uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Hao Liu

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.C.S. (Master of Computer Science)

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Model Based Enterprise Processes in a Service Oriented Architecture

TITRE DE LA THÈSE / TITLE OF THESIS

Liam Peyton

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

Thomas Tran

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Michael Weiss

Abdulmotaleb El Saddik

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Model Based Enterprise Processes In a Service Oriented Architecture

Hao Liu

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the MSc degree in Computer Science

Computer Science
School of Information Technology and Engineering
University of Ottawa

Your file Votre référence
ISBN: 978-0-494-18441-7
Our file Notre référence
ISBN: 978-0-494-18441-7

Acknowledgement

I would like to specially thank Dr. Liam Peyton for supervising my thesis research. He helped me finalizing my research topic, performing project implementation and reviewing my thesis writing. Besides, I also want to thank the financial support he provided to me during my graduate study.

Also thanks to Dr. Thomas Tran. He helped me reviewing my thesis drafts and gave me comments and suggestions on the writing skills and how to organize formal research documents.

I also want to take this opportunity to thank all the students who participated in our project implementations.

Thanks to my family. They gave me support and make me have enough energy and time to perform thesis research and writing.

This is for my lovely daughter Emily.

Abstract

Software systems are becoming increasingly complex. A single application needs to provide capabilities and functionalities that cut across an entire enterprise scope. At the same time, many different applications may need to interact with each other as part of an overall business process. Traditional approaches for software system development do not provide efficient solutions to address this problem. This thesis explores how to leverage emerging standards, BPEL (Business Process Execution Language for Web Service), XForms, SOA (Service Oriented Architecture) and XML (eXtensible Markup Language), to implement a Model-based SOA software development approach. It also compares Model-based SOA approach with two other traditional software development approaches in the implementation of an example application in the area of health care: an approval process for accessing data in a medical data warehouse.

Table of Content

List of Figures

List of Tables

Glossary

API (Application Programming Interface): A computer programming interface.

BPEL (Business Process Execution Language for Web Service): A high level XML based language which is used to define business process.

BPM (Business Process Management): Activities performed to manage and optimize business processes.

C#: An objected oriented programming language developed by Microsoft.

Correlation: A programming mechanism used to make sure the messages can be sent to the corresponding instances when there are multiple instances of application running.

DOM (Document Object Model): A programming approach to map XML structure as a tree structure

EAI (Enterprise Application Integration): An integration approach to bring together a set of enterprise software applications

Eclipse: An open source integrated development environment. It is widely used to develop Java programs

EJB (Enterprise Java Bean): A J2EE (Java 2 Platform, Enterprise Edition) specification for server side, enterprise-level application. Its architecture supports scalability, transaction and security

EPAL (Enterprise Privacy Authorization Language): A XML based privacy language used to specify enterprise privacy policies.

ERP (Enterprise Resource Planning): A set of enterprise software applications with predefined functionality that covers the major aspects of enterprise management needs

J2EE (Java 2 Platform Enterprise Edition): A Java specific software development environment for developing web based enterprise application.

Java: An objected oriented programming language first developed by SUN

JavaScript and VBScript: Script languages embed in the HTML pages to perform tasks like user input validations, popping up windows or animations and so on

JSP (Java Server Page) and Servlet: Technologies defined in J2EE standard to dynamically generate web page content

Model-Based Service Oriented Architecture (SOA) approach: A software development approach proposed by this thesis. It leverages emerging standards BPEL, XForms, Service Oriented Architecture and XML to build complicated enterprise processes

Non-Functional requirements (NFR): NFR refers, in software system engineering, to a software requirement that describes not what the software will do but how the software will do it. [Chung1999]For example, aviliability, reliability, performance, response time are NFR.

OASIS: Organization for the Advancement of Structured Information Standards

Policy language: A policy language is used to formally describe the details of privacy policy. Enterprise Privacy Authorization Language (EPAL) is an example of policy language

Privacy Officer: The person(s) within an organization to manage the privacy issues which includes regulating enterprise scope privacy policies, executing such policies and applying privacy laws

Rich client: The client side in a client-server architecture which performs intensive data processing, rendering user interface and so on. Rich client usually requires installing software component at client side

RPC (Remote Procedure Call): A protocol which allows one application communicates with another application

SOA: Service Oriented Architecture

SOAP: Simple Object Access Protocol

SQL (Structured Query Language): A common used database query language

SSOA (Simple Service Oriented Architecture): A software development approach which leverages SOA to develop simple application

Stub or Proxy: A stub or proxy is a local representative of remote function. When an application invokes remote web service functionality, it normally calls local stub or proxy. Then stub or proxy actually invokes web service provider.

SWA (Simple Web Application): A software development approach which leverages web technologies such as JSP and Servlet to develop simple web application

UDDI: Universal Description, Discovery, and Integration

W3C: World Wide Web Consortium

WSCI (Web Service Choreography Interface): A specification by the W3C and is a XML-based language for describing interfaces used to specify the flow of messages at interacting Web Services.

WSDL: Web Service Definition Language

WSFL: Web Service Flow Language

WSIF: Web Service Invocation Framework

XForms: An XML format for the specification of user interfaces, specifically web forms.

XLANG: A business language which is an extension of WSDL (Web Service Description Language) in order to describe the business process.

XML: eXtensible Markup Language

XML Schema: A language used to describe the structure and content of XML document.

XPath: An expression used to locate a specific node in the XML document.

XSLT (Extensible Stylesheet Language Transformation): A XML based language which is used to transform XML document to other XML documents or HTML etc.

1 Introduction

1.1 Problem statement

Software systems are becoming increasingly complex. A single application needs to provide capabilities and functionalities that cut across an entire enterprise scope. At the same time, many different applications may need to interact with each other as part of an overall business process. Traditional approaches for software system development do not provide efficient solutions to address this problem. This thesis explores how to leverage emerging standards, BPEL (Business Process Execution Language for Web Service)[1], XForms[2], SOA (Service Oriented Architecture) and XML (eXtensible Markup Language), to implement a new software system development approach. We illustrate our approach with an example application in the area of health care: an approval process for accessing data in a medical data warehouse.

Health care information is very sensitive data. Any misuse of these data can lead to serious privacy breach. This is especially true when health care information is electronically available. However, accessing and sharing electronic health care information is also very important. In order to provide high quality healthcare service to patients, doctors, nurses, researchers, insurance companies, employers all need to access patients' health information. When that information is catalogued, integrated and made available for study in a medical data warehouse, an approval process is needed to apply privacy protection while people access and sharing these data.

A comprehensive approval process for sensitive data access needs to include functionalities from different aspects, such as a security functionality to secure data access channel, a policy engine to define privacy policy, an audit trail system to track data access and an online collaboration mechanism to resolve policy dispute with privacy officer[3] [Peyton2005]. Each of those functionalities targets a specific aspect of the overall approval process.

---

[1] A high level XML based language which is used to define business process.
[2] An XML format for the specification of user interfaces, specifically web forms
[3] The person(s) within an organization to manage the privacy issues which includes regulating enterprise scope privacy policies, executing such policies and applying privacy laws

Those essential functions mentioned above may already exist within enterprise/hospital IT systems. For example, a hospital may already have a separate IT application running to secure data access. Also, another IT system may already be used to apply enterprise policy infrastructure. Due of their heterogeneous nature, these functionalities may run on different operation systems, be developed in different programming languages, or even delivered by different developing teams/venders. In order to leverage these functions and reuse them, we need an approach to integrate and orchestrate all of these disparate and loosely coupled functions to create a sensitive data access approval process. This approach must be able to adequately integrate all the disparate functions and flexible enough to accommodate any changes happened in the approval process.

Web services facilitate heterogeneous integration. But a simple web service invocation approach lacks business process definition and is difficult to orchestrate sophisticated functions in a business process needed by the approval process we are using as an example.

Service Oriented Architecture addresses the concept of orchestrating functionalities to implement a business process. Since there is no explicit business process definition language within this concept, the traditional approach hard codes business process inside programming languages. But programmers generally do not understand the business logic very well. Besides, in order to orchestrate a business process, programmers need to manually implement all the low level coding like generating web service stub/proxy[4], maintaining asynchronous message correlation[5], guaranteeing business transaction integrity and so on. Furthermore, because the business process is coded inside programming languages, it is hard to visualize and monitor and can not be easily orchestrated or changed on the fly.

---

[4] A stub or proxy is a local representative of remote function. When an application invokes remote web service functionality, it normally calls local stub or proxy. Then stub or proxy actually invokes web service provider

[5] A programming mechanism used to make sure the messages can be sent to the corresponding instances when there are multiple instances of application running.

1.2 Motivations and objectives

Our overall objective is to demonstrate that Model-based approaches to enterprise processes in a service oriented architecture is a superior to traditional web-based approaches. Specifically, this approach leverages emerging standards BPEL, XForms, SOA and XML.

The example chosen to illustrate the thesis contributions is building a prototype application to support a sophisticated approval process in order to apply privacy protection for health care data access

Addressing the difficulty in building such software based on the functionalities from heterogeneous IT systems, we are motivated to satisfy the soft-goals of the non-functional requirement[6] during the software system development. According to [Araujo2002], these soft-goals, such as development effort, maintainability, response time, scalability and so on, decide the selection of design patterns/approaches when there are different patterns/approaches can be applied to construct required system functionalities.

Therefore, the important part of our objectives includes:

- The business process which implements the approval process must be easy to be created.
- The business process can easily be understood and monitored
- Because of its volatile character, the business process of purposed approval process also needs to be easily changed without coding at programming level.
- Last, the user interface of purposed approval process also needs to be easily crated and changed.

[6] Non-Functional requirements (NFR) refers, in software system engineering, to a software requirement that describes not what the software will do but how the software will do it. [Chung1999]For example, aviliability, reliability, performance, response time are NFR.

1.3 Thesis contributions

In this thesis, we propose a Model-based service oriented architecture approach to build enterprise process. The enterprise process is orchestrated upon functionalities disperse among heterogeneous IT systems and it invokes them through web services. The process, user interfaces and data exchanged are all model-based using BPEL, XForms and XML.

In the model-based SOA approach, we use BPEL to define business process on top of application functionality layer and orchestrate the functionalities on the fly. Then we use XForms to define User Forms and make approval process's user interface to be driven by XML data model and XML Schema. The user side validation is also handled by XForms. Besides, by using XML data model, XML Schema, we define a cohesive message exchange format among interacting applications. Then, we implement our approach on an example from the Ottawa Hospital to automate their paper based data access approval process with proposed enterprise process.

Based on the comparison result on several criteria, development efforts, debugging, maintenance and performance, we demonstrate that the model-based SOA approach makes:

- the business process of the targeting system easy to be created, without having to write Java code for example
- the business process of the targeting system can easily be understood and monitored
- the business process of the targeting system can be easily changed, without code changes
- the user interface of the targeting system can be easily created and changed

Therefore, we believe that the Model-based approach in a service oriented architecture is superior to traditional web-based approaches.

1.4 Thesis organization

Chapter 2 covers background information from both business and technical aspects. It also introduces some related work from other researchers to address the similar problems.

Chapter 3 presents a software system development approach using BPEL, XForms, SOA and XML

Chapter 4 includes a case study based on an actual project to create a data access approval process using the Model-based SOA approach, proposed in this thesis.

Chapter 5 is the validation and verification for thesis proposed Model-based SOA approach based on the comparison of two other approaches, the SWA (Simple Web Application) and the SSOA (Simple Service Oriented Architecture) approaches. It also compares the thesis solution with several related work solutions

Chapter 6 is the summary of this thesis and gives the conclusion from the analysis of comparison results. Besides, it also presents some limitations of thesis solution and the future work directions.

2 Background

In Chapter 2, we give the background information from both business and technical sides. In Section 2.1, the importance of sharing health care data and its privacy concerns are introduced. Then the sensitive data access approval process and business process management concepts are presented. In Section 2.2, we list different integration technologies such as data level, API (Application Programming Interface), ERP (Enterprise Resource Planning), EAI (Enterprise Application Integration), Web Service and SOA. Section 2.3 introduces some related works performed by other researchers.

2.1 Business drivers

2.1.1 Motivations of access/sharing health care information:

Rindfleisch described the needs for different stake holders to access patient information [Rindfleisch1997]. In order to improve health care quality, doctors, nurses and hospital administrators need to access patient information. Medical researchers also need to access this information for better support of clinical research. Besides, as the scenarios, business to business share of patient information, presented in [Peyton2004], patients' employers and insurance companies may also want to get patients information for completing medical insurance claims.

Thanks to the development of information technologies, some of health care information is already electronically accessible like billing and scheduling, laboratory result reporting, and diagnostic instrument systems (such as radiology and cardiology) [Rindfleisch1997].

Once the health care information is in electronic format, a typical usage of these information allows a researcher using a PC to visit a web site which provides the access to health care information. The researcher input some identity information to login to system. Then, by filling out searching criteria, the researcher gets what the information he wants.

2.1.2 Threats of abuse usage of health care information

Since health care information contains very sensitive data about patients, any abuse of this information can bring them serious impact. The information about patients' fertility and abortions, emotional problems and psychiatric care, sexual behaviors, sexually transmitted diseases, HIV status, substance abuse, physical abuse, genetic predispositions to diseases etc are sensitive enough that any abuse or disclose of these information may cause social embarrassment/ prejudice to the patients, affect their insurability, or loss of jobs [Rindfleisch1997].

The electronic format of health care information can make these potential breaches worse. Pavlovic described the factors which bring more privacy concerns due to electronic format of the privacy sensitive data in [Pavlovic2004]. Once the information is online (electronically available), it makes the data easier to be collected, stored, disseminated, accessed and aggregated.

Therefore, we need an adequate data access approval process to prevent health care information from being abused while permitting necessary and legitimate access/share.

2.1.3 Essential parts of a sensitive data access approval process

There are several essentials parts for managing sensitive data access as described in [Arnesen2003]:

Reference Monitor is an access control mechanism which denies or permits data access based on the decision defined in policy languages[7]. According to the Reference Monitor's access decisions, a ***Personal Data Broker*** enforces the data access by locating actual requested data. It is used as a mapping from the requests to the actual data. Therefore, personal data broker can hide the identity of sensitive data and apply any necessary obligations before accessing it. Besides, a ***Monitoring*** element is used to monitor data access process and analyze the audit trails. Furthermore, a ***Dispute Manger*** offers a mechanism to

[7] Policy languages are used to formally describe the details of privacy policy. Enterprise Privacy Authorization Language (EPAL) is an example of policy language

resolve dispute by submitting complications to data collectors or some other relevant authority.

Peyton gave a clearer suggestion for a sensitive data access approval process in [Peyton2005]:

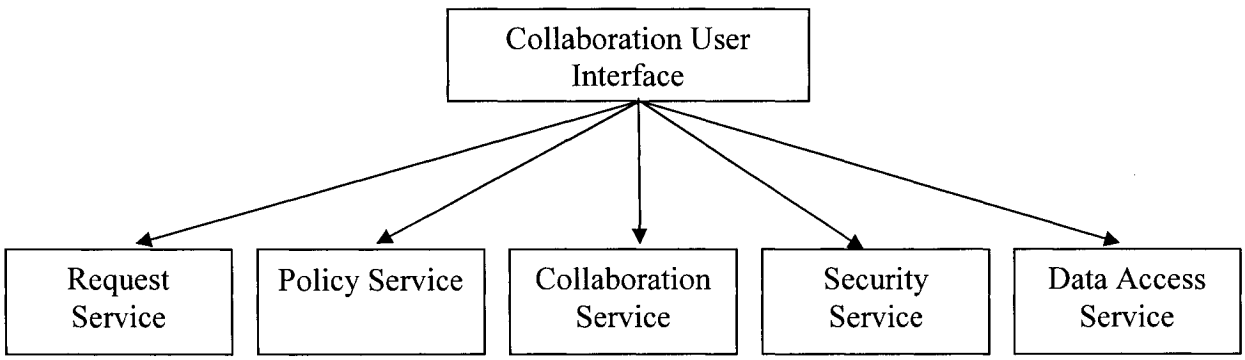

Figure 1: Simple Application

In Figure 1, the collaborative approval process proposed by Peyton includes:

Request Service: Request service implements user interface which accepts user input.

Policy Service: An EPAL (Enterprise Privacy Authorization Language)[8] based policy engine which defines/applies privacy policy within the organization.

Collaboration Service: A web based online application which permits the data requestor interacts with a privacy officer in order to get approval for the data required. The collaboration process is necessary because under some conditions, the request cannot be granted automatically by policy engine. Either additional information may be needed, or current policies prevent the access but permit the requestor to change some details under the instructions of the privacy officer in order to get the request be approved.

Security Service: Security service provides data anonymization and digital signature functionalities.

[8] A XML based privacy language used to specify enterprise privacy policies.

Data Access Service: It stores the actual sensitive health care data. To achieve efficient data access, the data is categorized and structured along different dimensions and is usually referred as star schema [Kimball2002]. After the request gets approved, data access service is invoked by the approval process as the last step to get the actual data. The data is given as an URL link which points to the actual health care data.

2.1.4 Business process management

Hammer defined business process in [Hammer1993] as "a collection of activities that takes one or more kinds of input and creates an output that is of value to the customer". In an order fulfillment procedure, although individual tasks within this process, such as receiving order form, picking the goods from warehouse etc., are important, it is the overall process to make the goods delivered to the customers. Therefore, process-based thinking is crucial for today's new business environment.

One interesting example in [Hammer1993] is about how IBM Credit streamlined its financing process. By observing the execution of the old business processes and analyzing it, IBM Credit managers found that the handing of the forms among the departments consumed most of processing time. They realized the problem did not lie in the tasks and the people performing them, but in the structure of the process itself. Therefore, IBM Credit re-designed and optimized the credit approval process. They replaced pervious departmental specialists with one generalist therefore eliminated the handoffs among departments which consumed most of the time in the old process. Finally, they achieved 90% reduction in cycle time, and at the same time a hundredfold improvement in productivity.

Since business process has a critical impact on the enterprises in achieving their objectives, the BPM (Business Process Management)[9] needs to be implemented to manage process efficiency and flexibility.

Scholz described a typical BPM lifecycle in [Scheer2004] which includes steps like Figure 2

[9] Activities performed to manage and optimize business processes.

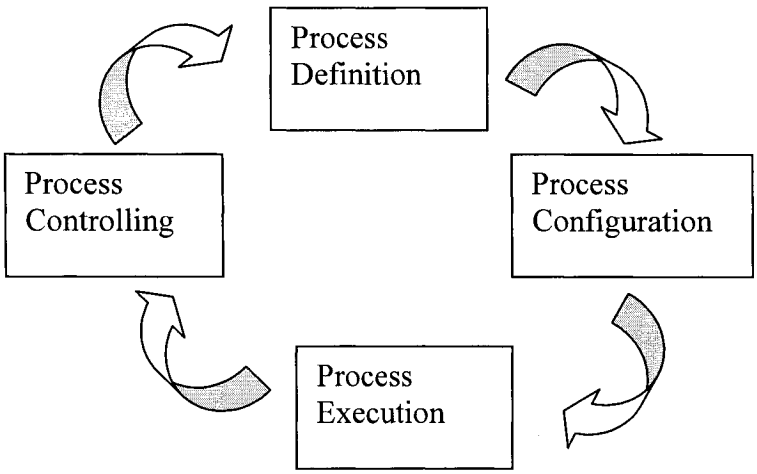

Figure 2: Business Process Lifecycle

He emphasized only the continuous improvement of business process can "generate sustained and lasting competitive advantages --- a genuine business process lifecycle". Therefore, BPM is not a "one-off action". On the contrary, it should follow the lifecycle as described in Figure 2 as a closed loop.

2.2 Integration Drivers

From above business aspect analysis, we need a business process to implement our approval process for sensitive data access. On the other side of the spectrum, this business process needs the integration of dispersed functionalities from technical means.

In both [Endlich2004] and [Raut2003], the authors summarized different integration technologies. We can categorize them according to different level they are implemented at as below:

2.2.1 Data Level Integration

Data level integration is a straightforward and relatively easier to implement integration solution.

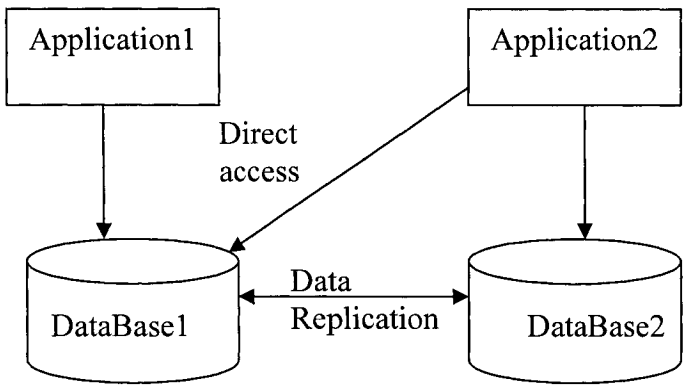

Figure 3: Data Level Integration

In the scenario as presented in Figure 3, the application2 can access the information resides in the database1 of the application1. There are two approaches to realize this data level integration. In the first approach, the application2 can directly access database1 if application1 permits this kind of access and provides appropriate interfaces. In the second approach, two databases can establish a replication mechanism which copy part of data, the part it needs to access, from Database1 to Database2. Depending on the requirement, the data copy can be uni-direction or bi-direction.

In order to access the low level data structure of another application, the invoking application not only needs to syntactically "read" other application's data, but also needs to semantically "understand" what is the meaning of the data. Due to its low level character, it is not easy to express syntactical and semantically meaning at data structure level[Endlich2004]. Also, it is hard to keep the data consistence if the application gives data write ability to another application which may not understand the business logic of invoked application very well. Besides, direct access the database from outside the application also arises the performance concerns because the invoked application need extra resources to handle these data access requests. Last, if the data replication approach is used, it is always a difficult decision on how to balance between data consistence and application performance. A more frequent replication schedule can decrease the data inconsistence between source and its copy. But it requires more system resource which may impact application performance.

2.2.2 Application Program Interface (API)

API is an integration technology works at function level, a higher level than data level. **Error!**

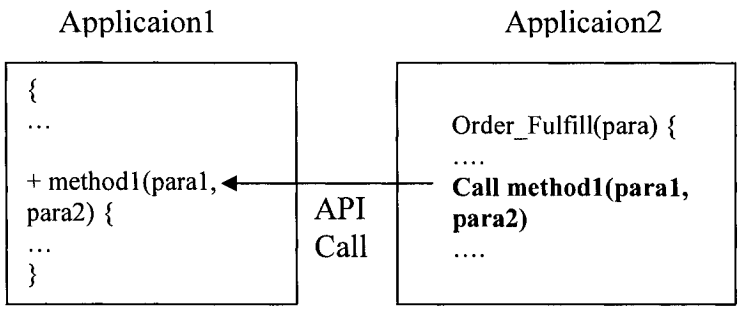

Figure 4: Application Program Interface (API)

As showed in Figure 4, if the application2 wants to use one function from the application1, it can make an API call to a function which application1 exposed as API. Those API functions can be the normal functions which applicaiton1 uses for implementing its own business logic or they can be some functions created specially for integration purpose. The latter ones generally come with some limitations or simplifications compared with the former ones. Since invoking application does not directly access the data structure of invoked application, it gives less chance for the invoking application to impact on the data consistence of the invoked application. The invoked application can put its constrains and limitations on the API functions to guarantee its own data consistence and application performance.

However, because invoking application directly invokes API functions of invoked application, it makes the two application are tightly coupled [Raut2003]. API integration approach normally requires the invoking and invoked application use same development environment since API calls are invoked within programming languages. Therefore, API is often used to integrate different modules within the same application scope. Once the integration excesses application boundary, the homogenous requirement becomes almost impossible to satisfy using API integration approach.

2.2.3 Enterprise Resource Planning (ERP)

Enterprise Resource Planning (ERP) is "a computerized set of integrated modular components used in the support of many or most of an organization's day to day processes" [Rodney2003]. It covers all the functionalities of a typical manufacturing company and extends to its suppliers and customers. Therefore, ERP itself is an integrated solution.

ERP includes a library of pre-packed "best business practice" called blueprints which involve enterprise applications in manufacturing resource planning, customer relationship management, finance, logistic, human resource etc. All these functionalities are designed and built upon a common ERP platform, therefore they are fully integrated.

On the other hand, because ERP pre-packs business blueprints, the enterprises which want to implement ERP must adapt their current business process to these blueprints. It may requires a complete re-engineering, sometimes dramatically change of their business process. This definitely brings high cost and high risk challenges to the ERP implementations. According to [Rodney2003], the average cost of ownership for an ERP implementation is about $15 million. Another drawback Kirchmer pointed in [Scheer2004] about ERP implementation is that if enterprises do not want to follow those ERP's best business practices, they have to make huge investment on software development to customize them.

2.2.4 Enterprise Application Integration (EAI)

Erasala defined Enterprise Application Integration (EAI) as "the integration of applications that enables information sharing and business processes, both of which result in efficient operations and flexible delivery of business services to the customer" [Erasala2003].

Compared with ERP's push oriented approach, which forces enterprises to accept its pre-packed business process, EAI is a pull oriented approach since it pulls functionalities within existing applications into an enterprise scope form [Lee2003].

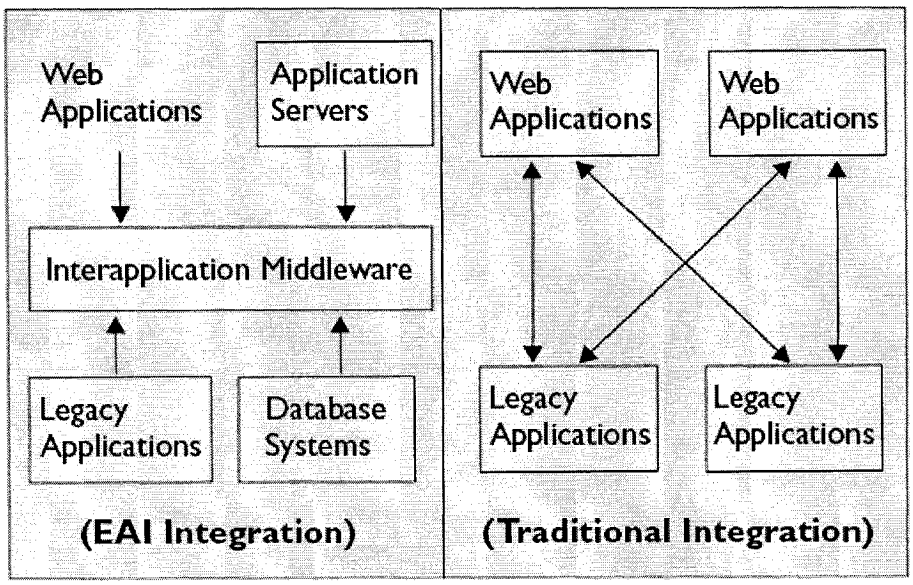

Figure 5: EAI Integration and Traditional Integration [Lee2003]

From Figure 5, we learn EAI approach is different from traditional point-to-point integration approaches, such as API integration, by introducing an "inter-application middleware". This "inter-application middleware" is normally a hub/bus/federated infrastructure which glues heterogeneous applications to form an enterprise scope EAI framework. The actual links between different applications and EAI framework are implemented by connectors/adaptors which wrap applications' specific format into a common format defined in EAI framework [Endlich2004].

In the EAI integration approach, the middleware tier and its connectors/adaptors are complicated and involve an expensive long-term investment in design [Lee2003]. Besides, there is no a standard implementation of EAI integration and different vendors have different solutions. Therefore, EAI solution is vendor depended.

2.2.5 Web Service

Web service implements the integration in the same way as API dose but releases the tightly coupled constraints. Because invoking and invoked applications communicate through a group of standards instead of direct invocation using the same programming language, web service is regarded as an enabler for heterogeneous integration.

There are several standards that are important in the concept of web service.

WSDL (Web Service Definition Language) is a W3C (World Wide Web Consortium) standard used to describe a remote service which is accessible through network. It is a XML based language which defines the details needed to invoke a web service. A WSDL file normally includes many sections, but basically it defines what are the **operations** remote service provides, what is the input and output **message** format of these operations and how to **bind** them.

SOAP (Simple Object Access Protocol) is a XML based message exchange protocol for web service invocation. According to WSDL definition, service requestor sends its request as SOAP messages through HTTP or SMTP [Endlich2004]. Since the request message is in SOAP format, web service provider knows how to parse this message according to SOAP message definition. SOAP message can be sent through HTTP which is widely used as an Internet communication protocol. Because HTTP can pass enterprise firewall, the web service requestor and provider can be located at different geographic areas and use completely different IT infrastructure. Therefore, web service realizes loosely coupled integration.

UDDI (Universal Description, Discovery, and Integration) is a registry and discovery mechanism used to register web service to a public registry. Like a telephone yellow page, UDDI allows interested parties to perform searches and downloads of the information about published web service [Newcomer2002].

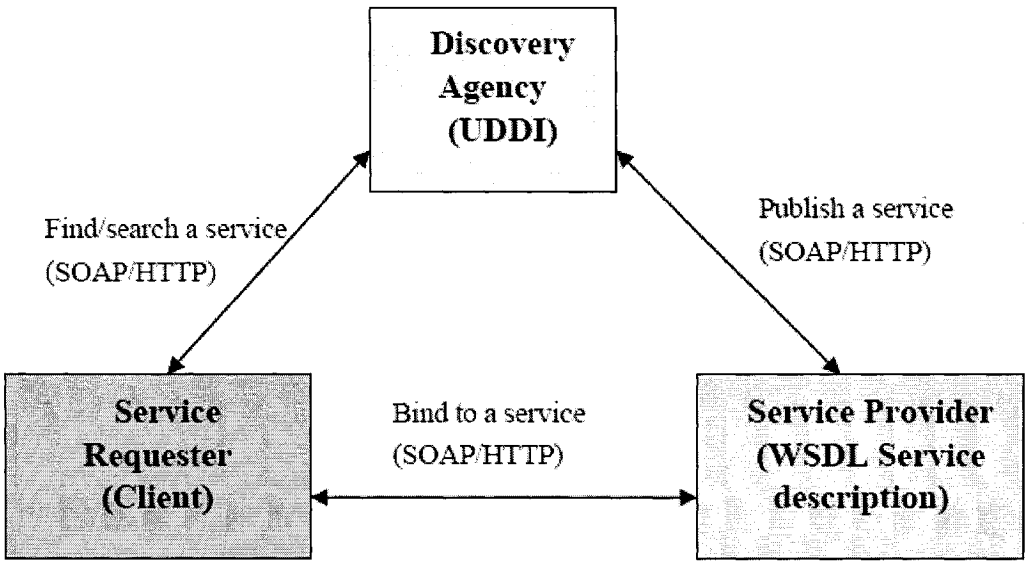

Figure 6: Basic Web Service Interaction [Endlich2004]

Figure 6 is a typical web service interaction diagram between service requester and provider. At first, service provider registers its services through UDDI. Then service requester finds the registered services on UDDI registry. UDDI also gives requester the WSDL address of web service provider. By parsing WSDL definition, requester sends requests as SOAP messages through HTTP to the provider. The provider interpreters SOAP message and redirects the input parameters to the corresponding endpoint for further processing.

WSIF (Web Service Invocation Framework) extends the above web service model. It allows other services, for example Java classes or EJBs (Enterprise Java Bean)[10], which do not communicate through SOAP as regular web service providers do, to also be described using WSDL. Therefore, the requester can invoke these services according to WSDL description just like invoking a regular web service. WSIF gives the flexibility to describe all the services in WSDL and bind them to different protocols at the back sense. The advantage of using WSIF is that the requester side code can keep unchanged if the communication protocol changes at the service provider side. Besides, since WSIF can describe Java classes and EJBs in WSDL, it also extends the integrating ability to support legacy applications which can not provide web service interfaces.

[10] EJB is a J2EE (Java 2 Platform, Enterprise Edition) specification for server side, enterprise-level application. Its architecture supports scalability, transaction and security

The web service interaction steps, as presented in Figure 6, implement a simple RPC (Remote Procedure Call)[11]. When an application needs occasionally invoke functionality from other applications, this simple web service integration can be used. But when the application needs invoke several participating functionalities or implementing a business process whose functionalities are completely composed from other applications, developers have to deal with lots of low level issues such as generating local stubs/proxies, keeping correlations between invocations and mapping message format for different applications. Therefore, simple web service RPC is not suitable to handle this requirement. Furthermore, Erl distinguished only using a few web services from establishing a service-oriented architecture in [Erl2004], the former approach "may be appropriate as a learning experience or to supplement an existing application architecture with a service-based piece of functionality that meets a specific project requirement". "There is a distinct different between an application that uses web service and an application based on a service-oriented architecture".

2.2.6 Service Oriented Architecture (SOA)

SOA is a design model which leverages the service concept and constructs new applications by building block of different services. A service encapsulates the actual implementation of application logic and interacts with invoking application via a common communication protocol. Web service is often used to establish this communication framework [Erl2004].

The benefit of using SOA to composite new application is that it can re-use the functionalities which disperse among current / legacy enterprise applications. Comparing with building all these functionalities from scratch, it can dramatically decrease the development effort in terms of manpower and time. Furthermore, by leveraging web service as the interchange framework, SOA inherits the loose couple and heterogeneous integration advantages from web service.

[11] A protocol which allows one application communicates with another application

Therefore, a typical web service SOA based application to address the data access approval process proposed by Peyton, as introduced in section 2.1.3, is constructed as Figure 7

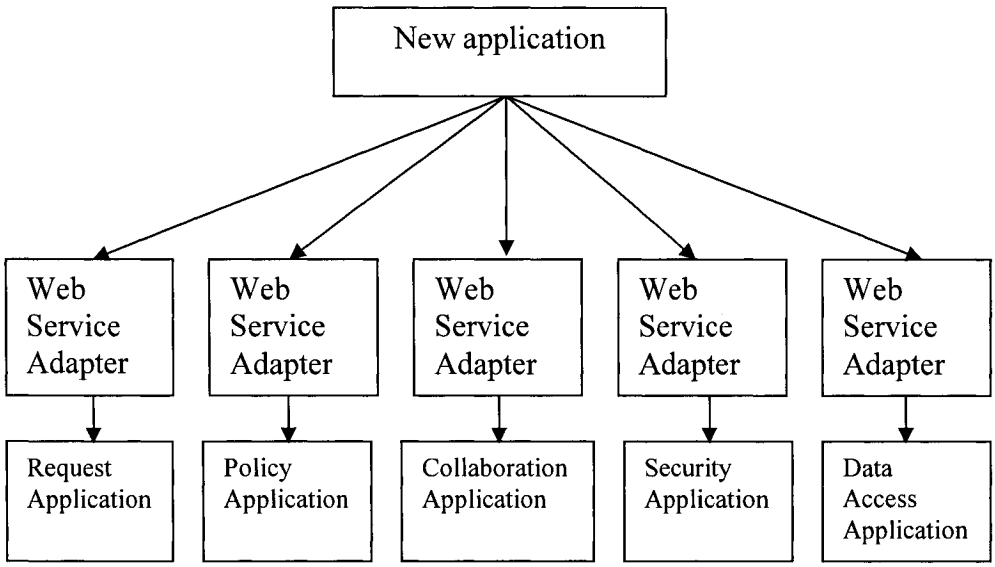

Figure 7: A simple SOA based application

The functionalities of the new data access approval process are dispersed among different applications. A web service adapter warps application functionality as a service which is well-defined, self-contained, and does not depend on the context or state of other services. A new application is developed to integrate all these services and fulfill the requirement of the data access approval process.

One drawback of above approach is that the data access approval process is implemented as a dedicated application and it is normally coded using a programming language, such as Java or C#[12]. Since both the process flows and the functionalities which are invoked by the process are defined at the same level --- programming level, there is no a clear separation of these two different logic layers. Besides, any change of process flow, like the order of invoking service or introducing a new service, needs corresponding coding change. This requirement, unfortunately, makes the understanding and maintenance of the process

[12] An objected oriented programming language developed by Microsoft.

depending on the reference to the programming languages like Java or C#. However, the programming language normally is not intuitive. Therefore, this approach limits the flexibility to change the process and make the business process hard to be understood.

Juric also presented the same limitations of using a dedicated application to integrate and automate business process in [Juric2004]. "Because the process flow and the business logic are tightly coupled, to change or modify a business process it would be necessary to modify the application." Therefore, it requires "a standard foundation for orchestrating service and a standard language for business process composition".

2.3 Related works

A privacy enforcement framework is given in [Arnesen2003] to safeguard the personal data during its whole life cycle, from collection, processing till deletion or depersonalization.

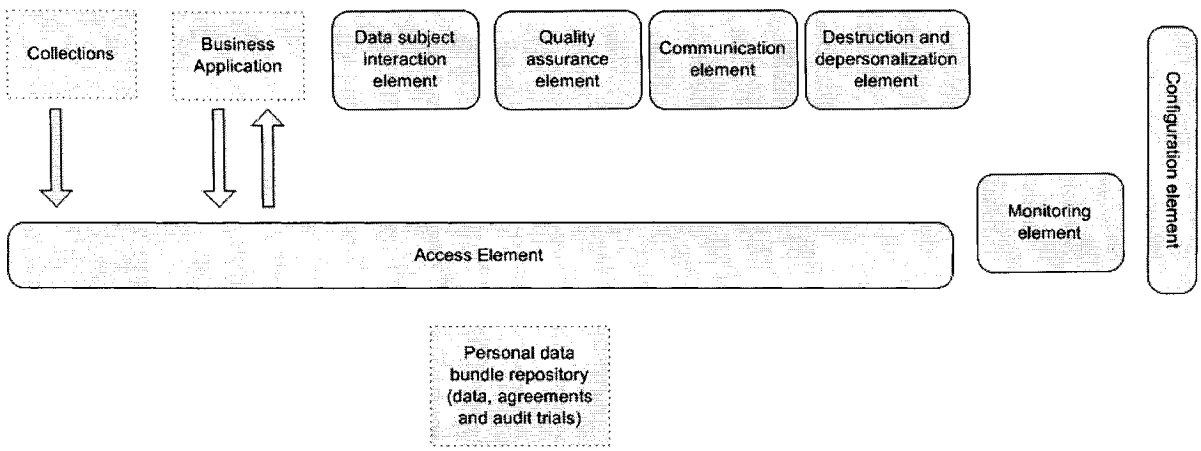

Figure 8: Overview of the framework in [Arnesen2003][13]

In the framework as shown in Figure 8, Arnesen proposes a sophisticated policy enforcement mechanism. Due to its complexity, the framework consists different **element**s. Each element is responsible for one aspect of the policy enforcement. For example, monitoring element monitors and analyses the audit trials in order to make sure the privacy policies are applied for data access. Furthermore, each element is composed of **component**s which implement

[13] Redraw by the author according to the original figure

the actual functionalities. For instance, the monitoring element includes audit manager, privacy violation detector and remote privacy audit manager components. Each of the above three components corresponds one functional point of the monitoring aspect.

From the concerns for privacy policy enforcement, Arnesen provides a sophisticated framework which is composed of various elements and components and covers different functional requirements. However, in order to implement all the functionalities of such complicated framework, it requires big development efforts. At the same time, as we mentioned in sections 1.1, some of these functionalities may already exist in the enterprise applications. The reuse of these functionalities can dramatically decrease development efforts and complexities. Even Arnesen acknowledges there is a need to integrate legacy systems in the design of the framework, but how to integrate these existing system functionalities is not addressed in [Arnesen2003]. The framework presented as in Figure 8 is not SOA based and requires big development efforts. Besides, the framework prototype Arnesen is working on is in the form of Java code. Because the framework and its workflow are implemented in Java code and there is no separate business process definition, any change of the workflow requires the code change and recompile of the whole system. This definitely increases the maintenance complexities.

In addressing the enterprise business process integration, Raut illustrates an architecture by leveraging BPM, web services and EAI technologies.

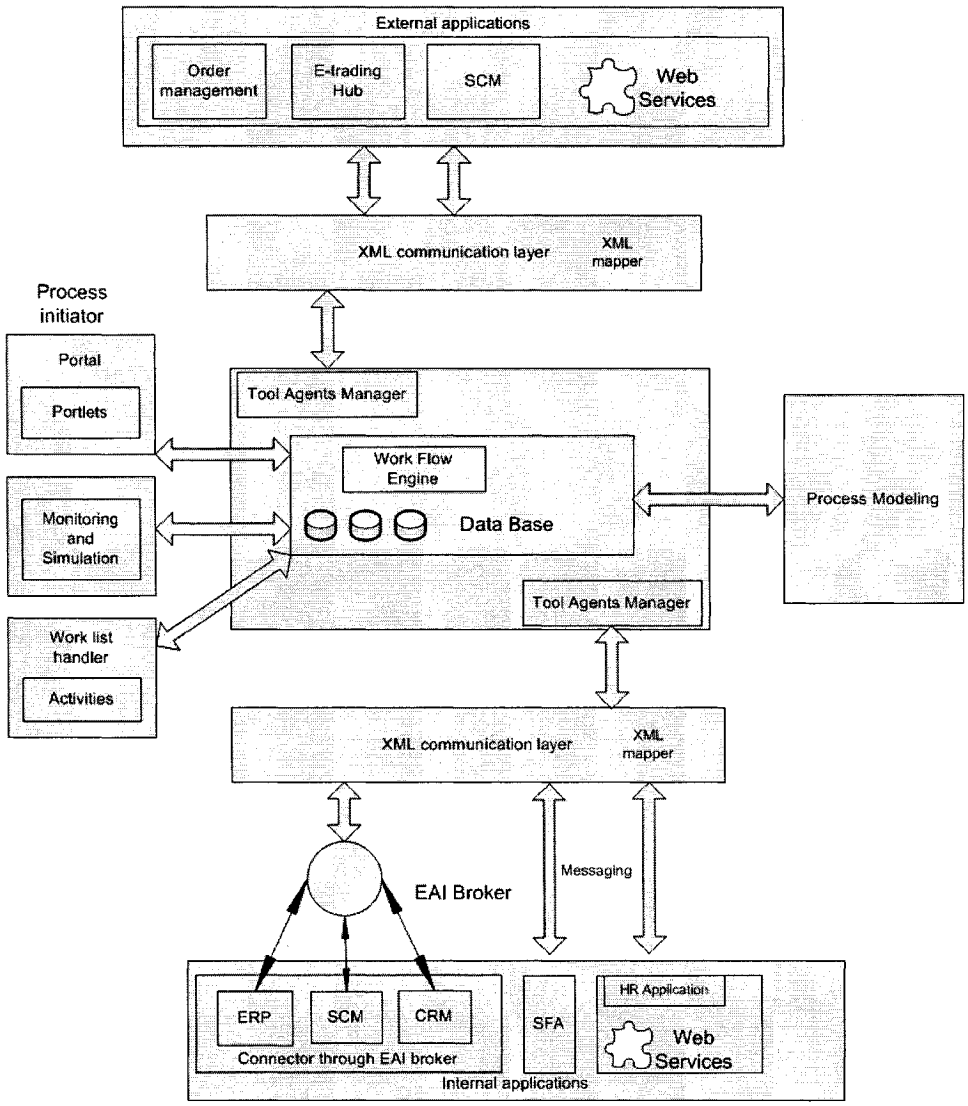

Figure 9: Enterprise Business Process Integration Architecture from [Raut2003][14]

Figure 9 is the architecture Raut proposed. The key component of this architecture is the BPM engine which is shown as "work flow engine" in the centre of
Figure 9. The BPM engine executes the business processes and dynamically invokes internal or external applications according to the work flow definition in the processes. Prior to its execution, the processes need to be defined in the business process modeler which is shown as "process modeling" in the right part of Figure 9

[14] Redraw by the author according to the original figure

While Raut proposes other integration approaches for application invocations, such as EAI and direct interacting, we pay special attention to its web service integration approach because we also use web service as the integration enabler in our approach. In his architecture, all the external applications which participate in the work flow are exposed as web services. WSDL is used to describe the endpoints of application functionalities provided by web services. In conjunction of the WSDL description, WSCI (Web Service Choreography Interface)[15] is used to specify the flow of message exchange by a web service in the context of a process. For example, WSCI is used to describe message correlation, business transaction and so on.

Since WSCI follows the choreography pattern, which will be introduced in details in section 3.2.1, the business process needs to be defined from the viewpoint of each participating web service. One drawback of the choreography invocation is that it requires each web service be aware of its participation in the business process and also needs to understand the context of the given process. It makes these participating web services process specific and loses their generalities. Therefore they can not be reused in other processes.

In [Peyton2005], a framework for business processes is given in order to address more sophisticated composite applications.

[15] A specification by the W3C and is a XML-based language for describing interfaces used to specify the flow of messages at interacting Web Services.

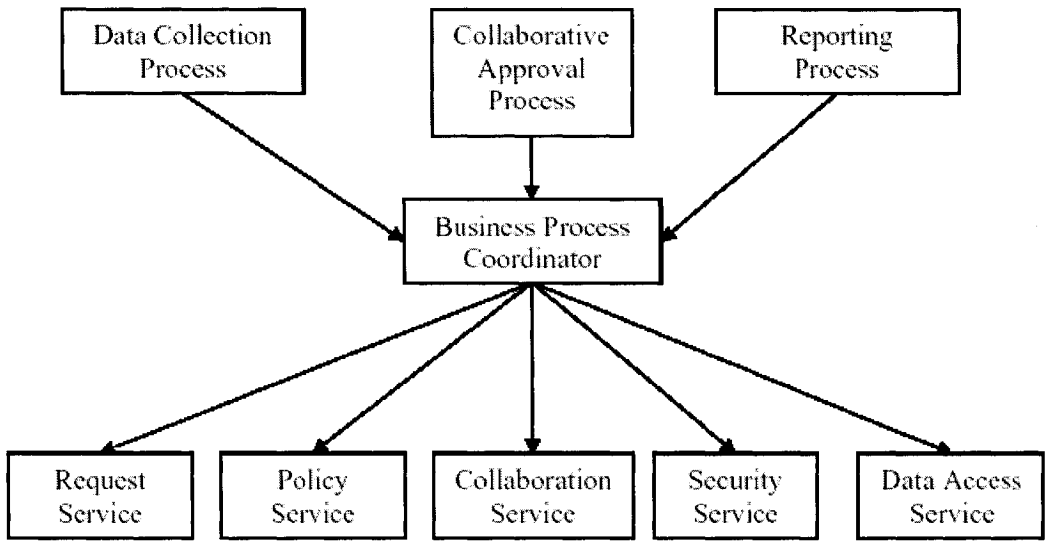

Figure 10: A framework for business process in [Peyton2005]

As illustrated in Figure 10, the framework proposed is based on SOA. All the major functionalities are provided as services. The "business process coordinator", which is shown in the centre of Figure 10, orchestrates and interweaves these participating services to implement sophisticated business process.

However, because Peyton is focusing on the collaboration environment topic, there is no detail information about how to design and implement the business process coordinator component in his framework. Neither the message exchange format for all the participating services and related user interactive details are defined in [Peyton2005]

As outlined for its future directions in [Peyton2005], this thesis is the continuing research on using model based approach integrated with business process management, XForms and XML technologies to address sophisticated enterprise process. Then Chapter 3 presents the detail information about thesis proposed approach.

3 A Model-based Service Oriented Architecture Approach to Business Process Software

In this Chapter, we adopt the emerging standards BEPL, XForms, SOA and XML to build a model based enterprise process in a service oriented architecture. First in section 3.1, we present the framework architecture of proposed approach. Then in the following sections 3.2, 3.3 and 3.4, we introduce the details of the key components in the framework architecture: business process engine, message exchange and user interface respectively.

3.1 Framework Architecture

We define a three layer framework architecture for this approach as illustrated in Figure 11:

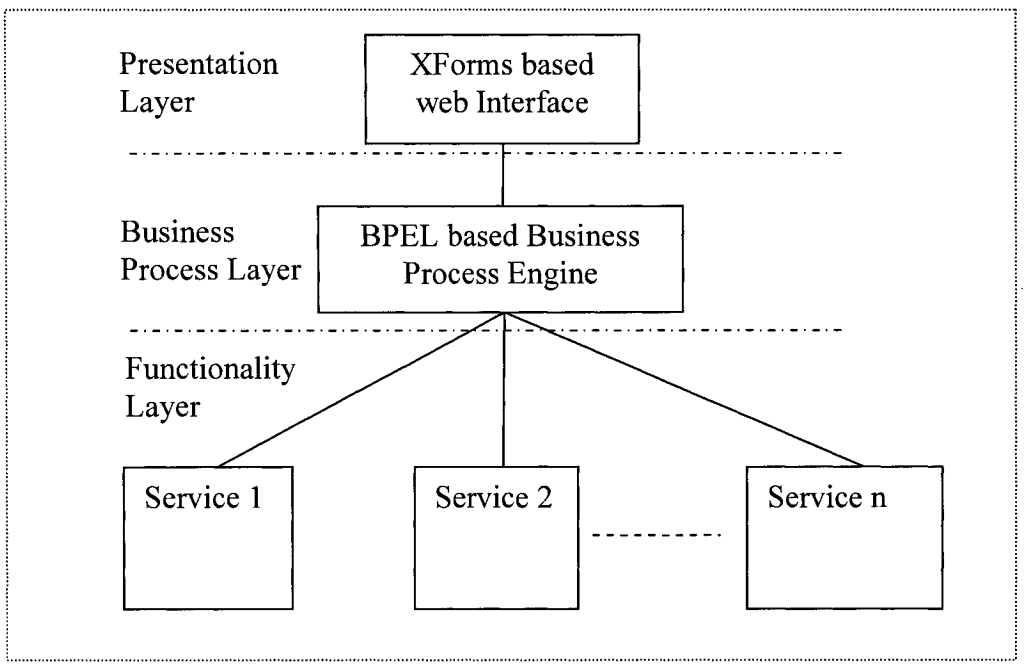

Figure 11: Three layer framework architecture

Presentation Layer: This layer provides the user interface to the end user. This layer is on top of other layers and it is the only interface by which end user communicates with the whole system functionalities. As we will introduce in section 3.4, the presentation layer is implemented using XForms based web forms which models the user interface.

Business Process Layer: This layer implements the workflow of the whole system. It applies business rules, conditions and constrains which are specific to the application domain and user requirements. In this layer, we only care about "what does it need to do" but not "how to do it". For example, in the business process layer, we need to define that after the system gets user input from user interface, it will send user input for initial technical review. But we do not implement the detail functionalities of the initial technical review module in this layer. As we will describe in section 3.2, BPEL as a high level XML based languages is used to model the business process in this layer.

Functionality Layer: This layer is the actual implementation of different functionalities which are required by the system. It is based on the SOA concept. Each functionality is provided as a service. The actual implementation of each functionality can be a dictated application specifically built for this functional requirement, or the reuse of an existing application. As the advantages we presented in section 2.2.5 and 2.2.6, these applications which provide the functionalities as services do not need to be tightly tied each other. They can be implemented in different programming languages and located in different network environments. In our framework, we use BPEL in our business process layer. As a standard BPEL implementation, it invokes functionalities using web service. Therefore, the functionality layer provides web service interface to business process layer for invocation.

3.2 Business Process Engine

The new trends for software development is summarized by Aalst are "from programming to assembling", "from data orientation to process orientation", and "from carefully planned designs to redesign and organic growth" [Aalst2003]. The crucial part within these new trends is the requirement of a business process engine to orchestrate functionalities and fulfill new business need. More precisely, Scheer suggested next generation process automation should separate the application software itself from the integration technologies and the business process design. "The application software provides the functionality need to support a business process. And business process automation engines consist logically of a workflow component that enforces the necessary process logic and data-integration component..." [Scheer2004]

Therefore, the first step in our solution is targeting on constructing a business process engine to orchestrate functionalities disperse among heterogonous applications. Furthermore, this business process engine has to be flexible enough to accommodate any new changes resulting from enterprise organic growth.

3.2.1 Service composition mode for business process engine

There are two approaches to construct the business process as introduced in [Juric2004].

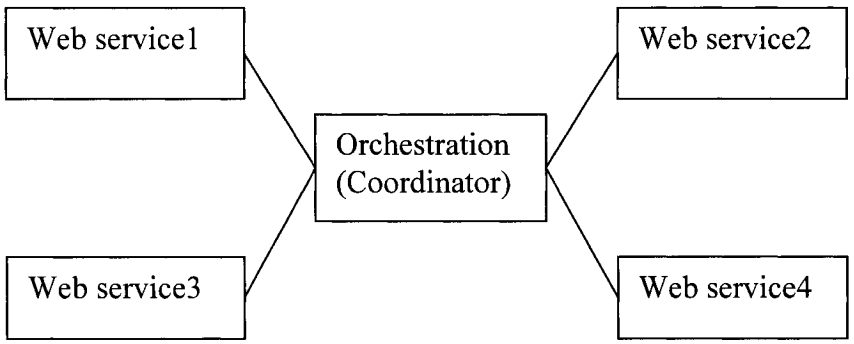

Figure 12: Orchestration invocation mode

In orchestration mode as presented in Figure 12, the central coordinator is responsible for orchestrating web service functionalities. Therefore, all the process logics and work flows are defined in the coordinator. These participating web services do not know the existence of the coordinator and are not aware that they are involved in a business process. So they only need to provide their own parts of functionalities.

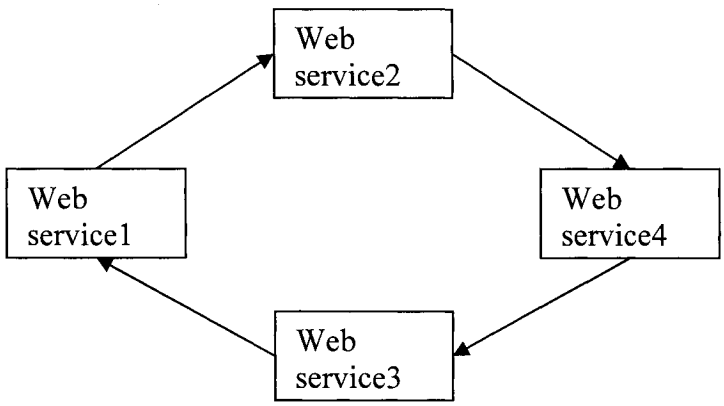

Figure 13: Choreography invocation mode

On the contrary, in choreography mode as presented in Figure 13, there is no centralized orchestration mechanism. The whole business process disperses among participating services. As showed in the Figure 13, each web service "knows exactly when to execute its operations and whom to interact with" [Juric2004]. We already noticed from section 2.3, this invocation mode is used in [Raut2003].

In this thesis, we propose a centralized orchestration mode for business process implementation based on the reasons below:

- A centralized business process engine is easy to be maintained. Any change of business process only requires change in one place

- Each participating web service does not need to add process handling ability. It gives better reusability and less customization effort for participating web services.

- Orchestration mode provides more flexibility to alternative business scenario and exception handling since we can specify them in the central process engine.

3.2.2 Invoking Web service

How a business process engine invokes other functionalities provided by dispersed services is an important consideration when designing a business process.

Synchronous web service invocation is used a lot in most of internal API call and communication between modules.

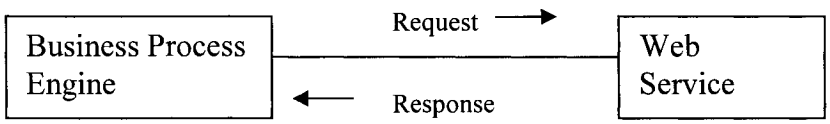

Figure 14: Synchronous Invocation Mode

In Synchronous invocation mode as presented in Figure 14, the invoking process sends a request to service provider and waits for its response. The whole process is blocked until the invoking process gets responses from the service provider. Due to the characteristics of different services, it takes a long time, sometimes, to fulfill the request at the service provider side. For example, in our health care data access approval process, the collaboration service is a manual process which requires data requestor and privacy officer negotiating through online communication. Therefore, synchronous invocation is not always suitable.

If the invocation is in asynchronous mode, the invoking process will not block its process execution and wait for the response. After the service provider finishes processing the request, it performs a callback to the invoking process. Basically, under asynchronous invocation mode, both parties act as a requestor and a provider. When orchestrating a business process, the participating services are loosely coupled and process engine does not have the ability to control the qualities, response time for example, of these external services. Therefore, asynchronous invocation mode can help create a more robust and better performed business process. Figure 15 illustrates the asynchronous invocation mode.

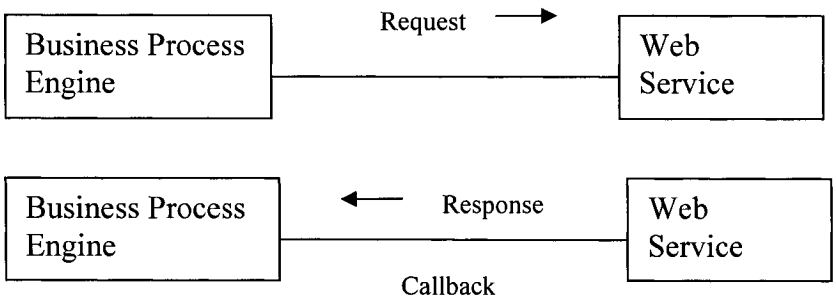

Figure 15: Asynchronous Invocation Mode

3.2.3 Business Process Execution Language (BPEL)

BPEL is a language which can address the above requirements of web service orchestration and asynchronous invocation. Besides, it provides easy ways, we will introduce through an example later, to define business process. Therefore, we choose BPEL to implement the business process engine in the business process layer.

BPEL is an OASIS (Organization for the Advancement of Structured Information Standards) standard and targeting to be used to write business process. It is created as the result of the mergence of two major business process languages, WSFL (Web Services Flow Language) from IBM and XLANG[16] from Microsoft. BPEL uses XML to describe business process.

The business process written in BPEL orchestrates all the participating functionalities which are provided by heterogeneous applications as web services. In the BPEL process definition, we first specify all the web services needed to be invoked. Then we define the sequence of the web service invocation according to the workflow of the business process. Since BPEL is a high level language which is focusing building business process, it provides easy way to support many traditional business process activities such as asynchronous invocation, compensation, concurrence and so on. The process design tools can use the predefined BPEL activities to realize these business process activities without caring about how to implement them at lower level, programming level for example.

[16] A business language which is an extension of WSDL (Web Service Description Language) in order to describe the business process.

The comprehensive introduction of BPEL is beyond the scope of this thesis but it can be found in [Juric2004]. In this section, a simple BPEL process is given in order to illustrate some basic BPEL activities which will be used in Chapter 4.

The simple BPEL process is a loan application system which is presented in [Lehmann2004] as an example for the BPEL introduction. Due to the limitation of the thesis space, we only extract the key steps, which is important to understand the basic BPEL process concept, of the original BPEL process.

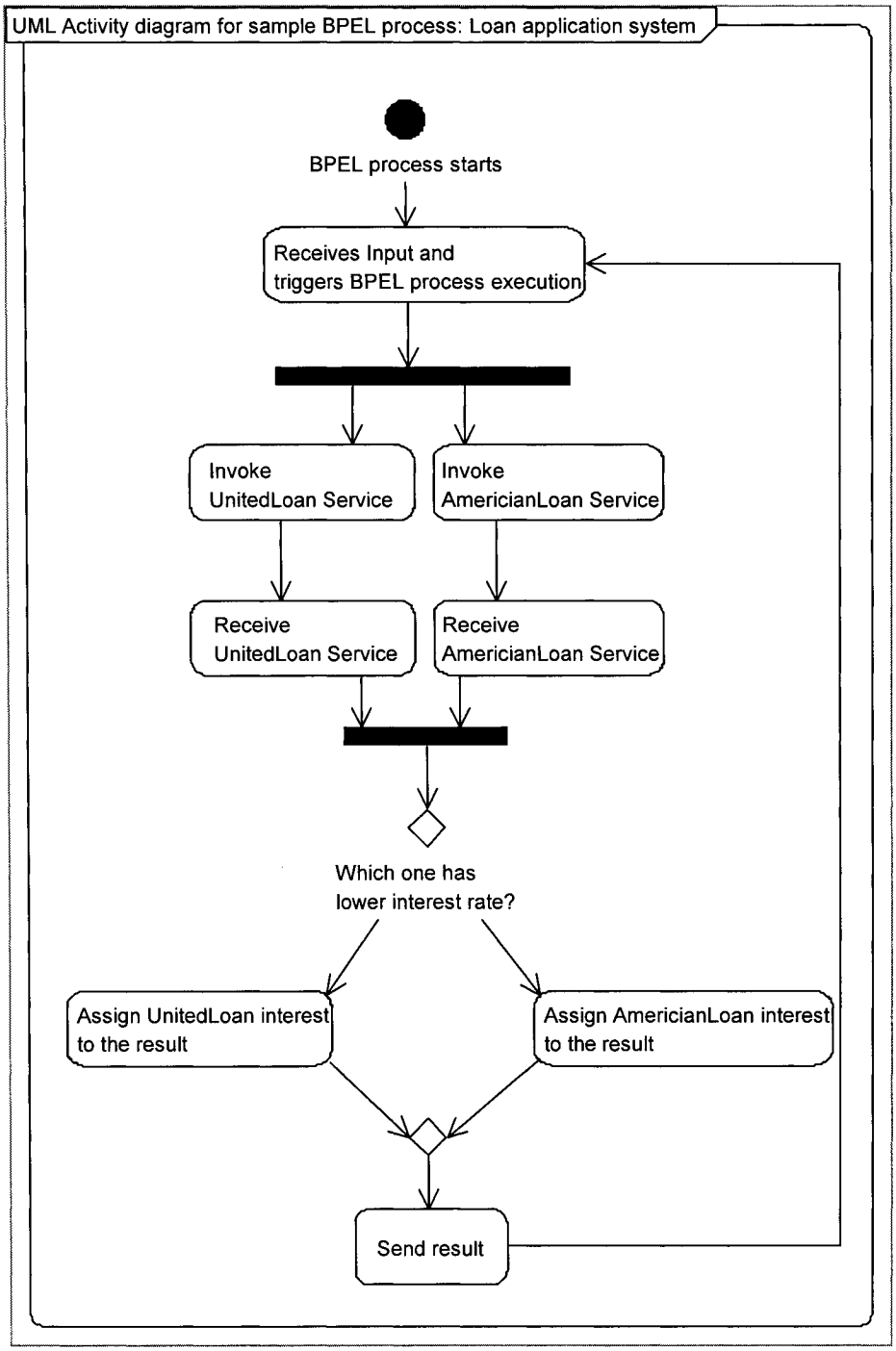

Figure 16: Process workflow of the sample Loan application

As described in Figure 16, the loan application has a simple process: It first accepts a loan application from a customer. Then it asks two bank loan services, UnitedLoanService and

AmericianLoanService, to provide quotes. Based on the comparison of the two quotes, it sends the lower interest quote to the customer as its result.

The BPEL process has five key parts as introduced below:

a. Partnerlink and Variable declaration

The first part of the simple BPEL process is the declaration of partnerlinks and variables.

BPEL partnerlinks are used to declare remote web services in order to invoke them later. In the below BPEL syntax, the first two partnerlinks, "UnitedLoanService" and "AmericaLoanService", are defined to invoke the two loan web services provided by the two banks.

```
...
<partnerLinks>
  <partnerLink name = "UnitedLoanService"
          partnerLinkType = "services:LoanService"
          myRole = "LoanServiceRequester"
          partnerRole = "LoanServiceProvider"/>
  <partnerLink name = "AmericanLoanService"
          partnerLinkType = "services:LoanService"
          myRole = "LoanServiceRequester"
          partnerRole = "LoanServiceProvider"/>
      <partnerLink name="client"
        partnerLinkType="tns:LoanFlow"
        partnerRole="LoanFlowRequester"
        myRole="LoanFlowProvider"/>
</partnerLinks>
...
```

The partnerlink "name" is used as the reference to invoke the corresponding web service in later BPEL syntax. The "partnerLinkType" refers to the definition in the WSDL document of the web service provider. Because the "partnerLinkType" is beyond the scope of BPEL, we can simply regard it as the binding to the actual web service provider. The "myRole" and "partnerRole" define the relationship between BPEL process and invoked web service. As

defined in this example, this BPEL process is a "LoanServiceRequester" and the invoked web services are "LoanServiceProvider".

The third partnerlink defines client, a web page accepting user input for example which kicks off the execution of the BPEL process, as a partnerlink. This is the way BPEL implements the asynchronous invocation. In this example, the loan application may be a long run process. Therefore, it is implemented as an asynchronous BPEL process which does not request the client application waiting for the execution of the BPEL process. When the BPEL process finishes its execution, it can invoke this client partnerlink and send back its result.

Like other languages, variables are used to temporally store some information in the BPEL process such as process states, communication messages

```
...
<variables>
      <variable name="input"
          messageType="tns:LoanFlowRequestMessage"/>
   <variable name = "loanApplication"
         messageType = "services:LoanServiceRequestMessage"/>
   <variable name = "loanOffer1"
         messageType = "services:LoanServiceResultMessage"/>
   <variable name = "loanOffer2"
         messageType = "services:LoanServiceResultMessage"/>
   <variable name="selectedLoanOffer"
          messageType="tns:LoanFlowResultMessage"/>
</variables>
...
```

In above BPEL syntax, several variables are declared. The "input" is declared as the input of the whole BPEL process. "loanApplication" keeps the request message from client input. "loanOffer1" and "loanOffer2" are used to keep the response information from the invocation of two invoked bank loan web services. Finally, the "selectedLoanOffer" is for storing the final loan interest rate which is the lower rate of loanoffer1 and loanoffer2. The "selectdLoanOffer" is sent back to customer as the result of this loan application process.

b. Kicking off the BPEL process workflow

```
...
<receive name="receiveInput"
                partnerLink="client"
        portType="tns:LoanFlow"
        operation="initiate"
              variable="input"
        createInstance="yes"/>
...
```

In above BPEL syntax, the "receive" activity receives a variable "input" from "client" partnerlink as the kicking off the BPEL process. The "createInstance='yes' " tells the BPEL engine to create a process instance in its runtime environment.

c. Invoking web services:

```
...
<flow>
    <!-- Invoke 1st loan provider in parallel to the second -->
    <sequence>
     <invoke name="invokeUnitedLoan"
          partnerLink="UnitedLoanService"
          portType="services:LoanService"
          operation="initiate"
          inputVariable="loanApplication"/>
     <receive name="receive_invokeUnitedLoan"
          partnerLink="UnitedLoanService"
          portType="services:LoanServiceCallback"
          operation="onResult"
          variable="loanOffer1"/>
    </sequence>
    <sequence>
    <!-- Invoke the 2nd loan provider in parallel to the first -->
    <invoke name="invokeAmericanLoan"
                  partnerLink="AmericanLoanService"
          portType="services:LoanService"
```

```
            operation="initiate"
            inputVariable="loanApplication"/>
        <receive name="receive_invokeAmericanLoan"
            partnerLink="AmericanLoanService"
            portType="services:LoanServiceCallback"
            operation="onResult"
            variable="loanOffer2"/>
    </sequence>
</flow>
...
```

In above BPEL syntax, it contains the workflow of the web service invocations. The "flow" activity represents the two "sequence" inside it are executed in parallel. While each "sequence" activity defines the process execution order inside it is sequential. In this case, the BPEL process invokes two bank loan services at the same time. For each asynchronous invocation, it first sends request then receives the response after the request is fulfilled.

As we mentioned before, the "invoke" and "receive" activity is the BPEL syntax to asynchronously invoke a web service. The partnerlinks declared previously, UnitedLoanService and AmericaLoanService, are used to bind the web service providers. Moreover, "operation" binds the corresponding interface provided by the invoked web service. Then, variable "loanApplication" is used as the input parameter of the invocation while "loanOffer1" and "loanOffer2" are used as the output parameters of service response.

d. Applying business logic/rules

Below BPEL syntax makes the decision on the final loan interest, which is sent back to client application, based on the comparison of two service invocation results.

```
...
<switch>
  <!-- If loanOffer1 is greater (worse) than loanOffer2 -->
  <case condition =      "bpws:getVariableData('loanOffer1','payload','/auto:loanOffer/auto:APR') >
bpws:getVariableData('loanOffer2','payload','/auto:loanOffer/auto:APR') ">
  <!-- Then take loanOffer2 -->
                    <assign>
                      <copy>
```

```
                <from variable="loanOffer2" part="payload"/>
                <to variable="selectedLoanOffer" part="payload"/>
                </copy>
              </assign>
            </case>
              <!-- Otherwise take loanOffer1 -->
              <otherwise>
                <assign>
                  <copy>
                  <from variable="loanOffer1" part="payload"/>
                  <to variable="selectedLoanOffer" part="payload"/>
                  </copy>
                </assign>
              </otherwise>
            </switch>
...
```

The BPEL "switch-case-otherwise" activity is same as the "if-then-else" syntax of other programming languages to implement the logic judgment. In this example, the lower interest rate result is selected as the final result sent to the client.

We also noticed from above BPEL syntax, there are some BPEL activities used to process variables. The "bpws:getVariableData" is a BPEL defined function which can access part of a BPEL variable, a WSDL message type for example, through XPath. The "assign" activity "copy" one part of the source variable to the part of a destination variable.

e. Sending return

```
...
<invoke name="replyOutput"
        partnerLink="client"
        portType="tns:LoanFlowCallback"
        operation="onResult"
        inputVariable="selectedLoanOffer"/>
...
```

Finally, after the loan application process finishes its workflow, it invokes the "client" partnerlink callback portType, a BPEL mechanism to support asynchronous invocation. This invocation sends "selectedLoadOffer" as the output parameter, the input parameter in terms of client callback service, to the client as the result of the BPEL process.

3.3 Message Exchange

In the previous section, we choose the BPEL to implement the business process engine. The BPEL implements the process workflow through invoking the web services provided by participating application functionalities. In this section, we need to define the message exchange format between the BPEL process and these web services.

Since all the participating services are loosely coupled and heterogeneous, it is difficult to implement a platform specific means for message exchange. For example, a Java object which represents the exchanging content can not be directly sent to a service which is written in Microsoft Visual Basic because this JAVA object is only meaningful inside a JAVA implemented environment. Therefore, we need a more generic and platform neutral means for our message exchange.

3.3.1 XML document

We choose XML document as the media for the message exchange between BPEL business process engine and participating services. There are several advantages regarding using XML document for message exchanging [Singh2004]:

- At first, the XML document is a text based standard format and various platforms now provide XML processing abilities. So, it gives better interoperability when integrating heterogeneous applications. And by using the XML processing tools, programmers do not need to handle low level message processing and can concentrate on implementing business logic.
- Second, XML document is well organized and self described by its tree like structure and customizable tags. Therefore, it is very suitable to represent data model.

- Then, the documents need to be exchanged between services may be very large. In later case study example, the message includes the information about the data access requestor's identity, requested data fields, the intended usage for the data and so on. Therefore, it is better to send it separately from the SOAP message as a XML document format.
- Also, a XML document provides a consistently legal binding for all participating services. The original document and each comment or change made by business partners can be kept for later reference.

3.3.2 XML Schema

After we decided using XML document as the message exchange media, it is important to define a document structure in order that each participating service can parse the XML document correctly. We choose XML Schema to define our XML document data structure.

XML Schema is a W3C recommendation to define the structure, content and semantics of XML documents [W3C-XMLSchema].

Elements and attributes: First, we use XML Schema to define the permitted elements and attributes in our XML documents. This step is important because it tells parsing application from where to get the wanted information. One issue during this step is whether to model a data item as a sub-element or as an attribute of an existing element. Our choice is based on the pattern in [Bodoff2004]:

The data item is modeled as element if it contains substructure, multiple lines and possibly occurs multiple times. For example, in the sample XML documents for data access process, the ***field*** item, which represents the data fields the requestor want to access, is modeled as element, because the requestor may ask for more than one field in a single request which is represented as a XML document.

On the other hand, the data item is modeled as attribute if the item is not intended to be visible for user or it is about a characteristic of the contents. For example, the data item

last_modified_time, which is automatically generated by application and used for internal management, is modeled as an attribute.

Structure: Secondly, XML Schema is used to define the structure of the XML documents. For instance, in the sample XML document for data access process, we specified a tree structure of XML documents that divides the document into several sections and each section contains several sub-sections. Some of these elements are compulsory while others are optional. For example, the ***field*** element is compulsory because it is meaningless if the request document does not include any data field information. The ***result*** element is optional inside a user input document since it should be filled out by corresponding application during the process execution. Once the data structure of XML documents is defined, the parsing application can use XPath to navigate the document tree and get specific information or transform the whole tree structure as a DOM[17] (Document Object Model) object for programming process.

Data type: Last but not the least usage of XML Schema in our solution is constraining data types. Even an element/attribute can be located according to the element/attribute name and tree structure, parsing application still needs to know the data type of the element/attribute in order to map to correct programming data type for later processing. Furthermore, for some data items, we also want to apply more restrictions on their permitted values. Such as ***purpose*** item, which specifies the data access purpose, instead of allowing user randomly input any string information, we can use the enumeration function of XML Schema to pre-define permitted values for this item. By doing this, we can guarantee the values can be understood by all the participating services.

3.3.3 Schema transformation

Until now, we have a consistent global XML documents and it is regulated by XML Schema, therefore all the participating services can understand it. On the other hand, due to their heterogeneous characteristics, each service may have its internal or local data model by

[17] A programming approach to map XML structure as a tree structure

which it applies its own business logic. Therefore, there is a need to map global XML Schema to application's local data model. We adopt the "meet-in-the-middle" approach as described in [Singh2004]:

We introduce a transformation layer between the application's local data model and the global XML Schemas. XSLT (Extensible Stylesheet Language Transformation)[18] is used to transform one schema format to another. This leaves room for the application to support additional schemas and most important, if the evolution happens at one side, either at global XML Schema side or local data model side, we only need to fine tune the mapping rules of XSLT and the other side of the application can be kept unchanged.

3.4 User Interface

3.4.1 Web based user interface

In the previous sections, we introduced using BPEL to implement the business process layer and defined the message exchange format among the process engine and participating services. As the framework architecture defined in section 3.1, we definitely need a user interface to accept user input then send it to BPEL process engine as the input for the process execution.

We choose web based forms to implement the user interface. Compared with stand alone rich client[19] software, web application has the advantages of easy client side maintenance, zero installation requirement, and wide accessibility from various devices.

3.4.2 Limitation of HTML web forms

One traditional implementation of web application is using HTML web forms to accept user input. Programmer defines name/value pair inside HTML file like below

[18] A XML based language which is used to transform XML document to other XML documents or HTML etc.

[19] The client side in a client-server architecture which performs intensive data processing, rendering user interface and so on. Rich client usually requires installing software component at client side

```
<input type="radio" name="car" value="1"/> 1 car<br/>
<input type="radio" name="color" value="2"/> Blue <br/>
```

At the server side, the JSP or Servlet[20] code gets those name/value pairs from HTTP request, then assigns to programming variables in order to apply business logic. There are some drawbacks within this approach as Dubinko presented in [Dubinko2003]:

At first, there is no an overview for the data structure of web form. The whole user input information is dispersed within individual name/value pairs. For our business process purpose, we need all the information sent as a single XML document as the input of BPEL business process engine. Therefore, a manually assembling of these name/value pairs into a XML document is inevitable if we use HTML form to accept user input.

Second, in order to verify user input or do some simple calculation, HTML form relies on server side functionalities or client side scripts languages. The former solution increases network traffic and delays response. The latter solution relies on scripts languages such as JavaScript and VBScript[21], but different browsers may execute script languages slightly differently and it causes compatibility problems.

Besides, HTML form mixes the data part of the form with its visual property. The name/value pairs of HTML form are very tied up with the visual layout. From above HTML syntax, the definition of a data named "car" and the specification of its visual property, radio button, are in the same HTML tag <input>. However, the data part of the web form is decided by the data structure, which represents the business model of the targeting system, of the back end system, it should not be tightly tied with its visual representation in the interface. During the implementation of the web form, the data part of the design should be handled by programmers, data modeler specifically, while how to visualize the data should be decided by web designers who are more focusing on rendering graphic interface. Since the data and its visual property are mixed together in HTML form, it is not easy to separate these two tasks for different persons.

[20] Technologies defined in J2EE standard to dynamically generate web page content
[21] Script languages embed in the HTML pages to perform tasks like user input validations, popping up windows or animations and so on

Because of above reasons, in this thesis, we use XForms to implement our user interface.

3.4.3 XForms web forms

XForms is a W3C recommendation for the next generation of web form. The main reason we introduce XForms to implement our user interface is its complete separation what the form does from how the form looks [W3C-XForms]. It defines the data structure, which represents the business model, part of the form separately from the definition of its visual presentation part.

Two major parts included in XForms realize the separation of the data and its visual representation. The model part defines the form data structure. It can include a XML data instance which gives a sample data structure and initializes the data with default values. The other part is the visual representation of the form. It defines different controls such as text input field, radio button, dropdown list and so on. The visual properties of the controls like color, font, layout are also specified here. Finally, an XPath reference is used to bind data to its visual representation.

XForms itself is written in XML format and embedded inside other web languages like XHTML or SVG. When the user finishes filling the form information and click "submit" button, the whole form data structure which is defined inside XForms model part is sent as a XML document, instead of the name/value pairs as in HTML forms, to the server. As we described in previous sections, all the participating services and BPEL business process use XML documents as inputs and outputs. So there is no middle component needed to assemble name/value pairs into required XML documents.

Another important functionality of XForms is its ability for user side validation and simple calculation. Inside the model part of XForms, we specified a XML Schema for the XML data structure. Therefore, XForms enabled browser will check user input against XML Schema and report any mismatches. Besides, also in model part, we can define the properties of any data item through XForms binding such as calculation, relevant, constrain and so on.

Figure 17 is one of the typical usages of how to use XForms and XML to model the user interface.

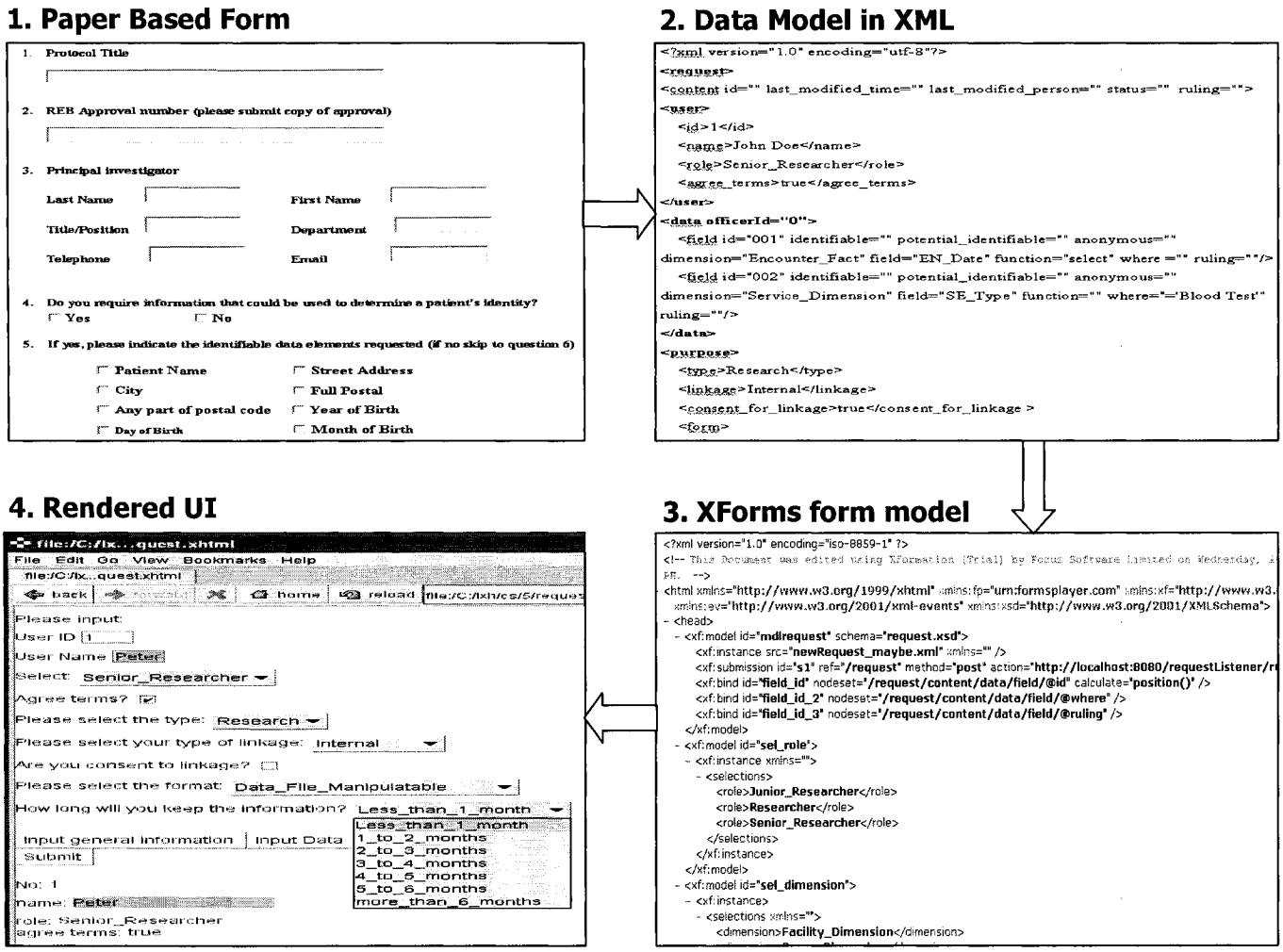

Figure 17 Using XForms and XML to generate User Interface

Step 1: At first, normally the building of the user interface starts from an existing paper based form which is used in current day to day business process.

Step 2: Then the original paper based form is modeled as an XML data model using XML related technologies and methods as we presented in section 3.3.1 and 3.3.2. Each input field in the paper form is mapped to element or attribute in the XML data model. For example, the

Firstname and Lastname fields on the paper form as in Figure 17 are mapped into an single element called "name" in the XML model with the XPATH "/request/user/name".

Step 3: After that, one models the user interface using XForms web form standard definition. The XForms web form includes the XML data model which is crated in step 2 and binds the element and attributes inside it to different predefined XForms controls. For example, the "name" element in the XML model is bond to an input field XForms control. The XForms web form definition is kept also as an XML file.

Step 4: Finally, an XForms enabled web browser renders the user interface. This browser takes care of how to generate the interface according to what has been modeled in the Forms web form XML files.

4 CASE STUDY

In this Chapter, we evaluate the processes of building a sample software system using three different approaches. These three approaches are two traditional software development approaches, the simple web application approach (SWA) and the simple SOA approach (SSOA), and the Model-based SOA approach that we presented in Chapter 3.

The sample system is an enterprise process for sensitive data access. This data access process is originated from the Ottawa Hospital data warehouse data request approval process (Ottawa Hospital Approval Process). The original Ottawa Hospital Approval Process is a paper based offline approval process. We applied three different development approaches to construct the electronic based online systems in order to implement the same offline tasks and flows, and at the same time, to streamline the approval process execution and shorten the waiting time to fulfill the data request.

4.1 The introduction of the Ottawa Hospital Approval Process

The sample process originated from the paper based data request process at Ottawa Hospital. There is a central data warehouse in the hospital and it stores all the sensitive health care information. The data is collected from different sources such as patient demographic data, patient visit data, billing data, laboratory results and so on. The accumulated historic data in the data warehouse is very useful for research purposes. At the same time, because the data is highly sensitive, Ottawa Hospital applies strict access approval process to safeguard it.

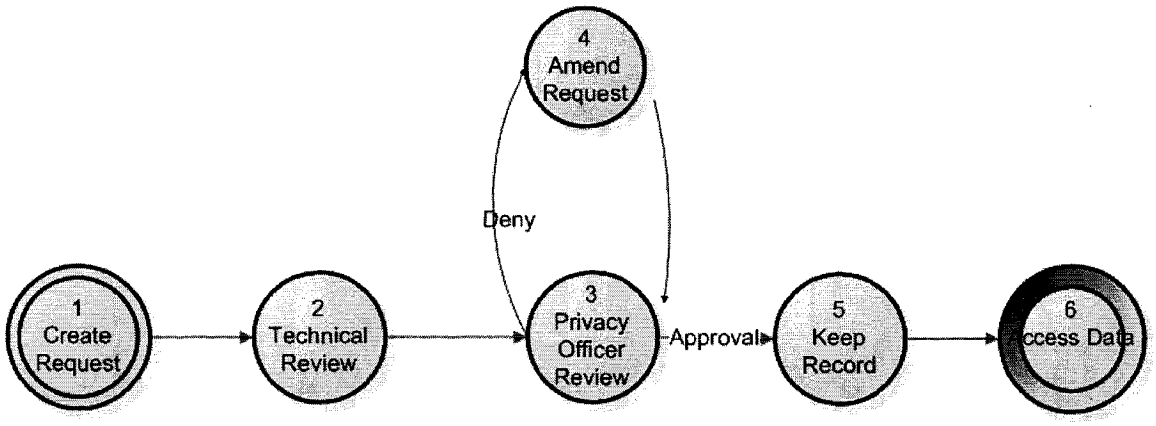

Figure 18: The workflow of the Ottawa Hospital Approval Process

Figure 18 is the workflow of Ottawa Hospital data request process. There are 5 steps within this data request process.

Step 1 (Create Request): If a researcher wants to access the data, he has to fill out a paper form. Inside the form, he needs to specify his identity, the detail data fields he wants to access, the reason why he needs to access the data, and what the safeguard methods he will use to protect the data from being abused. The sample paper request form is provided in appendix 1 for reference.

Step 2 (Technical Review): After the researcher fills out the request form, it is first submitted to the data warehouse administrator for technical review. The administrator verifies whether the request is feasible. For example, if the researcher asked for the data which is not stored inside data warehouse, this request is considered as infeasible. Also, the administrator checks the actual data fields to which researcher requested and specifies whether they can be technically used to identify patients' identities. Then, the request is forwarded to privacy officer for review.

Step 3 (Privacy Officer Review): The hospital privacy officer gets the request along with identity information and decides whether the data access will be permitted. The privacy officer's decision is based on hospital privacy policies and related legislations.

Step 4 (Amend Request): If the privacy officer denies the data access request, he sends the request form back to the researcher and lets him amend it. One possible reason for denying the initial request is that the privacy officer only allows part of requested data to be accessed. For example, he may allow access to patient's postal code but deny access to patient's street address since it can be used to locate the actual patient. Another reason for the rejection of the initial request is that the privacy officer requires the researcher to apply stricter safeguard methods. For example, he only allows the data access duration for 1 month instead of 3 months as researcher wanted. After the researcher changes his request details, he sends the amended form to the privacy officer for review again. The workflow will go back to step 3. If the amended request still gets denied, step 4 will be repeated, and then go back to step 3 for review again until the request finally gets approval from the privacy officer.

Step 5 (Keep Record): Before the researcher accesses the data, the details of his request information as well as the approval signature from the privacy officer will be kept as a record in an archive for a later audit trail.

Step 6 (Access Data): Finally, the request is forwarded to data warehouse administrator again who locates the actual data the researcher wanted. The data can be delivered to the researcher using different formats: hardcopy report, softcopy like excel spreadsheet, or even direct access to the data warehouse within a period of time.

Because the above data request process is paper based, it usually takes several days, or even weeks for the researcher to finally get access to the data. According to our observations made from interviews with some researchers, most of the delays are caused by the handling and processing of paper form. The actual decision making and data access operations do not require too much time. Therefore, we decided to introduce an electronic based online approval process to automate the original paper based request process and shorten the request processing time.

In the following sections, we will describe how we implemented the online approval process using different development approaches. In section 4.2, we will briefly demonstrate how we

used two traditional approaches to develop the online approval process. Then, in section 4.3, we will illustrate the details of applying the Model-based SOA approach as presented in Chapter 3 to implement this online approval process.

4.2 Traditional Development Approaches

4.2.1 The Single Web Application (SWA) Approach

At first, we used a single web application development approach to transfer the original paper based Ottawa Hospital Approval Process to an online approval process. The whole online system was built into one web application and the implementation details are given as described below.

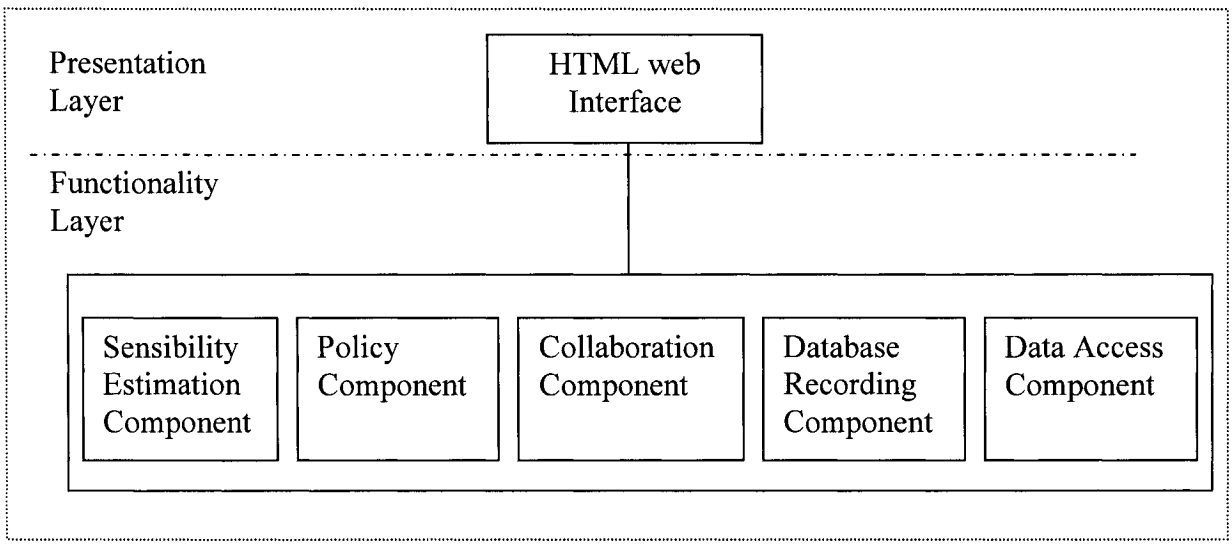

Figure 19: The framework architecture of the SWA approach

The framework architecture of the SWA approach is given as Figure 19. In this approach, there are two layers in its framework architecture: presentation layer and functionality layer. The presentation layer is implemented as traditional HTML forms which accept user input. The functionality layer implements all the process workflow and functionalities. In this layer, all the functionalities are built specifically for this data access system and they are tightly coupled as different components in one programming language environment. There is no a separate business process definition layer in the SWA approach. The process workflow is dispersed among these components and also written in programming languages, Java code

for example, same as all other functionalities. We built the sample system using the SWA approach because it is a typical implementation before introducing SOA concept.

The SWA approach can be used to easily implement some small scale applications because of its simple framework architecture. Since there are no extra layers and sophisticated technologies/concepts in the SWA approach, only regular application development skill is required. However, when the targeting application becomes complicated and needs to integrate the functionalities crossing different domains, like the situation in the sample online data access system, it is very hard to build all these functionalities and workflows from scratch. The detail comparisons of using the SWA approach with the other two approaches are given in Chapter 5.

The simple web application we implemented is an approach which is completely based on plain Java, JSP and Servlet technologies. We also acknowledge there are other frameworks available, such as Spring Framework, which are web based applications at the same time can also easily describe workflow. Due to the limitation of the implementation resources and thesis space, we did not adopt Spring Framework in the implementation of the simple web application neither provide more details about it. The detail information about Spring Framework can be found in [Raible2004].

4.2.2 The SSOA (Simple Service Oriented Architecture) Approach

Then, based on the simple SOA concept, we rebuilt the online data access process. The implementation details are given below.

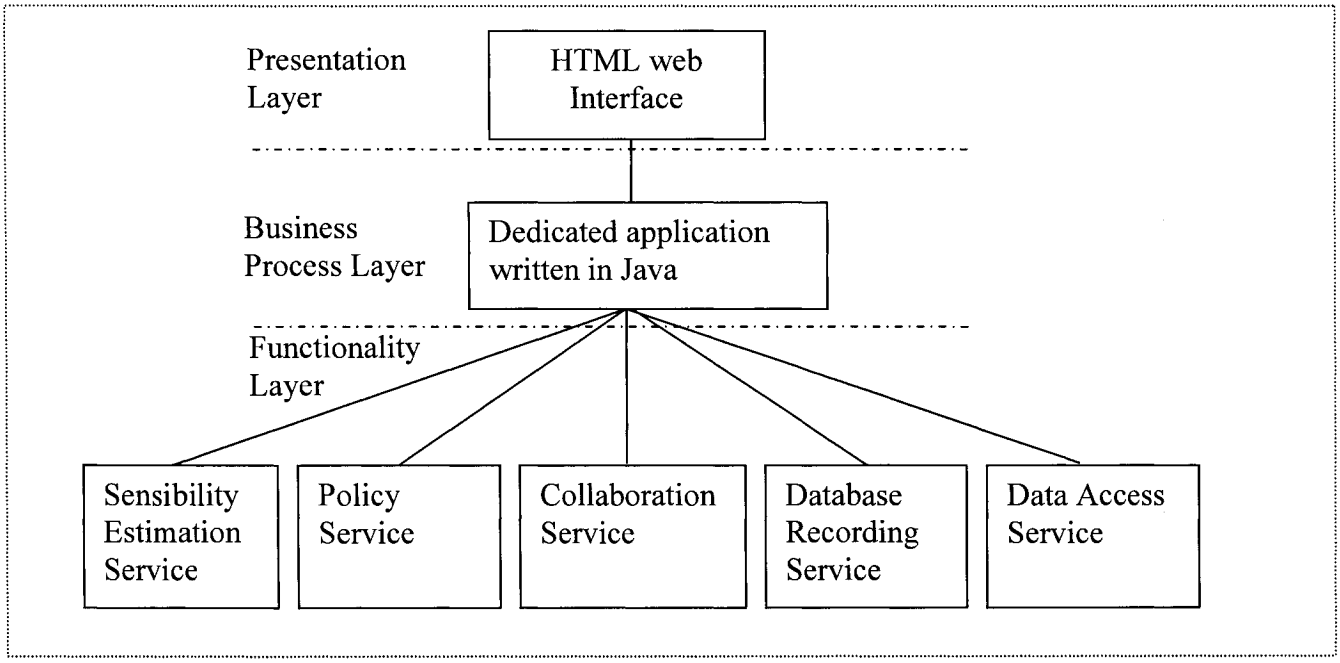

Figure 20: The framework architecture of the SSOA approach

As presented in Figure 20, the SSOA approach has a three layer framework architecture. The presentation layer uses traditional HTML forms to accept user input, the same way as the SWA approach. A separate business process layer is adopted in the SSOA approach to define the workflow of its business process. The functionality layer provides all the functionalities needed by the system. By leveraging the SOA concept, all the functionalities in functionality layer are provided as web services for business layer invocation. Therefore, SSOA approach does not require all the functionalities specially built for the online approval process. It allows reuse of existing applications even they are written in different programming languages. As a simple SOA implementation, however, the business process layer is implemented using programming languages, Java code for example. The invocations of web service functionalities, definitions of process workflow are coded by Java developers. Before the introduction of process descriptions/execution languages for defining/orchestrating business process, the process definitions coded in programming language is a typical SOA based simple implementation. Therefore, we also adopted the SSOA approach to rebuild the sample online data access system in order for later comparisons.

The SSOA approach is based on SOA concept. Because each major functionality is provided as a service, it is easy to scale the application and its business process to accommodate more

complicated system. However, the hard coded business process does require the programming skills to develop and maintain. The detail comparisons of using the SSOA and other two approaches are given in Chapter 5.

As well, the functional layer definition is common between the SSOA approach and the Model-based SOA approached described next in Section 4.3. Please see section 4.3.1 Functionality Definition for a detailed description of the services that were implemented for the Ottawa Hospital Approval Process.

4.3 Model-Based Service Oriented Architecture (SOA) approach

In Chapter 3, we presented a Model-based SOA software system development approach based on BPEL, XForms, SOA and XML standards/concepts. We use XForms for user interface models, BPEL for business process modeling, SOA to integrate all the participating functionalities and XML technologies, such as XML document, XML Schema and XSLT, for data modeling. In this section, we will illustrate the detail how we used the Model-based SOA approach to implement the sample online data access process.

In section 3.1, we defined a three-layer framework architecture for our Model-based SOA approach. Therefore, in the following sections we will use a bottom-up methodology to construct these three layers for the online data access process.

In section 4.3.1, we will define the functionality layer. Then in section 4.3.2, we will give the business process layer definition. Finally, in 4.3.3, we will introduce the presentation layer.

4.3.1 Functionality Layer Definition

The first step we have done when we transformed the paper based data request process to online process was defining what the software functionalities needed in the online process were.

According to the paper based steps as described in section 4.1, we abstracted the human activities in each of the step and assigned software functionality as its online counterpart. By

using this approach, we defined the major functionalities of the online process and each of these functionalities can be mapped to a step in the paper based process.

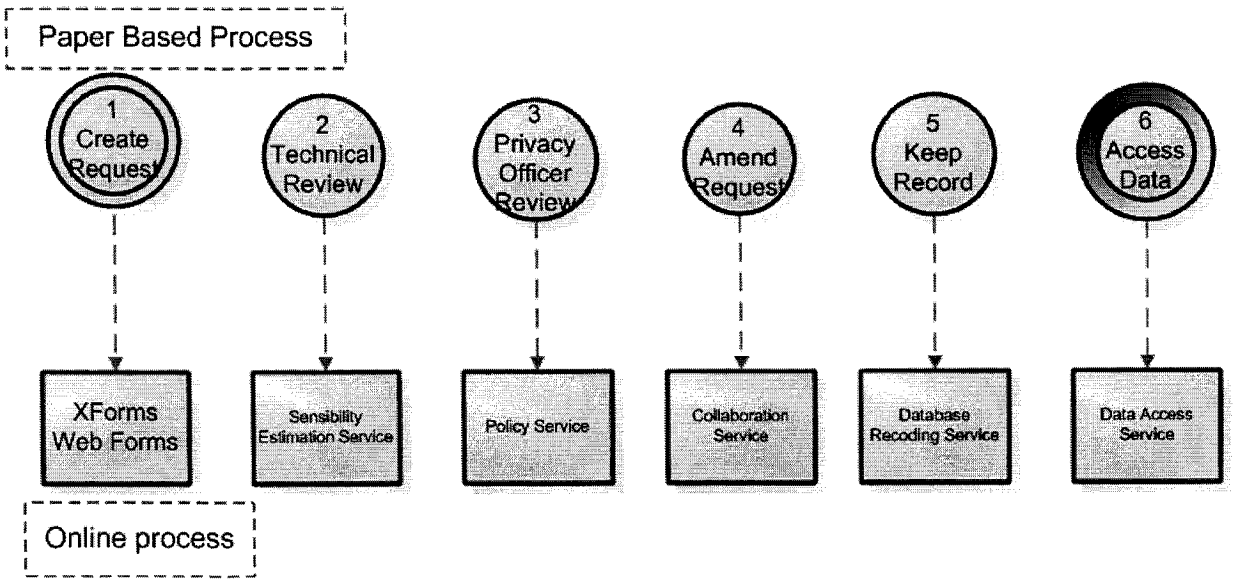

Figure 21: Functionality mapping

As in Figure 21, except the first "create request" to "XForms Web Forms" mapping which is implemented at the presentation layer (we will introduce this in section 4.3.3), the other five major human activities correspond to five major software functionalities.

As we will introduce in details in section 4.3.1.6, The "Policy Service" and "Collaboration Service" in Figure 21 already exist from other projects and we just reused their functionalities in the implementation. And the other service functionalities had been created in the implementation of the SWA approach and we also reused them in the implementation of this approach.

Below sections, 4.3.1.1 to 4.3.1.5, provide brief descriptions for each of the software functionalities.

4.3.1.1 Sensibility Estimation Service

Functioning as the technical review step in paper based process, sensibility estimation service estimates whether the fields which researcher asks can be used to identify patients' identities.

This service application maintains a list which keeps the sensibility of each data field. There are three possible sensitivities values: "identifiable", "potential identifiable" and "anonymous", which stand for different levels of sensitivities of the data field. For example, field "Street Address" has "identifiable" sensibility property since it can be directly used to identify patient's identity. While "admission date" is "potential identifiable". Although "admission date" itself is not "identifiable" because there are more than one patient visits in an "admission date", it can contribute to identify patient when connected with other information such as "patient symptom" and "laboratory information". Each organization has its own sensibility definition according to its privacy policies and related regulations.

The input of this service is the data fields in the request. The output of this service is the sensitivities of the data fields in XML document format.

4.3.1.2 Policy Service

Policy service implements the same functionalities as privacy officer review step in paper based process. The purpose of this service is to automatically check request against predefined policies and decide whether accepts access without human involving.

The policies are defined using EPAL. As we will introduced in 4.3.3, the request we got from XForms web interface has exact four sections which corresponds to four EPAL categories.

Policy A:

Ruling = DENY
User = senior researcher & agree to terms
Data = identifiable
Purpose = research & linkage to internal db & consent for linkage
Action = data file manipulation & less than one month & file destroy after

Figure 22: A simple EPAL policy definition

Figure 22 is a simple policy definition. The ruling part in the policy definition represents the verdict of this policy based on the combined information from the four categories.

The input of this service is the XML document format request which is get from user interface. And the actual data field of the original XML document request has been replaced with its sensibility properties which are get from sensibility estimation service. The reason we used data field's sensibility properties, such as "identifiable", "potential identifiable" and "anonymous", instead of the actual data field is that data field information often needs to be added or deleted depending on the availability of the data in data warehouse. Therefore, the policy also needs to be changed if it uses actual data field. On the other hand, the policy is used for decision making and it should not be so volatile. Therefore, we adopted data field's sensibility properties in policy definition which are relatively constant compared with data field.

The output of the policy service is the verdict of applicable policy or a "maybe" verdict if there is no applicable policy found.

4.3.1.3 Collaboration Service

Collaboration service is a web based application which provides an online interactive interface for researcher and privacy office to resolve the privacy related dispute.

The research is redirected to the collaboration service when his request can not be approved automatically by the policy service. In the collaboration service interface, which resembles an online chat system, research and policy officer can negotiate the request details using different technical means such as chatting, shared editing and browsing related regulations. After them both agree on the amended details of the request, they digitally sign on it using their own digital signatures for later audit purpose.

The input of the collaboration service is the original request in XML document format. The output is the amended request in the same XML format with digital signatures of researcher's and privacy officer's attached.

4.3.1.4 Database Recording Service

Database recording service stores the detail information of the request into the database for later audit and statistics purpose. For each request, database recording service assigns an unique request ID to it. Along with the request details, database recording service also keeps the verdict of the request and its current status. The verdict of a request can be "allowed", "denied" or "maybe" depending on the decision of the policy service. A request with "in process" status means this request still under the negotiation between the researcher and the policy officer. Therefore, even the current verdict of this request is "denied", the verdict may be changed to "allowed" after it finishes the collaboration service. The verdict is only treated as final and used to decide whether allow the data access when the request's status is "completed" which means it already finishes the collaboration process. Besides, all the digital signatures researcher and privacy officer signed, during they discussing the request using the collaboration service, are also stored in the database of this service.

The input of database recording service is the request in XML document format. The output is the generated request ID in XML document format.

4.3.1.5 Data Access Service

Data access service fulfills the last step in the access process --- access data. Because the actual healthcare data is stored in the Ottawa Hospital Data Warehouse, data access service provides the connection to data warehouse but hides the technical complexities of its communication to data warehouse.

Before the request is sent to data warehouse for query, the data access service needs to get the requested fields from the request and create a SQL (Structured Query Language)[22] query using its own query builder.

In order to secure the data access, data access service also keeps the researchers' and privacy officer's original signatures. It compares the digital signatures attached to the request with their original ones. If the signatures are different, it will deny the data access.

[22] A common database query language

The input of data access service is the request in XML document format and the output is the web link in XML format which is pointing to the web pages where the data requestor can finally get the requested data.

4.3.1.6 Functionalities integration methodology

When we implemented the above five major functionalities of our online data access process, we adopted SOA based development approach to create these functionalities as web services. This approach not only gave us lots of flexibilities in the creation of functionalities but enabled us to reuse some of the existing applications in stead of building them from scratch.

For example, collaboration service is a reuse of one existing online collaboration system. This system was developed by one of our teammates who built it for an electric learning system. Therefore, the system architecture and programming language are defined with the concerning to satisfy electric learning system requirement. But when we integrated this collaboration system as a service into our online data access process, she just provided a web service interface for her original collaboration system and rest of the modules were still unchanged. Also, the policy service is implemented by an open source EPAL engine written in Java. All the work another teammate has made is wrapping the original open source EPAL engine Java methods into web service methods.

Besides, since all these services are loosely coupled, the developing works which were assigned to different teammates did not depend on each other. Also, the internal changes within one service application did not impact on other services. So, our development works were implementing in parallel. Because there is no need of a central point to integrate and host all the functionalities at development stage, we did not setup a complicated central software and hardware environment during our development procedure in order to provide all the functionalities. The only information we needed to invoke a service is the web service's WSDL web address. As the result of this distributed development approach, it gave us great flexibilities to develop and integrate our functionalities. During our development, most of the integration testing was performed when participating developers were at different

locations. In this distributed integration testing, each developer only needs to be responsible for his/her part of application and provide the WSDL address to others for access.

4.3.2 Business Process Layer Definition

As presented in our framework architecture, after we built the functionality layer of the online data access process, then we need to define the business process layer on top of it. The BPEL process in the business process layer is used to orchestrate the services, which are provided from functionality layer, and implement the workflow.

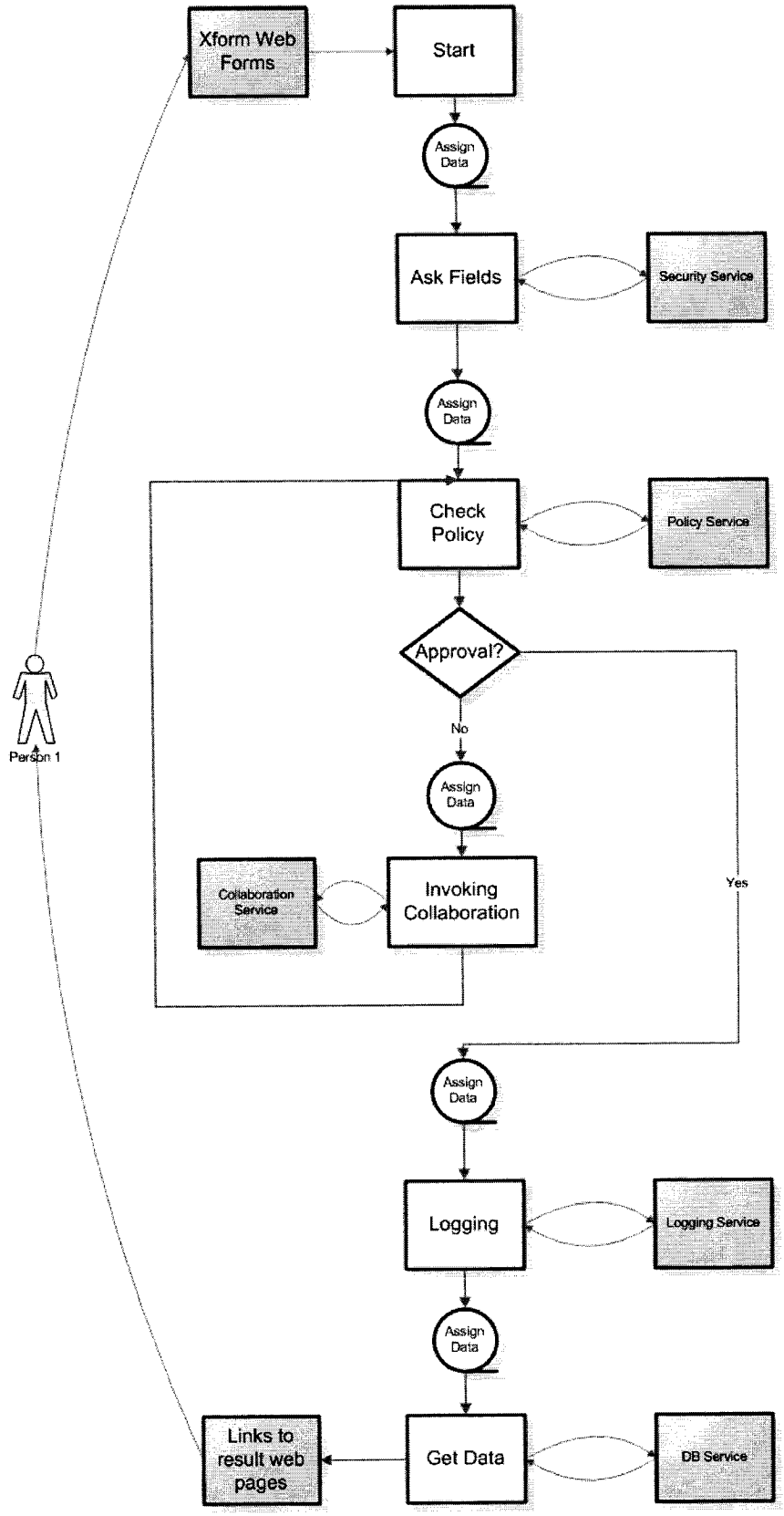

Figure 23: The process workflow of online data access approval process

Figure 23 shows the process workflow of the online data access system. This process is originated from the paper based data request process as presented in section 4.1. The workflow of the process is implemented as a BPEL business process, represented as the components in white color in the Figure 23. Then, we give some detail information about this BPEL business process.

4.3.2.1 BPEL business process

Since BPEL is a high level business language, we only need to define our process at business level and do not need to concern those issues at programming level such as generating web service local stub and so on. We highlight some of the key BPEL sections below. The whole XML format BPEL process definition is also attached in Appendix 3 for reference.

PartnerLinks and Invokes: There are 5 BPEL <partnerlink> activities defined in the BPEL process which are corresponding to 5 different major functionalities provided by separate applications as web services. The BPEL <invoke> activity is used to invoke each of the <partnerlink> activity. In Figure 23, these BPEL <invoke> activities are represented as 5 rectangle shapes named "Ask Fields", "Check Policy", "Invoking Collaboration", "Recording Request" and "Get Data" respectively. Also, each of these invocations has a corresponding external service as in Figure 23. These external services are the web services provided from functionality layer.

Flow Control: The workflow of our data request process is realized using several BPEL flow controls. The BPEL <sequence> activity is used to specify the order of the partner invocations. Fox example, "askfields" invocation is placed before the "checkpolicy" in the <sequence> activity. BPEL <switch> activity and <case> activity are used to execute different flows based on policy decision. In Figure 23, the diamond shape named "Approval" represents the <switch> activity.

Assigns and Variables: As we mentioned in section 3.3.3, there is a need for the transformation from global XML Schema to each service's local XML Schema. Therefore, we perform a message transformation before each service invocation. This transformation is

implemented using BPEL <assign> activity. In our case, we take an input XML document and apply a XSLT transformation. Then assign the result XML document as the input of the next invocation. The BPEL variables are used to temperately store message results. In Figure 23, the <assign> activity is represented as the circle named "Assign Data".

4.3.3 Presentation Layer definition

4.3.3.1 XForms Web Forms

According to our framework architecture, we then need to define the presentation layer and provide an interface to accept end user input in order to trigger the execution of BPEL process. Our presentation layer provides the user interface as web forms which include the same information as paper form.

The web forms are implemented using XForms. As introduced in section 3.4.3, there are two major parts inside the XForms form definition.

The model part defines the XML document data instance which is used to store request details. There are four sections in the XML data instance which are represented using four elements: user, data, purpose and action. **User** element includes the information about researcher himself/herself like name, role, whether agreed to certain terms and so on. **Data** element includes the actual data fields researcher wants to access. It may include more than one data fields because researcher often needs to access more than one field in a single request. **Purpose** element is used to store the reasons why researcher needs to access the data. **Action** element is the place to specify what kinds of safeguard means researcher will perform in order to protect data privacy.

A complete XML document for the request with Simple data filled is attached at Appendix 2.

The rest part of XForms form definitions describes the user interface of web forms, just like normal HTML does. XForms provides many predefined controls, such as input field, drop down list or check box and so on, to accept user input. Through XForms reference, it refers user input to the corresponding place as specified by XPath and keep it in the XML data instance.

```
<xform:input ref="/request/content/user/name"
model="mdlrequest">
   <xform:label>User Name</xform:label>
```

Figure 24: A simple XForms syntax

Figure 24 is an example of XForms reference which maps an input field, an XForms pre-defined user interface control, to the value of an element in XML data instance with the XPath /request/content/user/name.

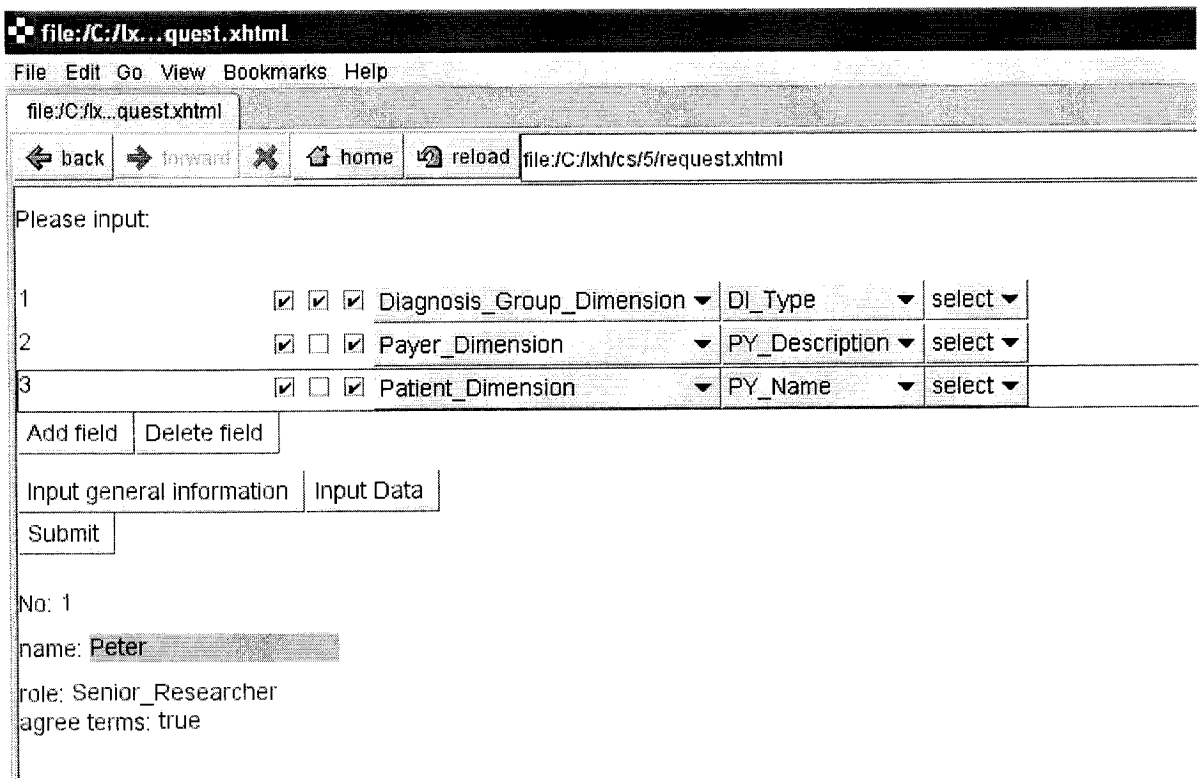

Figure 25: User interface 1

Figure 25 is a screenshot of the XForms web form user interface for inputting **data** fields of the request. Because one request may contain more than one data fields, the XForms repeat control is used to handle repeat XML elements in the data instance.

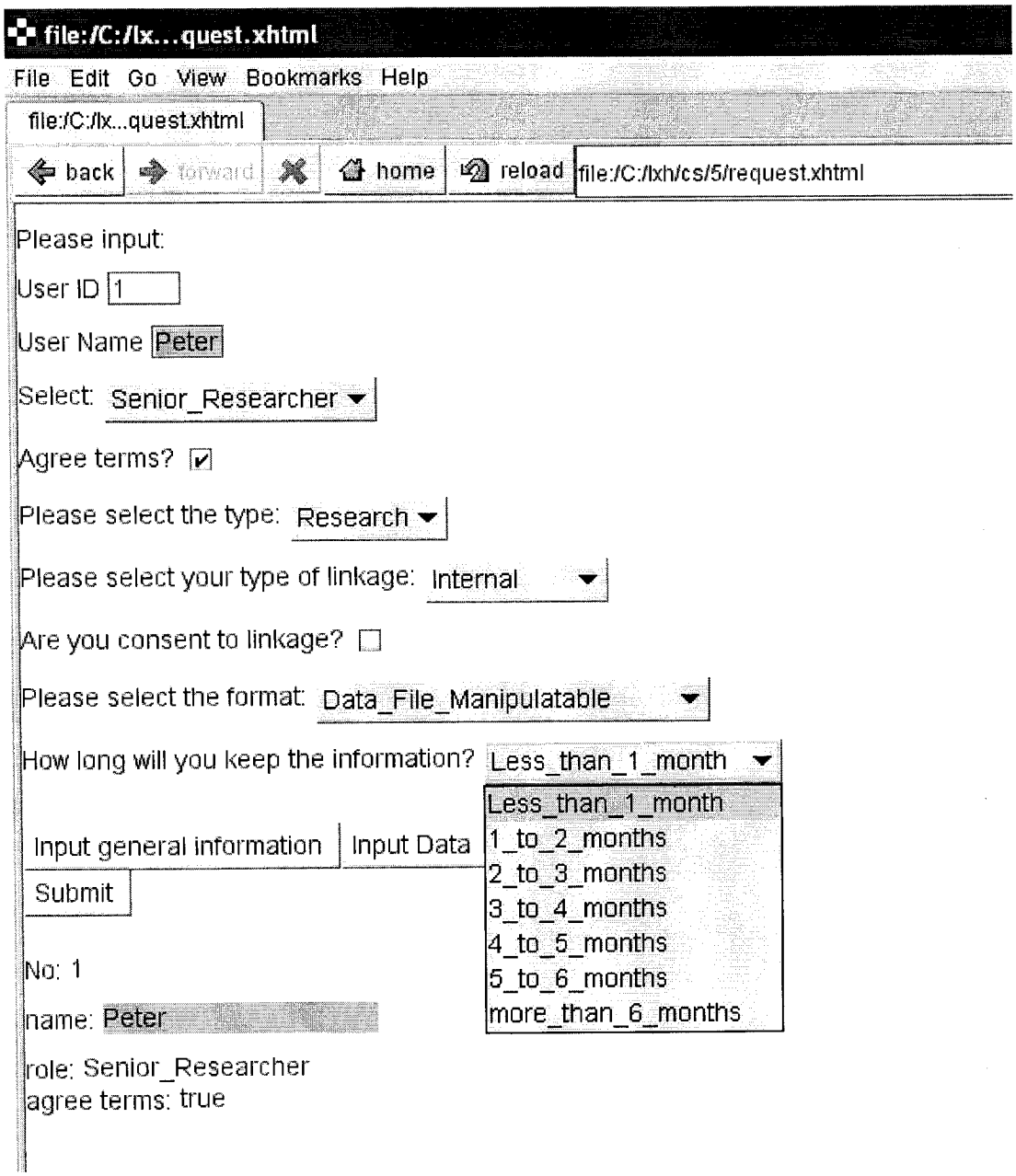

Figure 26: User interface 2

Figure 26 is another screenshot of the user interface for inputting the rest three parts (**users, purpose, action**) of the request. As we introduced in section 3.4.3, the XML data structure of the XForms web forms is defined by a XML Schema file. Any invalid user input, according to XML Schema definition, will be checked by XForms enabled browser without requiring writing extra script language. In Figure 26, the user input "Peter" for "user name"

field is invalid according to the definition in XML Schema, we only allow certain users login for example, and therefore it is highlighted for reminding user to change the input.

Once researcher fills the web forms, the XML document is sent to server and triggers the execution of the online data access process.

4.4 BPEL Business Process Development methodologies, tools and system architecture

The lifecycle of the development of BPEL business process is divided into three stages: Process Design, Process Run time environment and Process execution monitoring and debug. We will briefly introduce them in the following sections.

4.4.1 BPEL Process Design:

The tool we used to design BPEL process is Oracle BPEL Designer. It is a plug-in component running inside Eclipse[23]. As the screenshot shown in Figure 27, this software provides a visual modeling tool. The one can design and create BPEL process by simple drag and drop pre-defined BPEL components, such as "assign", "invoke", "switch" and so on, into the process map. The whole designing procedure does not require writing code.

Because BPEL is becoming a more and more popular business process language standard, there are many other BPEL designing tools available such as IBM WebSphere Studio Application Developer, Microsoft BizTalk Server Orchestration Designer, ActiveWebflow and more. These tools also provide the similar user friendly interface for process design and creation. The processes created by these tools are all compatible with BPEL standard definition.

[23] An open source integrated development environment. It is widely used to develop Java programs

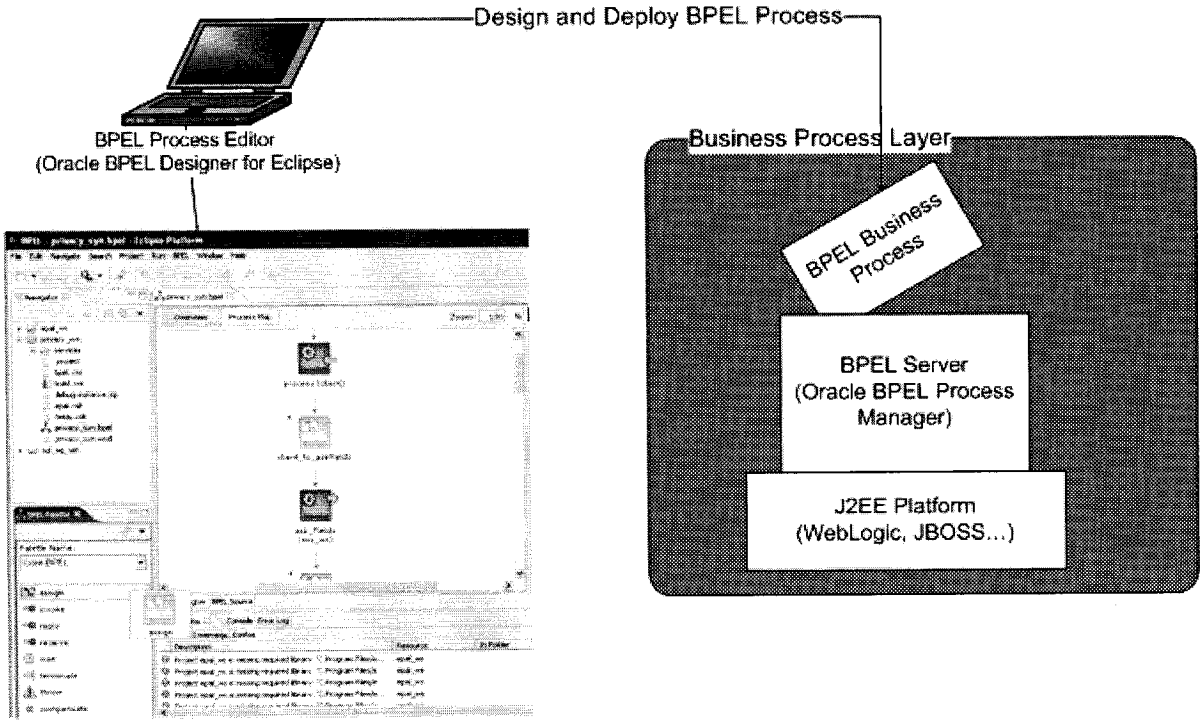

Figure 27: BPEL process development and deployment

After the BPEL process is created, the BPEL Designer can also be used to deploy the BPEL process to the BPEL server for execution. We will introduce the architecture of BPEL Server in the next section.

4.4.2 Online data access process runtime architecture/environment

4.4.2.1 System Architecture

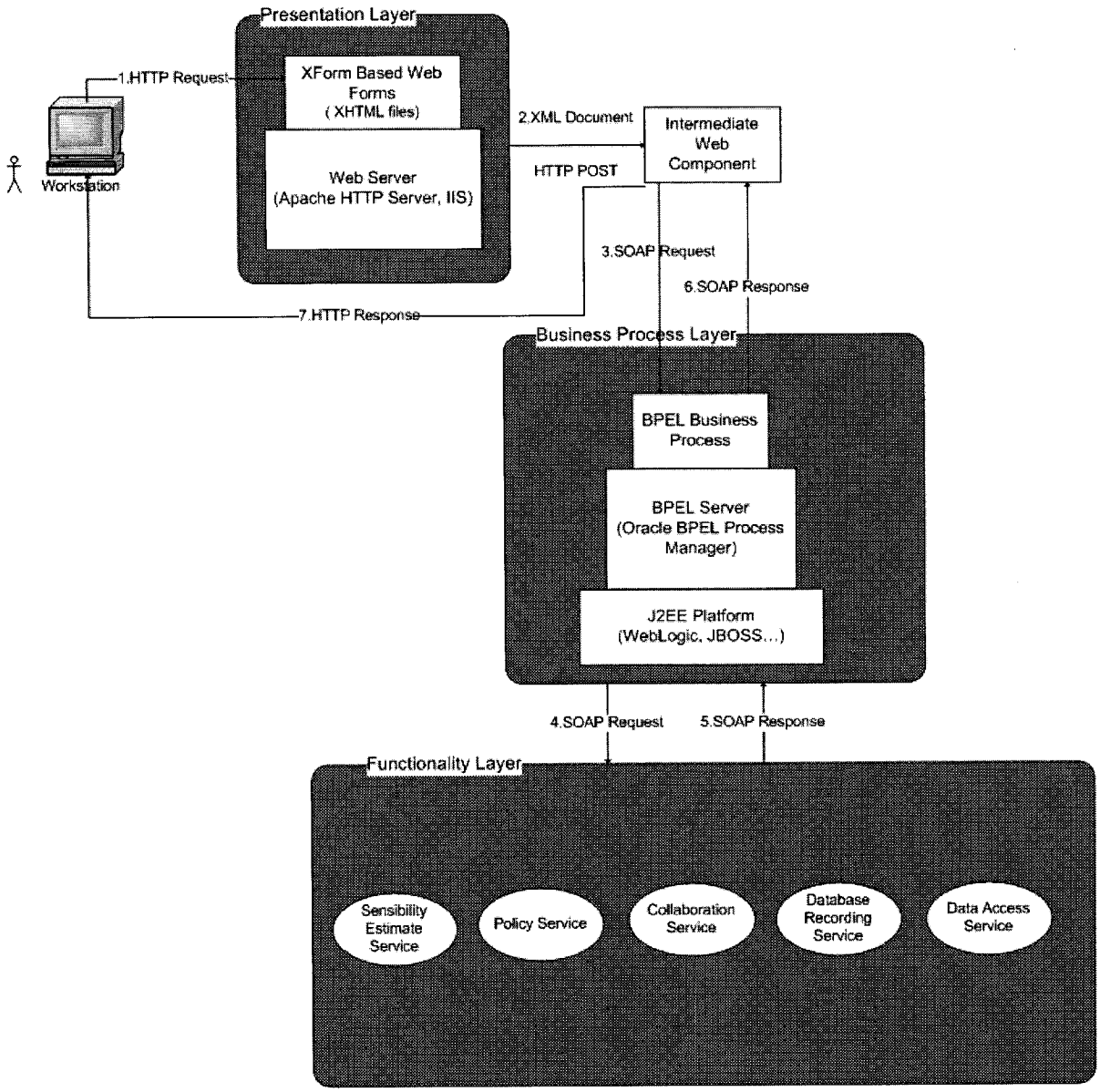

Figure 28: The runtime environment of the simple online data access approval process

In Figure 28, we present the three layer system architecture of the online data access process in the run time environment and its software configuration.

In Presentation Layer, the XForms based web forms are running inside a regular web server. These web forms are stored as XHTML pages and can be accessed from end users' browser.

In order to correctly render XForms web forms on end users' screen, the browser must be compatible with XForms standard such as X-Smiles, Deer Park or formsPlayer

In Business Process Layer, a standard J2EE (Java 2 Platform Enterprise Edition)[24] platform is at the bottom, such as Weblogic or JBOSS. On top of J2EE platform is the BPEL server which provides the runtime environment for BPEL process. BPEL server interprets BPEL process to standard J2EE components and manages/monitors the BPEL process execution. Above BPEL server, it is BPEL business process which is developed by the Oracle BPEL Designer.

In Functionality Layer, it provides all the major functionalities of the system. All the functionalities are provided as web services to BPEL process. Each functionality is a stand along and self contained application which does not depend on other functionalities. They may be hosted at different physical servers and software environments.

Intermediate Web Component: According to the BPEL standard, the BPEL process in the business process layer exposes its own functionality also as a web service for client invocation. Since the output of the presentation layer is a XML document contains user input, there is a need of an intermediate component which can accept the XML document from user inputs, then sends it as the input for BPEL process and triggers process execution. Also, it needs to get the response from BPEL process and forwards to the user browser. In Figure 28, the Intermediate Web Component is composed of Java JSPs and Servlets. It invokes BPEL process as a web service and sends BPEL process results to user browser.

4.4.2.2 A Simple Scenario

In this section, we present a simple scenario covers the steps as represented by the numbers in Figure 28. In this scenario, we assume the request gets approval. Therefore, it covers the steps from a researcher fills a web form till he finally gets the results.

[24] A Java specific software development environment for developing web based enterprise application

Step 1: A researcher opens a XForms compatible web browser, X-Smile for example, on his computer. The browser sends regular HTTP request to a web server where the XForms web forms are hosted and gets web forms rendered on the browser.

Step 2: After the researcher fills the form, the form content is sent as a XML document through HTTP POST action.

Step 3: The intermediate web component gets XML document and uses it as the input to make a web service invocation, SOAP request, to the BPEL business process.

Step 4 and Step 5: According to the workflow and business logic defined in the BPEL process, it makes web service invocations, SOAP requests, to several functionalities and gets their SOAP responses.

Step 6: BPEL process sends its result, a web link to the requested data, as SOAP response to intermediate web component.

Step 7: Intermediate web component gets the web link from the SOAP response message and redirect it as a regular web page containing the link to researcher. Researcher can click this link and gets the data.

4.4.3 BPEL process monitoring/debug UI

In section 4.3.2, we mentioned BEPL server can monitor the execution of BPEL process. In our implementation, the Oracle BPEL process manager provides us a visualized interface to monitor BPEL process execution. When we debugged the BPEL process, it gave us a graphic based web interface through which we can track each step during the execution of business process. If any error happens when executing the process, the problem step will be highlighted and the error message as well as the input/output message of all the preceding steps can be tracked in order to help troubleshooting. Figure 29 gives screenshot of the Oracle BPEL process monitoring/debug interface.

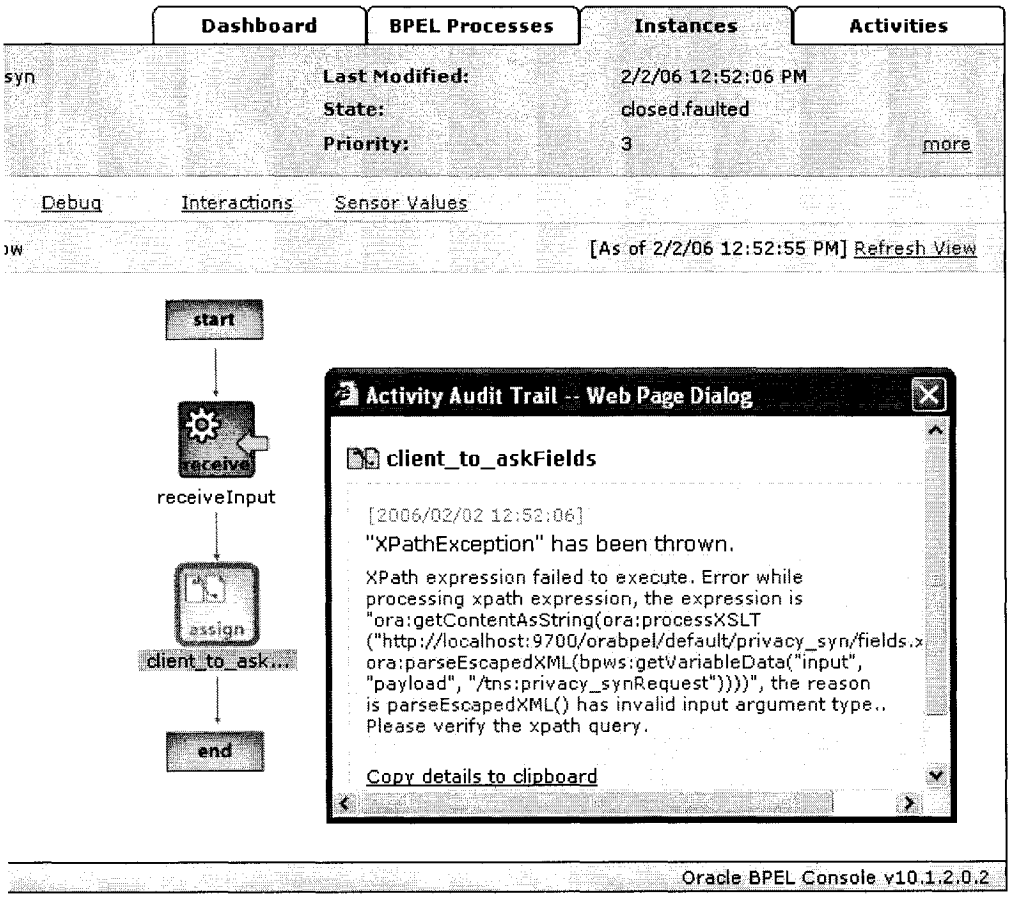

Figure 29: BPEL process debug interface

5 Validation and Verification

In this Chapter, we compare the three different implementation approaches we introduced in Chapter 4: SWA, SSOA and Model-based SOA. We compare development effort, debugging, maintenance and performance. We also make the comparison between Model-based SOA and the related work solutions at the end of this chapter.

In the following sections, we describe the comparisons for each criterion. At the beginning of each section, we will use a table to give the comparison result for all the aspects under that criterion. Then, there is a short explanation about how we got these results for each given aspect.

5.1 Effort with reusability

This criterion compares the effort, in terms of involved coding works and development time, needed to build the sample data access process, reusing existing services and infrastructures where it is possible.

Approaches / Criterion aspects	SWA	SSOA	MODEL-BASED SOA
Involved coding works	Java coding; JSP/Servlet	Enterprise Java coding; XML process coding	Model definition; Dynamic binding based on attributes
Development time+	5 people,4 months *	5 people,2 months **	1 people,2 weeks***
Each person spends 10 hours, in average, per week for this project development * 4 developers, 1 technical business analyst. All services and infrastructure built from scratch. ** 4 developers, 1 technical business analyst. Reuse existing services and infrastructure but re-architect as web services. Developer custom codes process coordination and forms interfaces. *** 1 developer, who is familiar with business processes according to the participation of previous two developments, defines business model using BPEL and XForms. + The data specified for this criterion aspect measures the incremental effort based on the preceding approach implementation.			

Table 1: Comparison result for development effort criteria

5.1.1 Involved Coding Works

In the implementation of the SWA approach, both of the functionalities and business process of the online system were implemented in a single web application. This web application was coded in Java JSP and Servlet in order to dynamically generate web content according to the user input.

In the implementation of the SSOA approach, the business process was coded in Java and required to invoke the web services provided by participating applications. We spent extensive works to define the business process workflow in Java code and programming the web service invocation. Besides, we also wrote Java code to process XML data instance. Therefore, all developers had to be enterprise developers with a strong understanding of the SOA. Each developer working on services and infrastructure had to wrap their components in web service interfaces. The senior developer writing the code for coordination of the business processes had to understand and leverage those web interfaces.

In the Model-based SOA approach, because BPEL provides the ability to design the business process at high level, it does not require writing Java code to design the business process. The persons who design the process does not need to concern about the implementation details of low level programming such as how to generate a local proxy/stub in order to invoke web services. For example, asynchronous invocation, business compensation, business transaction and concurrent business activities are provided as palettes which one can drag and drop to the visualized process design interface without writing code. The declarative XML format BPEL process definition of the sample data access system, as attached in Appendix 3, was automatically generated by the BPEL designing tool, Oracle BPEL designer in our implementation, from the visually defined business process model. The whole process development procedure was completed without programming.

5.1.2 Development time

In the implementation of the SWA approach, because all the functionalities were specifically coded for the data access process, we had to build them from scratch. It took the development team 4 months to build this single web application. This included interaction

with a technical business analyst who defined the business process and mocked up forms in an HTML editor.

In the implementation of the SSOA approach, we did not build all the functionalities from scratch since it allows reuse of existing functionalities as web service. However, because the business process is coded in Java in this approach, a programmer was intensively involved in coding the business process. Besides, since HTML web form sends the user input as many individual name/value pairs, we had to spend lots of processing efforts to assemble these individual name/value pairs into a single XML documents as the input for the business process. It took the development team 2 months to build this simple SOA based application (Since the functionalities were reused from existing applications, the building time for these applications is not included in the overall development time of this approach. However, the time spent on wrapping existing applications as web services is taken into consideration). This included interaction with a technical business analyst who verified that the business process was still working properly, and verified the data represented as XML and any changes to the user interface.

In the implementation of the Model-based SOA approach, the graphic based BPEL design tools enabled creating the business process by simple drag and drop without Java coding. Also, BPEL standard itself and the BPEL designing tools provide easy ways to implement some specific requirements for business process support such as asynchronous invocation correlations, business transaction and so on. At presentation layer, the XForms web forms directly submit XML document to the server side. Therefore, there is no extra effort needed to process user input. In our implementation, the developer, who was familiar with the business process through participating previous developments using other approaches, finished building the online data access process in 2 weeks using BPEL and XForms. (The time spent on wrapping web service interfaces was not taken into account as this was already done in the SSOA approach.)

Although the way we measure the development time is based on the incremental effort from the previous implementation approach and it looks the comparison result does not

completely reflect what is the real development effort, we can still see the advantages of using SOA and model-based approaches. By reusing the functionalities created in the SWA approach instead of rebuilding them again, the SSOA implementation only takes half of the original development time. Furthermore, after introduced Model-based method using BPEL and XForms, the extra effort to rebuild the sample system based on the result from the implementation of the SSOA approach is only 2 weeks. In the latter two implementation approaches, if we do not reuse the existing functionalities and Model-based technologies, we believe the difference of the development time will not be so significant.

5.2 Debugging

This criterion compares the effort need to understand, debug, and monitor the business process of the online data access system

Approaches / Criterion aspects	SWA	SSOA	MODEL-BASED SOA
Understanding business process	No structural architecture; Not model based	Structural architecture; Not model based	Structural architecture; Model based
Debugging	Code level	Code level	Code level; Business process level
Monitoring	Code execution	Code execution; Service access	Code execution; Service access; Process execution

Table 2: Comparison result for debugging criteria

5.2.1 Understanding business process

In order to facilitate system reengineering, the business process of the system needs to be easily understood. However, since source code is not a visual and intuitive way to represent the business process, it is not easy to understand the business process if one has to refer to source code of the process definition.

In the implementation of the SWA approach, the business process of our system was coded in Java language and mixed with functionality implementation. Therefore, it is not a structural and model based approach. The analysis of the Java source code is required to understand the business process. The mixing structure of the process and functionality

definition makes the source code even harder to be read. Therefore, the business process implemented in this approach is hard to be understood.

In the implementation of the SSOA approach, we defined business process separately from functionalities and made it easier to be distinguished from functionalities. Therefore, it has better structural architecture than the SWA approach. However, since the whole data access process workflows were coded in a dictated Java application, the understanding of the process workflow still relies on the reference to the Java code definition.

In the implementation of the Model-based SOA approach, we defined the business process in BPEL. Because the BPEL is in XML format which is written in plain text and structured by XML tags, it is easier to be understood than Java code. Besides, the BPEL design tools (we used Oracle BPEL designer in our implementation but it is the same for most of BPEL design tools) provide visualized view of BPEL process and may help one understand BPEL process workflow easier because no source code reference is needed. Furthermore, because XForms was used to implement the user interfaces, and their data instances are based on the business model of the targeting system, their definitions give a better understanding of the corresponding business process.

5.2.2 Debugging

In the implementation of the SWA approach, because we coded all the functionalities and workflows inside a single web application, the application can only be debugged by analyzing the code. The effort of code level debugging became more complicated with the increase of the functionalities we integrated into the system. Especially, if the application does not have a good packaging strategy to organize its code components, it is very hard to debug it.

In the implementation of the SSOA approach, the code level debugging still needs to be performed for each of the functionalities. On the other hand, because each functionality is provided as a self contained, standalone web service, when developers analyze one functionality, they do not need to concern the factors from other functionalities. It dramatically simplified the complexity for system debugging. While the business process is

coded using programming language, the process debugging still can only be performed at the code level.

In the Model-based SOA approach, not only the debugging complexity is decreased brought by leveraging SOA concept, also the BPEL server provides visualized high level debug information for process level debugging. For example in Figure 29, the debug tool reminds users that the BPEL "assign" activity encountered an "XPathException" with a very intuitive graphic interface. Therefore, a developer familiar with the business process can easily debug problems in the business process level using BPEL server interface without having to do code level debugging. At the same time, the functionality level debugging is still at code level.

5.2.3 Monitoring

Monitoring the execution of business process can help finding business trends then optimizing process workflow.

In the SWA approach, because there is no separate business process definition, we could only monitor the programming code execution. However the code execution is not an intuitive monitoring because it is hard to associate business flows and activities with their corresponding representations in the programming code.

In the SSOA approach, for each functionality, we could monitor it at code execution level. Beyond that, because each service access/invocation from business process uses standard web service invocation, we could also monitor every service access. However, since the business process is coded using programming languages, the service access monitoring still requires monitoring the code execution of the business process layer. As the reason we explained above, because the code execution is not an intuitive monitoring method, some useful information such as business trends, inappropriate process steps may not be found when one monitors process execution.

In the Model-based SOA approach, not only we could monitor code execution and service access, BPEL server also provided us the ability to track the execution of business process

and presented a visualized runtime environment. Therefore, the business process can be easily monitored and optimized.

5.3 Maintenance

This criterion is focusing on the comparison of effort need to maintain the data access system when the changes are needed for the business process and data model.

Approaches / Criterion aspects	SWA	SSOA	MODEL-BASED SOA
Business process change	Always code change	Code change	Model change
Form format change	Manual UI change	Manual UI change	Model change

Table 3: Comparison result for maintenance criteria

5.3.1 Business process change

The business process may need to be changed periodically in order to apply new business needs such as optimizing current workflow, adapting marketing trends and so on.

In the implementation of the SWA approach, because the business process and functionalities were coded in the same layer, the change of the business process inevitably requires not only code change for process definition, but also re-packaging of the whole application.

In the implementation of the SSOA approach, we defined the business process separate from functionalities, therefore the process change did not affect functionality layer. While the business processes were coded using programming language, any change of business process still required change of code.

In the implementation of the Model-based SOA approach, since business process, which represents the business model, was implemented in BPEL, the developer could easily change the business process using visualized BPEL designing tools. The change of business process, workflow change for example, only affected the change at BPEL process layer.

Because BPEL process layer completely separated from the functionality layer, there was no code change needed for business process change.

5.3.2 Form format change

The data request form which accepts user input may need to change its format from time to time. For example, the input field "agree to term" in the data request form needs to be changed from a text input box, which allows user inputs any text values, to a checkbox which only allows user only to choose from "true" or "false".

In both the SWA and the SSOA approaches, we used HTML form to accept user input. Because there was no data model defined in the HTML form and the data definition and its visual presentation were mixed together. Therefore, in order to make such change, we had to change the HTML control from an input field type to a radio button type and specify "true" and "false" options for it.

In the Model-based SOA approach, as we described in section 3.4.3, the XForms UI was driven by the XML data model. Since we used XML Schema to define the XML data model, we applied the form format change by changing the schema definition of the "agree to term" field. After we changed its schema definition from xml:String to xml:Boolean, the corresponding input filed, which was rendered on XForms compatible browsers, was automatically changed from an input box to a radio button selection with only "true" and "false" options.

5.4 Performance

The performance criterion compares the response time of the systems to fulfill a same data request.

Approaches / Criterion aspect	SWA	SSOA	MODEL-BASED SOA
Response time+	5 seconds *	8 seconds *	15 seconds *
The tests have been performed on a PC with P4 3.0G CPU, 1G memory and Microsoft Windows XP installed. We only allowed necessary processes running when we did the test to minimize the performance impact from other processes. + The response time is calculated based on the average of 3 times of execution for fulfilling the same sample request.			

Table 4: Comparison result for performance criteria

5.4.1 Response time

The data request we choose for comparing the response time has all the requested fields automatically approved according to policy definition. Otherwise, the process workflow would need to involve the collaboration service which is a manual process and lets researcher and privacy officer communicate each other through an online web interface. Since the data request can not be automatically fulfilled, it is hard to get the response time under the same conditions. Therefore, we used an automatic approval request as the input for this comparison.

In the implementation of the SWA approach, there was no extra software platform infrastructure. Besides, since all the functionalities and business process definition were in the same Java runtime environment, the communications between them were direct function calls. When it ran on the test PC, it took 5 seconds to finish the sample request.

In the implementation of the SSOA approach, there was some extra overhead for web service invocations. Each web service invocation needed to assemble the requests and responses into SOAP message format. Additionally, each application ran on its own software environment, although they were on the same physical testing PC, and it was out of the control of the online data access process. During our testing, it took 8 seconds to finish the sample request on the testing PC.

In the implementation of the Model-based SOA approach, according to the software environment requirement as presented in section 4.4.2, there were several extra software platforms needed to be setup in this approach. A BPEL runtime environment was needed to parse the BPEL process. Secondly, in order to view the XForms web forms correctly, it required a XForms enabled browser which did extra processing than just rendering HTML pages as in the previous two approaches. It took 15 seconds to finish the sample request on the testing PC.

It should be noted, however, that although the response time required for a single request was higher on a single machine for both SSOA and Model-based SOA, both approaches are designed to leverage a multi-machine environment for greater scalability. In an SOA environment, each service can run on a separate machine and this enables a far greater number of requests to be handled in parallel. On a single machine in support of SWA we would expect performance to degrade substantially as the load on the machine increased and there is no easy way to run a SWA across multiple machines [Arlitt2001]. Whereas with SSOA and Model-based SOA the extra infrastructure running on a single machine slows down processing and causes competition for resources for a single request. But running each service on a separate machine would both avoid competition for resources on a single request, and allow for much greater scalability in handling multiple requests.

5.5 Comparison with related work

In section 2.3, we listed 3 related works and introduced how other researchers resolved the related problems. The comparison between the thesis solution and the 3 related work solutions is summarized in Table 5:

Solutions / Criterion / Aspects	[Arnesen2003] Solution	[Raut2003] Solution	[Peyton2005] Solution	MODEL-BASED SOA Solution
Privacy protection	Included	Not included	Included	Included
SOA enabled	No	Partly	Yes	Yes
Business process definition	No	Choreography	No	Orchestration
Development effort (expected)	High	High	Medium	Low
Maintenance effort (expected)	High	High	Medium	Low
Framework, architecture and Implementation details	Covers framework privacy specific functionalities; No system architecture; No implementation details	Covers system architecture; No implementation details	Covers privacy related functionalities; Covers system architecture; No implementation details	Covers privacy specific functionalities, framework architecture and implementation details

Table 5: Comparison with related works approaches

In addressing the similar problems as presented in section 1.1, each related work resolves part of the issue but with its limitations.

[Arnesen2003] introduces a comprehensive policy enforcement framework for privacy protection. But it does not consider integrating with existing functionalities. Therefore, the development effort needed is high. Besides, because there is no separate business process definition, all the functionalities and processes are implemented in Java code. The maintenance effort is also high.

[Raut2003] is not a privacy protection specific solution. So there is no coverage about privacy related functionalities. For the internal integration, it adopts EAI approach. According to the introduced in 2.2.4, EAI approach requires high implementation effort. Furthermore, the choreography mode for business process invocation makes participating services process specific and hard to reuse and maintain.

[Peyton2005] gives the concept of a SOA approach to support privacy and collaboration within enterprise processes. Specifically, it focuses on addressing collaborative process for sensitive data access and presents privacy protection related functionalities and technologies. However, the solution does not include detail information about its system implementation, nor does it focus on the model-based aspects of process coordination (BPEL) or forms (XForms).

The thesis proposed solution, in comparing with above three solutions, gives definitions of all the major functionalities related to privacy protection in the implementation of the Model-based SOA approach, as presented in section 4.3.1. Based on SOA, the BPEL business process engine orchestrates participating services. And these services are provided as the result of reusing and integrating existing applications. According to the analysis from section 5.1 to 5.4, the Model-based SOA approach of the thesis solution makes the system easy to develop and maintain. Also, the thesis proposed solution illustrates the technical details of all aspects from framework architecture to business process engine, message exchange format and user interface.

5.6 Objectives achievement verifications

In section 1.2, we presented the important parts of the objectives of our research. We verify the accomplishment of these objectives in this section.

Objective 1: Business process must be easy to be created

First, as illustrated in section 5.1, the Model-based SOA approach requires less programming efforts and man power to implement the sample approval process. Therefore, the approval process is easy to be created.

Objective 2: Business process can easily be understood and monitored

Secondly, section 5.2 presents using the Model-based SOA approach, one can leverage the text based XML format process definition and visualized BPEL process design and runtime tools to understand and monitor business process without reference to Java source code which is not a intuitive and easy way for analysis.

Objective 3: Business process needs to be easily changed without coding at programming level

Thirdly, section 5.3 compares the maintenance efforts needed for three implementing approaches. According to the comparison results, in the Model-based SOA approach, the change of business process can be easily realized without coding change.

Objective 4: User Interface needs to be easily created and changed

Finally, according to the comparison in 5.1.1 and 5.3.2, XForms and XML technologies adopted in the Model-based approach make the user interface easy to be created and changed.

6 Summary and Conclusion

6.1 Summary

In this thesis, we presented a model based service oriented architecture approach to build the enterprise process. Specifically, we leveraged emerging standards BPEL, XForms, SOA, and XML to implement our solution and apply it on the health care field to build a sensitive data access process.

First, we gave some background information from both business and technical aspects. At the business side, we introduced what intentions drive us to build such sensitive data access process. Then, we highlighted the importance of the business process and the related business process management technologies. At the technical side, we introduced some traditional software development and integration approaches such as Data Level Integration, API, ERP, EAI, Web Service and SOA. At the same time, we also demonstrated the limitations of these traditional approaches. Besides, we introduced some related works performed by other researchers in addressing the similar problems.

Secondly, we proposed a Model-based SOA approach in a three layer framework architecture. At the user interface layer, the XForms web forms make the user interface driven by the XML data model and XML schema. At the business process layer, we use BPEL to define the workflow of the business process and orchestrate all the participating functionalities on the fly[25]. And at the functionality layer, we provide all the major functionalities as web services based on SOA concept. Furthermore, we adopt XML technologies, XML document, XML schema and XSLT, to define a cohesive message exchange and transformation format among all the participating applications and the three layers.

Thirdly, we applied the Model-based SOA approach as well as two other traditional software development approaches, a simple web application approach (SWA) and a simple SOA

[25] Although BPEL allows flexibly orchestrating business process at the design phrase, in order to enforce the new process workflow, it still requires shutting down the current process execution and re-deploying the new one

approach (SSOA), to build a simple sensitive data access process based on the Ottawa Hospital data warehouse data request approval process. We transferred the original paper based data request approval process to the online data access processes using the three different approaches.

Finally, we compared the three development approaches based on several criteria: development effort, debug, maintenance and performance. We also compared three related work solutions with the thesis proposed solution.

6.2 Conclusion

According to the comparison result as we described in section 5.1 to 5.4, the thesis proposed Model-based SOA approach is better than the other two traditional approaches, SWA and SSOA at the following aspects:

The Model-based SOA approach does not require programming, nor does it require the knowledge of a programming language like Java, to build the business process. One who is familiar with the business process can create the BPEL business process by simple drag and drop actions. Furthermore, it takes less time to develop the software system using this approach.

Also, the business process created using the Model-based SOA approach is easier to be understood by one who is familiar with business process because it has structural architecture and is based on business model. The visualized BPEL runtime management tools provide a high-level view of process execution without the need to reference to Java source code. Therefore, it is easier for them to debug and monitor for code execution, service access and process execution.

Equally important, the Model-based SOA approach makes business process can be easily changed without re-writing code. This minimizes the effort needed for maintaining the business process and keeping it adapting to new changes. The XForms and XML technologies in the Model-based SOA approach also make the user interface driven by the

underline data model, without changing the XForms web form sources, to adapt the change of the form data model.

The overall benefit of the Model-based SOA approach is to build a concise declarative description of WHAT are the functionalities and business process workflow are needed for the targeting system. By reusing the existing functionalities and having the BPEL and XForms engines to interpret and execute the process workflow and generate user interfaces, the developers or business modelers do not need to write tangled procedural code to describe HOW the system behavior.

However, because the Model-based SOA approach relies on several extra software platforms, such as BPEL server, special XForms enabled browser, it has lower performance behavior on a single testing machine compared with the other two approaches during our implementation. It takes longer time to fulfill a user request in our sample data access process system. But we expect the performance difference between the Model-based SOA and the other two approaches will be minimized when the building systems have more complicated business process and functionalities, especially in a typical corporation enterprise IT environment, and a distributed multi-machine environment can be leveraged for scalability. Under those conditions, according to the analysis at the end of section 5.4.1, the SWA and the SSOA approach may reach their performance limitation and the Model-based SOA approach may perform better.

We also acknowledge that, except BPEL standard we adopted in the Model-based SOA approach to describe the business process, there are other viable notations and technologies can be used to define and automate the business process, such as UML activity diagrams [Lethbridge2001] and Use Case Maps [Weiss2005]. Due to the limitation of the thesis space, we do not compare them in details.

As well, according to the comparison of section 5.5, the thesis proposed solution shows improvement over three related solutions proposed by other researchers in several aspects.

The Model-based SOA approach leverages SOA and BPEL business process engine therefore minimizes software system develop and maintain effort.

The thesis solution presents a practical implementation of the Model-based SOA approach on an example of online health care data access approval process. According to this case study, it addresses several important functionalities for privacy protection.

The thesis proposed solution is illustrated in details including framework architecture, message exchange format and user interface.

6.3 Future work

The Model-based SOA approach we proposed in this thesis is heavily depend on web service. In the functionality layer of the framework architecture, we need every participating application to provide web service interface for BPEL business process invocation. However, web service is still a relative new standard and not all the legacy applications can provide web service interfaces. In order to make our solution also applicable to these legacy applications, we are planning to introduce WSIF, as defined in section 2.2.5, which can bind not only a SOAP based web service but also other implementations such as Java class as the service for BPEL invocation. Some BPEL product venders also extended WSIF in their BPEL product solutions. In the future, we need to investigate the probabilities to applying it in our approach.

Thesis Reference

[Aalst2003] Wil M.P. van der Aalst, Arthur H.M. ter Hofstede, Mathias Weske, "Business Process Management: A Survey", International Conference, BPM 2003

[Araujo2002] I. Araujo, M. Weiss, "Linking Non-Functional Requirements and Patterns", Pattern Languages of Programming (PLoP), 2002

[Arlitt2001] M. Arlitt, D. Krishnamurthy, J. Rolia, "Characterizing the Scalability of a Large Web Based Shopping System", ACM Transactions on Internet Technology, Vol. 1, No. 1, August 2001, Pages 44–69

[Arnesen2003] R. Arnesen, J. Danielsson, "A Framework for Enforcement of Privacy Policies", Nordic Security Workshop 2003

[Bodoff2004] Stephanie Bodoff, Eric Armstrong, Jennifer Ball, Debbie Bode Carson, "The J2EE Tutorial, Second Edition", Addison-Wesley Professional, ISBN 032124575X

[Chung1999] Lawrence Chung, Brain A Nixon, Eric Yu, John Mylopoulos , "Non-Functional Requirements in Software Engineering", 1999, Springer, ISBN 0792386663

[Dubinko2003] Micah Dubinko, "XForms Essentials", 2003, O'Reilly, ISBN 0596003692

[Endlich2004] Ulrich Endlich, "An Evaluation of Enterprise Application Integration Solutions and the Role of Web Services", Master Thesis, Vienna University of Technology (TU Wien), 2004

[Erasala2003] Naveen Erasala, David C. Yen, T.M. Rajkumar, "Enterprise Application Integration in the electronic commerce world", Computer Standards & Interfaces 25 (2003) 69–82

[Erl2004] Thomas Erl, "Service-Oriented Architecture: A field Guide to Integrating XML and Web Services", 2004, Prentice Hall PTR, ISBN 0-13-142898-5

[Hammer1993] Michael Hammer and James Champy, "Reengineering the corporation: A manifesto for business revolution", 1993, HarperBusiness, ISBN 0-88730-640-3

[Juric2004] Matjaz B.Juric, Benny Mathew, Poornachandra Sarang, "Business Process Execution Language for Web Service", 2004, PACKT, ISBN 1-904811-18-3

[Kimball2002] Ralph Kimball, Margy Ross, "The Data Warehouse Toolkit, second edition", 2002, WILEY, ISBN 0-471-20024-7

[Lee2003] Jinyoul Lee, Keng Siau, Soongoo Hong, "Enterprise Integration with ERP and EAI", February 2003/Vol. 46, No. 2 COMMUNICATIONS OF THE ACM

[Lehmann2004] Mike Lehmann, "Weaving Web Services Together", 2004, Oracle Magazine July/Augest 2004

[Lethbridge2001] Timothy C. Lethbridge, Robert Laganière, "Object-Oriented Software Engineering: Practical Software Development using UML and Java, Second Edition", McGraw Hill, 2001, ISBN 0-07-710908-2

[Newcomer2002] Eric Newcomer, "Understanding Web Service", 2002, Wesley, ISBN 0-201-75081-3

[Pavlovic2004] Marina Pavlovic, "DCL 7301B: Regulation of Internet Commerce" course material, University of Ottawa, Winter 2004

[Peyton2004] Liam Peyton, Max Nozin, "Tracking Privacy Compliance in B2B Networks", Sixth International Conference on Electronic Commerce, 2004

[Peyton2005] Liam Peyton, Jun Hu, Hao Liu, Mansour Al-Saleh, Abdulmotaleb El Saddik "A Collaborative Approval Process for Accessing Sensitive Data", International Journal of Computer Applications in Technology (IJCAT), Vol. , No. , 2005

[Raible2004] Matt Raible, "Spring Live", 2004-2005, SourceBeat

[Raut2003] A. Raut , A. Basavaraja, "Enterprise business process integration", TENCON 2003. Conference on Convergent Technologies for Asia-Pacific Region

[Rindfleisch1997] Thomas C. Rindfleisch, "Privacy, information technology, and health care", Communications of the ACM, v.40 n.8, p.110-117, Aug. 1997

[Rodney2003] Jeffrey D.Rodney, "ADM6276: Enterprise Resource Planning (ERP) Systems Management" course material, University of Ottawa, Fall 2003

[Scheer2004] Scheer, Abolhassan, Jost, Kirchmer (editors), "business process automation", 2004, Springer, ISBN 3-540-20794-5

[Singh2002] Inderjeet Singh, Beth Stearns, Mark Johnson, Enterprise Team, "Designing Enterprise Applications with the J2EE Platform", 2002, Addison-Wesley Professional, ISBN 0201787903

[Singh2004] Inderjeet Singh, Beth Stearns, Sean Brydon, Greg Murray, Vijay Ramachandran, "Designing Web Services with the J2EE 1.4 Platform (The Java Series): JAX-RPC, SOAP, and XML Technologies", 2004, Pearson Education, ISBN 0321205219

[W3C-XForms] "XForms - The Next Generation of Web Forms", Retrieved April 7, 2006 from http://www.w3.org/MarkUp/Forms/

[W3C-XMLSchema] "XML Schema", Retrieved April 7, 2006 from http://www.w3.org/XML/Schema

[Weiss2005] Michael Weiss, Daniel Amyot, "Business Process Modeling with URN", International Journal of E-Business Research, 1(3), 63-90, July-September, 2005

Appendix 1: The Ottawa Hospital Data Warehouse Data Request Form

☐ **New application for data**

☐ **Addendum to existing application**

1. **Protocol Title**

2. **REB Approval number (please submit copy of approval)**

3. **Principal investigator**

Last Name **First Name**

Title/Position **Department**

Telephone **Email**

4. **Do you require information that could be used to determine a patient's identity?**

☐ **Yes** ☐ **No**

5. **If yes, please indicate the identifiable data elements requested (if no skip to question 6)**

☐ **Patient Name** ☐ **Street Address**

☐ **City** ☐ **Full Postal**

☐ **Any part of postal code** ☐ **Year of Birth**

☐ **Day of Birth** ☐ **Month of Birth**

☐ **Discharge Date** ☐ **Date of Death**

☐ **Age** ☐ **SIN**

☐ Phone number ☐ MRN

☐ OHIP number ☐ Admission Date

☐ Other

6. Please justify the need for access to the identifiable information above

7. Please list any de-identified data elements required

8. Sometimes de-identified data can be used to determine the identity of individuals. For example, a patient with a very rare medical condition who could be identified based on the diagnosis. Please describe why it will be difficult to determine the identity of the patients in your sample with the de-identified data elements you are requesting.

9. Are you planning on linking data records from TOHDW data with other data sources?

☐ Yes ☐ No

If yes, please describe your linking procedure:

a. Please list the TOHDW data element(s) that will be used for linkage.

b. Please indicate the name of the other data source that will be used for linking.

c. Please describe the specific data elements you are interesting in linking in the data set.

d. Please indicate the physical location where linking will occur.

e. Please indicate where the final data set will reside.

f. **Is the data set that you are linking to an external organization with their own privacy and confidentiality policies and procedures?**

☐ Yes ☐ No

If yes, please name the organization and attach a copy of the organizations Privacy Policy

g. **Is there a possibility that linking the data will render de-identified data identifiable?**

☐ Yes ☐ No

If yes please explain.

10. **Has the patient consented to the data linkage?**

☐ Yes ☐ No

If no please comment on the reason patient consent was waived.

11. **Please indicate the safeguards you will employ to ensure the confidentiality of the data.**

- ☐ Information will not be stored on a portable devi ce
- ☐ PC is password protected
- ☐ CD, diskett, paper report will be kept under lock and key
- ☐ Data will not be stored on a shared PC
- ☐ Other

12. **How will the data be disposed?**

- ☐ Hard copy shredded
- ☐ Disquettes and/ or CDs destroyed

☐ **Hard drive deleted (scratched)**

☐ **Files on server deleted**

☐ **Other**

13. Please indicate the amount of time in months that you will require the data?

14. By signing this document the researcher acknowledges that he/she has read and fully understands The Ottawa Hospital Data Warehouse Governance and Operating document and the privacy and confidentiality policies governing the Ottawa Hospital and the Ottawa Hospital Data Warehouse.

The Researcher agrees that he/she will:

(1) Access only the data that they have been approved to access
(2) Use the data in the manner outlined in this document and no other
(3) Store the data as described in this document
(4) Dispose of the data in the timeframe and method outlined in this document, and
(5) Maintain the confidentiality of the information and respect the Privacy of the data subjects

Researcher _____ Date _____

Data Custodian _____ Date _____

Appendix 2: The XML document sample representation of the Request Form

```
<?xml version="1.0" encoding="utf-8"?>
<request>
<content id="" last_modified_time="" last_modified_person="" status="" ruling="">
<user>
  <id>1</id>
  <name>John Doe</name>
  <role>Senior_Researcher</role>
  <agree_terms>true</agree_terms>
</user>
<data officerId="0">
  <field id="001" identifiable="" potential_identifiable="" anonymous="" dimension="Encounter_Fact" field="EN_Date" function="select" where ="" ruling=""/>
  <field id="002" identifiable="" potential_identifiable="" anonymous="" dimension="Service_Dimension" field="SE_Type" function="" where="='Blood Test'" ruling=""/>
</data>
<purpose>
  <type>Research</type>
  <linkage>Internal</linkage>
  <consent_for_linkage>true</consent_for_linkage >
  <form>
    <question1>yes</question1>
    <question2>yes</question2>
  </form>
</purpose>
<action>
    <format>Data_File_Manipulatable</format>
    <time>Less_than_1_month</time>
```

<disposal_method>File_destroyed</disposal_method>

</action>

<result>

<link></link>

</result>

</content>

<signature_user>

</signature_user>

<signature_officer>

</signature_officer>

</request>

Appendix 3: The BPEL process definition for the sample data access process

```
<!-- privacy_syn BPEL Process [Generated by the Oracle BPEL Designer] -->
<process name="privacy_syn" suppressJoinFailure="yes"
xmlns:tns="http://acm.org/samples"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
xmlns:ora="http://schemas.oracle.com/xpath/extension"
xmlns:nsxml0="http://localhost:8080/epal_ws/services/epalWS"
xmlns:nsxml1="http://localhost:7001/sql_sig_sen/services/sss"
xmlns:nsxml2="http://localhost:7001/epal_ws/services/epalWS"
xmlns:ns0="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:nsxml3="http://localhost:7001/privacy/services/collaborate"
xmlns:nsxml4="http://localhost:8080/privacy/services/collaborate"
targetNamespace="http://localhost:9700/orabpel/default/privacy_syn"
xmlns:nsxml5="http://localhost:8080/sql_sig_sen/services/sss"
xmlns:epal="http://www.research.ibm.com/privacy/epal/interface">
<!-- ================================================================= -->
<!-- PARTNERLINKS                                                      -->
<!-- List of services participating in this BPEL process               -->
<!-- ================================================================= -->
<partnerLinks>
<!-- The 'client' role represents the requester of this service. -->
<partnerLink name="client" partnerLinkType="tns:privacy_syn"
myRole="privacy_synProvider"/>
<!--
<partnerLink name="privacy_tomcat"
partnerLinkType="nsxml4:CollaborateImpLink"
partnerRole="CollaborateImpProvider"/>
-->
<partnerLink name="sss_ws" partnerLinkType="nsxml5:Signature_SQLLink"
partnerRole="Signature_SQLProvider"/>
<partnerLink name="epal_ws" partnerLinkType="nsxml0:testdriverLink"
partnerRole="testdriverProvider"/>
</partnerLinks>
<!-- ================================================================= -->
<!-- VARIABLES                                                         -->
<!-- List of messages and XML documents used within this BPEL process  -->
<!-- ================================================================= -->
<variables>
<!-- Reference to the message passed as input during initiation -->
<variable name="input" messageType="tns:privacy_synRequestMessage"/>
<!--
Reference to the message that will be returned to the requester
-->
<variable name="output" messageType="tns:privacy_synResponseMessage"/>
<!--
<variable name="signUserRequest" messageType="nsxml4:signUserRequest"/>
<variable name="signUserResponse" messageType="nsxml4:signUserResponse"/>
-->
<variable name="askFieldsRequest" messageType="nsxml5:ask_FieldsRequest"/>
```

```
<variable name="askFieldsResponse"
messageType="nsxml5:ask_FieldsResponse"/>
<variable name="no_of_fields" type="xsd:int"/>
<variable name="epalRequestMessage"
messageType="nsxml0:epalQueryRequest"/>
<variable name="epalResponseMessage"
messageType="nsxml0:epalQueryResponse"/>
<variable name="requestWithFieldSensitivities" type="xsd:string"/>
<variable name="requestTemp" element="tns:requestTempType"/>
<variable name="maybe" type="xsd:int"/>
<variable name="deny" type="xsd:int"/>
<variable name="allow" type="xsd:int"/>
<variable name="epalPolicy" type="xsd:string"/>
<variable name="epalRuling" type="xsd:string"/>
<variable name="epalRulingOverall" type="xsd:string"/>
<variable name="loopCount" type="xsd:int"/>
</variables>
<!-- ================================================================= -->
<!-- ORCHESTRATION LOGIC                                               -->
<!-- Set of activities coordinating the flow of messages across the    -->
<!-- services integrated within this business process                  -->
<!-- ================================================================= -->
<sequence name="main">
<!-- Receive input from requester.
Note: This maps to operation defined in privacy_syn.wsdl
-->
<receive name="receiveInput" partnerLink="client"
portType="tns:privacy_syn" operation="process" variable="input"
createInstance="yes"/>
<!-- Generate reply to synchronous request -->
<assign name="client_to_askFields">
<copy>
<from
expression="ora:getContentAsString(ora:processXSLT(&quot;http://localhost:
9700/orabpel/default/privacy_syn/fields.xslt&quot;,
ora:parseEscapedXML(bpws:getVariableData(&quot;input&quot;,
&quot;payload&quot;, &quot;/tns:privacy_synRequest&quot;))))"></from>
<to variable="askFieldsRequest" part="XML_Doc"/>
</copy>
</assign>
<invoke name="invoke-1" partnerLink="sss_ws"
portType="nsxml5:Signature_SQL" operation="ask_Fields"
inputVariable="askFieldsRequest" outputVariable="askFieldsResponse"/>
<assign name="countNoOfDataAndInit"><copy>
<from
expression="count(ora:parseEscapedXML(bpws:getVariableData(&quot;askFields
Response&quot;, &quot;ask_FieldsReturn&quot;))/field)"></from>
<to variable="no_of_fields"/>
</copy>
<copy>
<from variable="no_of_fields"></from>
<to variable="loopCount"/>
</copy>
<copy>
<from expression="0"></from>
<to variable="maybe"/>
</copy>
```

```
<copy>
<from expression="0"></from>
<to variable="deny"/>
</copy>
<copy>
<from expression="0"></from>
<to variable="allow"/>
</copy>
</assign>
<while name="while-1"
condition="bpws:getVariableData('loopCount')&gt;0"><sequence><assign
name="requestToEPALrequest"><copy>
<from
expression="ora:parseEscapedXML(bpws:getVariableData(&quot;askFieldsRespon
se&quot;,
&quot;ask_FieldsReturn&quot;))/field[bpws:getVariableData(&quot;no_of_fiel
ds&quot;)]"></from>
<to variable="requestTemp"
query="/tns:requestTempType/tns:request/tns:field"/>
</copy>
<copy>
<from
expression="ora:parseEscapedXML(bpws:getVariableData(&quot;input&quot;,
&quot;payload&quot;,
&quot;/tns:privacy_synRequest&quot;))/content"></from>
<to variable="requestTemp"
query="/tns:requestTempType/tns:request/tns:content"/>
</copy>
<copy>
<from
expression="ora:getContentAsString(ora:processXSLT(&quot;http://localhost:
9700/orabpel/default/privacy_syn/epal.xslt&quot;,bpws:getVariableData(&quo
t;requestTemp&quot;,
&quot;/tns:requestTempType/tns:request&quot;)))"></from>
<to variable="epalRequestMessage" part="query"/>
</copy>
</assign>
<invoke name="invoke-1" partnerLink="epal_ws" portType="nsxml0:testdriver"
operation="epalQuery" inputVariable="epalRequestMessage"
outputVariable="epalResponseMessage"/>
<assign name="countRuling"><copy>
<from expression="bpws:getVariableData(&quot;loopCount&quot;) - 1"></from>
<to variable="loopCount"/>
</copy>
<copy>
<from
expression="ora:parseEscapedXML(bpws:getVariableData(&quot;epalResponseMes
sage&quot;, &quot;epalQueryReturn&quot;))/epal:originating-
rule/@refid"></from>
<to variable="epalPolicy"/>
</copy>
<copy>
<from
expression="ora:parseEscapedXML(bpws:getVariableData(&quot;epalResponseMes
sage&quot;, &quot;epalQueryReturn&quot;))/@ruling"></from>
<to variable="epalRuling"/>
</copy>
```

```
</assign>
<switch name="assignRuling">
<case condition="bpws:getVariableData(&quot;epalPolicy&quot;) =
&quot;DEFAULT&quot;"><assign name="countMaybe"><copy>
<from expression="bpws:getVariableData(&quot;maybe&quot;) +1"></from>
<to variable="maybe"/>
</copy>
</assign>
</case>
<case condition="bpws:getVariableData(&quot;epalRuling&quot;) =
&quot;ALLOW&quot;">
<assign name="countAllow"><copy>
<from expression="bpws:getVariableData(&quot;allow&quot;) +1"></from>
<to variable="allow"/>
</copy>
</assign>
</case>
<otherwise><assign name="conutDeny"><copy>
<from expression="bpws:getVariableData(&quot;deny&quot;) +1"></from>
<to variable="deny"/>
</copy>
</assign>
</otherwise>
</switch>
</sequence>
</while>
<switch name="assignRulingOverall">
<case condition="bpws:getVariableData(&quot;maybe&quot;) &gt; 1"><assign
name="maybe"><copy>
<from expression="&quot;MAYBE&quot;"></from>
<to variable="epalRulingOverall"/>
</copy>
</assign>
</case>
<case condition="bpws:getVariableData(&quot;deny&quot;) =
bpws:getVariableData(&quot;no_of_fields&quot;)">
<assign name="deny"><copy>
<from expression="&quot;DENY&quot;"></from>
<to variable="epalRulingOverall"/>
</copy>
</assign>
</case>
<case condition="bpws:getVariableData(&quot;allow&quot;) =
bpws:getVariableData(&quot;no_of_fields&quot;)">
<assign name="allow"><copy>
<from expression="&quot;ALLOW&quot;"></from>
<to variable="epalRulingOverall"/>
</copy>
</assign>
</case>
<otherwise><assign name="maybe"><copy>
<from expression="&quot;MAYBE&quot;"></from>
<to variable="epalRulingOverall"/>
</copy>
</assign>
</otherwise>
</switch>
```

```
<assign name="assign-2">
<copy>
<from variable="epalRulingOverall"></from>
<to variable="output" part="payload"
query="/tns:privacy_synResponse/tns:result"/>
</copy>
</assign>
<reply name="replyOutput" partnerLink="client" portType="tns:privacy_syn"
operation="process" variable="output"/>
</sequence>
</process>
```