static final String MAIL_CONTENT_CREATOR_DEPLOY_SUCCESS
         = "You have deployed a collaboration, an email has been sent to each"
        + " participant and you can run the collaboration after all participant"
        + " done registration.\n\nClick the following link to check the"
        + " collaboration:\n\n";
public static final String MAIL_CONTENT_PARTICIPANT_REMIND_REG
         = "You are asked to participant a collaboration!\nThis is a remind email"
        + " notifying you to do the registration!\nPlease click the following"
        + " link to register into the collaboration using your given id:\n\n";
public static final String MAIL_CONTENT_PARTICIPANT_REMIND_SUBMIT_TASK
         = "This is a remind email notifying you to submit the task!\nPlease"
        + " click the following link to view your task using your given id:\n\n";
public static final String MAIL_CONTENT_PARTICIPANT_REG_REQUIRE
         = "You are asked to participant a collaboration!\nPlease click the"
        + " following link to finish registration using your given id:\n\n";

public static final String LINK_CREATOR
        = "http://ec2-50-16-63-235.compute-1.amazonaws.com:8080/pcsf/index.jsp?"
        + "action=creatorLogin";
public static final String LINK_PARTICIPANT
        = "http://ec2-50-16-63-235.compute-1.amazonaws.com:8080/pcsf/index.jsp?"
        + "action=participantLogin";

public static final int MAX = 1000000;

public static final String LOGPRE = ">>>>";
public static final String CREDENTIAL_FILE_PATH = "AwsCredentials.properties";

// collaboration save path
public static final String SAVE_PATH
         = System.getProperty("user.home") + File.separator + "pcsf";

// web service url
public static final String WSURL_PRE = "http://localhost:8080/";
```

Vita

Candidate's full name:

Dong Dong

University attended (with dates and degrees obtained):

Jilin University, China
September, 2004 - July, 2008
Bachelor of Software Engineering

Publications:

Dong Dong, Weichang Du, *Peer Collaboration Service Framework*, UNB Computer Science 2012 Research Expo Poster, http://www.cs.unb.ca/research-expo/2012/submissions/Dong%20Dong_q72ul@unb.ca_Poster.pdf, April 10, 2012

Dong Dong, Weichang Du, *Peer Collaboration as a Service*, AWoSS 2012, the 4th Atlantic Workshop on Semantics and Services, http://awoss.org/slides/Dong%20Dong%20-%20UNB.pdf, May 4, 2012, Halifax, NS