PRIVACY MONITORING AND ENFORCEMENT IN A WEB SERVICE ARCHITECTURE (WSA)

by

Kai Tong

Submitted in partial fulfilment of the requirements
for the degree of Master of Electronic Commerce

at

Dalhousie University
Halifax, Nova Scotia
May 2012

DALHOUSIE UNIVERSITY

Faculty of Computer Science

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled “Privacy Monitoring and Enforcement in a Web Service Architecture (WSA)” by Kai Tong in partial fulfilment of the requirements for the degree of Master of Electronic Commerce.

Dated: May 3rd, 2012

Supervisor: ____________________________________

Readers: ____________________________________

____

DALHOUSIE UNIVERSITY

DATE: May 3rd, 2012

AUTHOR: Kai Tong

TITLE: Privacy Monitoring and Enforcement in a Web Service Architecture (WSA)

DEPARTMENT OR SCHOOL: Faculty of Computer Science

DEGREE: MEC CONVOCATION: October YEAR: 2012

Permission is herewith granted to Dalhousie University to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions. I understand that my thesis will be electronically available to the public.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

The author attests that permission has been obtained for the use of any copyrighted material appearing in the thesis (other than the brief excerpts requiring only proper acknowledgement in scholarly writing), and that all such use is clearly acknowledged.

____
Signature of Author

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The growth of online activities in our daily lives has led to substantially increased attention on how organizations and their computer systems handle Personal Information (PI). Independently, the wide adoption of Web Service Architecture (WSA), for the integration of software, creates an opportunity to facilitate support for privacy by monitoring the use of PI by web services and enforcing applicable privacy policies.

This thesis designs an agent for privacy monitoring and enforcement in a WSA environment and creates a prototype as a proof of concept. The agent is based on a specific multi-agent architecture for privacy compliance. The design of the agent has led to extension of the architecture to bring out its full potential in monitoring PI flows and enforcing privacy policies in a WSA environment. The evaluation of the prototype has led to suggestions on its implementation for an operational environment.

LIST OF ABBREVIATIONS USED

XACML	eXtensible Access Control Markup Language
APPEL	A P3P Preference Exchange Language
P3P	The Platform for Privacy Preferences Project
EPAL	The Enterprise Privacy Authorization Language
DPAL	The Declarative Privacy Authorization Language
EPA	Enterprise Privacy Architecture
PI	Privacy Information
MWSL	Monitored Web Service List
KB	Knowledge Base
WSA	Web Service Architecture
XML	Extensible Markup Language
UDDI	Universal Description, Discovery and Integration
UTC	Coordinated Universal Time
SOAP	Simple Object Access Protocol
HTTP	Hypertext Transfer Protocol
ISTPA	The International Security, Trust, and Privacy Alliance

ACKNOWLEDGEMENTS

This thesis is dedicated to my parents, my all-time supporters. Their love is the biggest support and motivation in my life.

This thesis is also dedicated to my brother, Jianyu Chen. Without his support, I could not make it through those most difficult days of my life.

I owe my deepest gratitude to my mentor, Dr. Peter Bodorik. With his supervision in my study, it has been an enlightment journey in the preparation of this thesis.

CHAPTER 1 INTRODUCTION

1.1 PRIVACY IN E-COMMERCE AND E-BUSINESS

The growth of Internet and electronic commerce has changed how information is exchanged among organizations and individuals. When an organization publishes an e-Commerce web service on the Internet, the organization needs to collect privacy information from customers to honor the provided services. How the organization is going to handle the collected privacy information has drawn accelerated attention from legislative and regulatory mandates and also reflects the nature of an information-rich business environment. As a result, the need for privacy monitoring and privacy policy enforcement in e-Commerce and e-Business has increased substantially. To implement a privacy enforcement system within an organization, the organization needs to know which private data is used by which application, how the private data is used by the applications and where the private data is stored.

1.2 WEB SERVICES ARCHITECTURE

Simultaneous to the growth of the Internet, applications of web services have expanded. Nowadays, as more and more enterprises and organizations are switching to Clouds, Web Services Architecture (WSA), and hence the usage of web services, thrives. No matter whether the web services are for internal clients or external customers, they provide a new way to exchange information among business units and customers and to integrate software within and across organizations. Usage of web services for the integration of software creates an opportunity to facilitate support for privacy by monitoring the use of private data by web services. According to World Wide Web Consortium's (W3C) definition, a web service is a software system designed to support inter-operable machine-to-machine interaction over a network. Other systems interact with the web service by communicating with it using Simple Object Access Protocol (SOAP). There is an opportunity for an organization to determine which web services and applications use

Privacy Information (PI) and also the flow of PI through the organization, by checking the SOAP messages used to communicate with web services. Furthermore, WSA also provides the opportunity to actually enforce compliance with privacy policies. Thus the need to monitor PI and enforce privacy policy arises and that is the focus of this thesis.

1.3 ARCHITECTURE FOR PRIVACY ENFORCEMENT IN A WSA

The need to monitor Privacy Information (PI) flows and enforce privacy policies in a WSA environment presents a challenge to companies and organizations. In a WSA environment PI flows are bi-directional. Flow from the client to a server and flow from the server to a client are both possible. For instance, to use the web services provided by companies and organizations, external customers may need to submit their private information through the Internet. Internal clients in these companies and organizations may need to use an application client (for the definition of application client please refer to section 3.7) to access the privacy information collected from external customers to complete the promised services. As a result of these business activities, PI flow travels among web services and their applications. From the business service provider's point of view, using WSA provides them with an opportunity to boost business and a new way of doing business with customers. However, it also brings some challenges and problems to protect privacy information in a WSA infrastructure. One of the problems is that in a WSA environment, information regarding web services is published in Universal Description Discovery and Integration (UDDI) but there is no platform to publish the information regarding the application client. Therefore, in the absence of good software documentation, information regarding application clients is limited and sometimes unknown. Another problem is that WSA is a dynamic environment where PI flows amongst web services and applications are changing, possibly from one invocation of a web service to another. Another major problem is that web services and applications have been developed without awareness of privacy. These challenges have received more and more attention from businesses, lawmakers and government agencies. As more privacy regulations and 'best practices' of privacy and information protection have been developed, a semi-automatic mechanism to enforce them is needed for the WSA environment.

In the paper, entitled "Privacy Compliance with Web Services" (Bodorik et al., 2009), a multi-agent architecture is presented to support privacy in a WSA environment in order to enforce privacy policies for private information used by applications, including a legacy system, without modifying the system's code. A diagram of the architecture is shown in Figure 1. The architecture addresses the privacy enforcement issue in a WSA environment. Privacy enforcement is based on the premise that, in WSA, communication between applications and system software and resources is through the web service and is invoked by an exchange of XML messages. The authors proposed that a PI Monitor Agent be used as a middleware between applications and web services to intercept requests for web services and replies from web services. The PI Monitor Agent checks these requests and replies against the Enforcement/Monitoring Rules to determine how the requests and replies should be handled. The Enforcement/Monitoring Rules are created to represent the company's privacy policy and also direct the logging of web service requests and reply messages. Logs are stored in Audit Logs and are mined by a PI Agent off-line to acquire useful information, such as: information on private data elements, privacy policies, which applications use PI in what context, and which privacy policies apply to the use of PI. Such acquired information is stored in the Privacy Knowledge Base (KB), which consists of a collection of schema/sub-knowledge bases and agents that manage them and provide assistance to PI administrators.

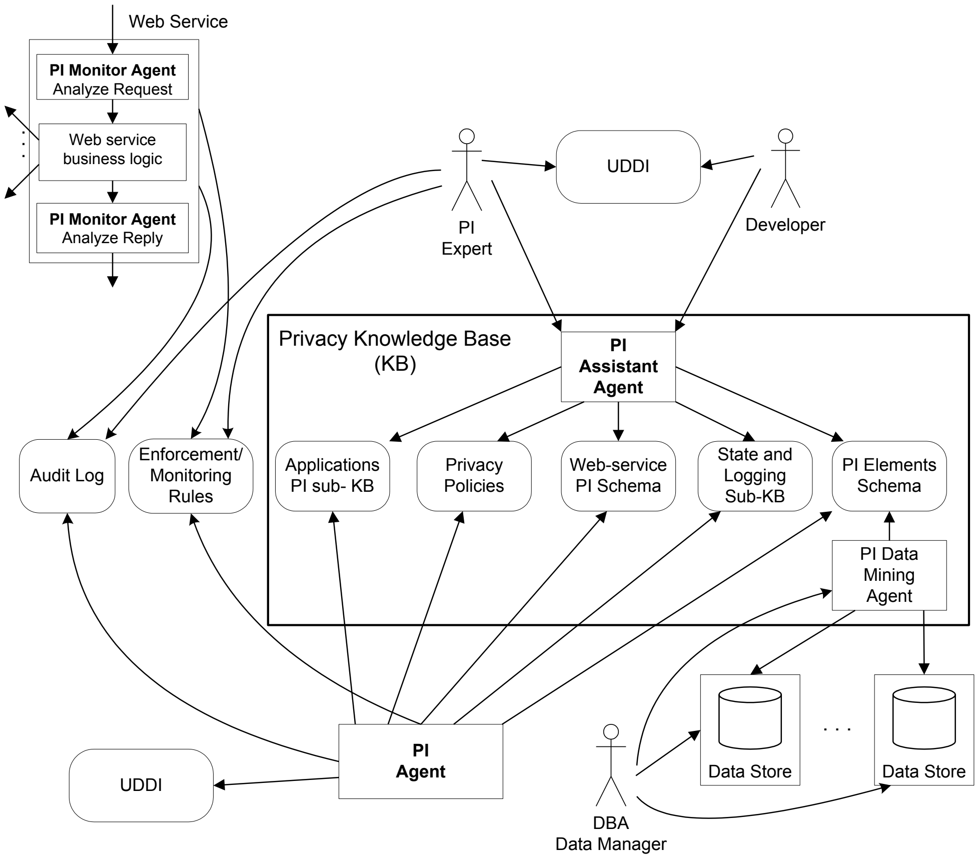


Figure 1 Architecture Proposed in paper Privacy Compliance with Web Services (Bodorik et al., 2009).

One of the principal components of the architecture is a PI Monitor Agent that oversees the invocation of web services. The PI Monitor Agent first checks the PI elements in the requests to web services and replies from web services. Then the PI Monitor Agent consults rules on the action to be taken. Rules are generated by the PI Agent based on the information contained in the Privacy Knowledge Base. The Privacy Knowledge Base consists of seven sub-knowledge bases:

- PI Elements Schema
- PI Data Mining Agent
- Privacy Policies
- Web Services PI Schema
- Applications PI sub-KB
- PI Assistant Agent for PI Experts and Developers

- State/Logging sub-KB

These sub knowledge bases either contain information regarding web services, applications, PI elements, privacy policies and how to log or assist the PI Administrator to manage the knowledge base. The purpose of the Privacy Knowledge Base is to form a dynamic learning mechanism for the architecture to adapt to the changes in the WSA environment.

1.4 Research Question

From the above it follows that to ensure compliancy to privacy policies when PI is used by software, monitoring the use of PI by various software components is required. Although software developers are becoming aware of privacy requirements and privacy standards are being developed, most software is still being developed without much privavcy consideration. Furhtermore, it is clearly unrealistic to expect that the current software used by public and private organization would be modified to ensure compliance to privacy policies. As a consequence, some add-on mechanism, that does not require modification of existing application software, is required for enforcement of privacy policies – the heart of the research question here.

Our literature review revealed only one proposal (Bodorik, 2009) for monitoring of software to ensure privacy compliance on the use of PI, such that existing applications need not be modified. However, feasibility of implementation of the proposed architecture is yet to be established – and that is the main objective of this thesis.

1.5 Objectives

This thesis is targeted at the issue of monitoring PI flow and enforcement of privacy regulations and policies in an enterprise's WSA infrastructure and trying to solve the three problems that the implementation of such a privacy system faces. We assume the WSA architecture in which all communication between the providers and requesters in a WSA infrastructure are in SOAP format. Furthermore, the assumption is made that databases are accessed by applications by utilizing web services and that EPAL is used to express the privacy policy.

Our goal is to prove the concept for the PI Monitor Agent, which is the central piece of the multi-agent architecture in paper "Privacy Compliance with Web Services" (Bodorik et al., 2009), in order to evaluate the feasibility of implementation. Furthermore, proof of the concept is also used to identify any missing components of the architecture and, if found, to enhance the architecture in order to bring out its full potential in supporting privacy compliance.

1.6 CONTRIBUTIONS

A prototype of the PI Monitor Agent concept was developed and it has lead to the identification of the following components that enhance the multi-agent architecture for enforcement of privacy:

- The PI Client Agent is added to the proposed architecture;
- User KB is added to KBs;
- An obligation enforcement component is added to the architecture;
- A database query component is added to the architecture;
- Interfaces of KBs are designed to provide information that is required by a PI Monitor Agent.
- A SOAP header format is presented to carry useful information to be used by the PI Monitor Agent to make decisions; Monitored Web Service List (MWSL) is introduced to the architecture.

1.7 OUTLINE

The rest of this document is organized as follows:

Chapter 2: Literature Review

In this chapter we discuss related research work on the protection of privacy information in computer systems.

Chapter 3: Privacy Architecture with Client Agent

In this chapter we describe the enhancements added to a specific multi-agent architecture for monitoring privacy and enforcing compliance. The importance of including these enhancements is also discussed.

Chapter 4: The Details of Architectural Components

In this chapter, we describe the information that is required by the PI Monitor Agent to make decisions. We then introduce the enhanced architecture and how it works to achieve the goal of monitoring privacy and enforcing privacy policy in a WSA environment. As the core components in the enhanced architecture, the PI Monitor Agent and PI Client Agent are also described in detail. Finally, we describe how the PI Monitor Agent interacts with other supporting components to achieve its goal.

Chapter 5: System Working Scenarios

In this chapter, we describe how the PI Monitor Agent and the PI Client Agent work, in different scenarios, to achieve the goal of monitoring PI flows and enforcing privacy policy in a WSA environment.

Chapter 6: Proof of Concept

In this chapter, we use simulated examples of web services and application client to present four cases demonstrating how the PI Monitor Agent, working with other supporting components in the architecture, works to provide privacy monitoring and privacy policy enforcement in a WSA environment.

Chapter 7: Implementation and Experimentation

This chapter describes how the PI Monitor Agent and PI Client Agent are implemented in ASP.NET. We then describe how the experiment environment is set up and how the performance of the PI Monitor Agent is measured. Finally, we analyze the results of the experiments and make suggestions on implementing the PI Monitor Agent in a real operational environment.

Chapter 8: Conclusion

This chapter concludes the paper.

Appendix

The Appendix provides all the SOAP templates and database design tables that are used in this thesis.

CHAPTER 2 LITERATURE REVIEW

Different countries may have different definitions for privacy (Cranor et al., 2006). As defined by Schoeman (1984) in the book *Philosophical Dimensions of Privacy: An Anthology*, privacy is a state or condition of limited access to a person. Privacy protection has recently drawn a lot of attention~~s~~ from both private and public sectors. This is simply because in the current telecommunications climate, as web services are booming on the Internet, they have become a very popular technology for information exchange for business-to-business and customer-to-business applications (Adams, C. and Barbieri, K. 2006).

A typical case is that in order to finish an online service provided to customers, the service provider may need to invoke a 3rd party organization's web service for the purpose of an information exchange to fulfill the customer's request. Another typical case is that when a customer requests a web service provided by a service provider, the service provider, in order to finish the service needs to collect private information from the customer. Transfering of this privacy information among different business units in the service provider may also be needed to finish the web service.

This increased attention has resulted in the introduction of regulations and legislations to this field. "In the USA, the Privacy Act of 1974 requires that federal agencies grant individuals access to their identifiable records that are maintained by the agency, ensure that existing information is accurate and timely, and limit the collection of unnecessary information and the disclosure of identifiable information to third parties" (Davis, 2000). Today, increased cross-border information flows, network information processing, use of federated systems, application outsourcing, social networks, ubiquitous devices and cloud computing bring greater challenges and management complexity to privacy risk management. To address these issues, the ISTPA (2009) has proposed a Privacy

Management Reference Model to increase the awareness in this area and promote the development of software tools and systems to manage privacy information.

With regard to privacy and information protection, many methods have been developed and proposed in this field. Standards were proposed to fasten the development of agents. For example, the Platform for Privacy Preferences (P3P) (Cranor et al., 2006) proposed by W3C, is designed to express organization's privacy practices to the end customers in XML. This design allows the information to be read by an automatic agent on behalf of the end customer and to compare the policy content with the end customer's privacy preferences that are specified in a privacy preference language such as a P3P Preference Exchange Language (APPEL). The P3P is not designed for tackling the enforcement of privacy policy in the organization's internal computing systems except for the purpose of expressing the privacy policy to external customers.

Research on the use of an ontology language to represent privacy preferences and contextual information has been examined by Gandon and Sadeh (2004), Rao et al. (2006) and Jutla et al. (2006). Jutla et al. (2006) also presented PeCAN, a multi-agent architecture for client side user privacy. Garcia and Toledo (2008) presented an ontology method to translate the privacy policies expressed in P3P vocabulary into assertions that are used to control access to private data.

To enable privacy protections within a business's processes, we also need a language that can be used to express privacy policies for internal computing systems. Enterprise Privacy Authorization Language (EPAL) (IBM, 2003) and DPAL [5] are proposed for this purpose. EPAL is an interoperability language developed by IBM to define enterprise privacy policies on private data handling practices according to fine-grained positive and negative authorization rights. The goal of EPAL is to provide an enterprise with a means of encoding its privacy-related data policies and practices in an XML format document, which can be imported and enforced by an enterprise's privacy enforcement system. However, EPAL does not present specific ideas on the design of a system aimed at the goal of enforcing privacy policy.

A lot of research has also been done on using database technologies to enhance privacy at the data level. Iyengar (2002) presented a technology to apply data anonymization technology to protect private information on the data subject level. Similar work can also be found in the paper by Kobsa and Schreck (2003). Song et al. (2006) presented pseudonym technology to be used to anonymize private data in their paper. According to the research of Dragovic and Crowcroft (2004), privacy risks can be mitigated through data manipulation. Agrawal et al. (2002) presented the concept of Hippocratic databases, which brings privacy protection to relational database systems. This system uses privacy metadata to store privacy policies and privacy authorizations. According to the research in the paper by Jajodia and Sandhu (1991) and the research in the paper by Sandhu and Chen (1998), multilevel secure relational databases can be used to design a fine-grained secure data model which can be used to achieve private data access control.

Since Adam and Worthmann (1989) presented the idea of using access control technology to achieve privacy information protection in databases in their paper, much research~~es~~ has been done on using access control technology to manage private data in a computing system. For example, Adams and Barbieri (2006) presented the idea of utilizing tools and technologies in access control to solve privacy and information protection problems. Byun et al. (2004 and 2005) presented a privacy preserving access control model based on the notion of purpose. Byun et al. (2006) also presented a new class of access control systems based on the notion of micro-view.

The Enterprise Privacy Architecture (EPA), proposed by Karjoth et al. (2002), is designed to achieve privacy compliance in an enterprise environment. However, this research did not provide many details about enforcement of the model. Parker (2005) proposed a methodology for creating and managing privacy compliance at a high level of abstraction dealing with organizations, procedural issues, systems and managing technological changes. Details of how the system should be designed were discussed in the Parker’s paper.

Bodorik et al. (2009) presented an adaptive agent-based enterprise information architecture to support enforcement of privacy policies on private information used by applications. The architecture contains KBs, which are used to store the knowledge for the PI Monitor Agent to make decisions. Its agents are able to self learn from the logs and databases to acquire knowledge in KBs. The architecture is placed in a WSA environment with SOAP messages as the information exchange format among applications and web services.

CHAPTER 3 PRIVACY ARCHITECTURE WITH CLIENT AGENT

3.1 INTRODUCTION

Based on the multi-agent architecture proposed by Bodorik et al. (2009), we plan to design an agent for privacy monitoring and enforcement in a WSA environment. The prototype development has led to the discovery of a need for a PI Client Agent. To make a decision on what action to take on a request to, or reply from a web service, the PI Monitor Agent needs to know information about the client that invokes the web service. To acquire such information, the PI Client Agent is added to the architecture to extend the functionality of the architecture to the client side. The development of the prototype has also led to the conclusion that the architecture proposed by Bodorik et al. (2009) needs to be expanded and enhanced to fully support privacy monitoring and privacy enforcement in a WSA environment. The enhancements to the architecture include:

- The PI Client Agent is added to the proposed architecture extending the architecture to the application clients to acquire the information required by the PI Monitor Agent;
- User KB is added to KBs;
- An obligation enforcement;
- Monitored Web Service List (MWSL) is introduced; and
- A privacy database query component.

Furthermore, interfaces of KBs are designed to provide information that is required by PI monitor. As well, a header format is presented to carry useful information to be used by the PI Monitor Agent to make decisions.

The enhanced architecture is diagrammatically shown in Figure 2.

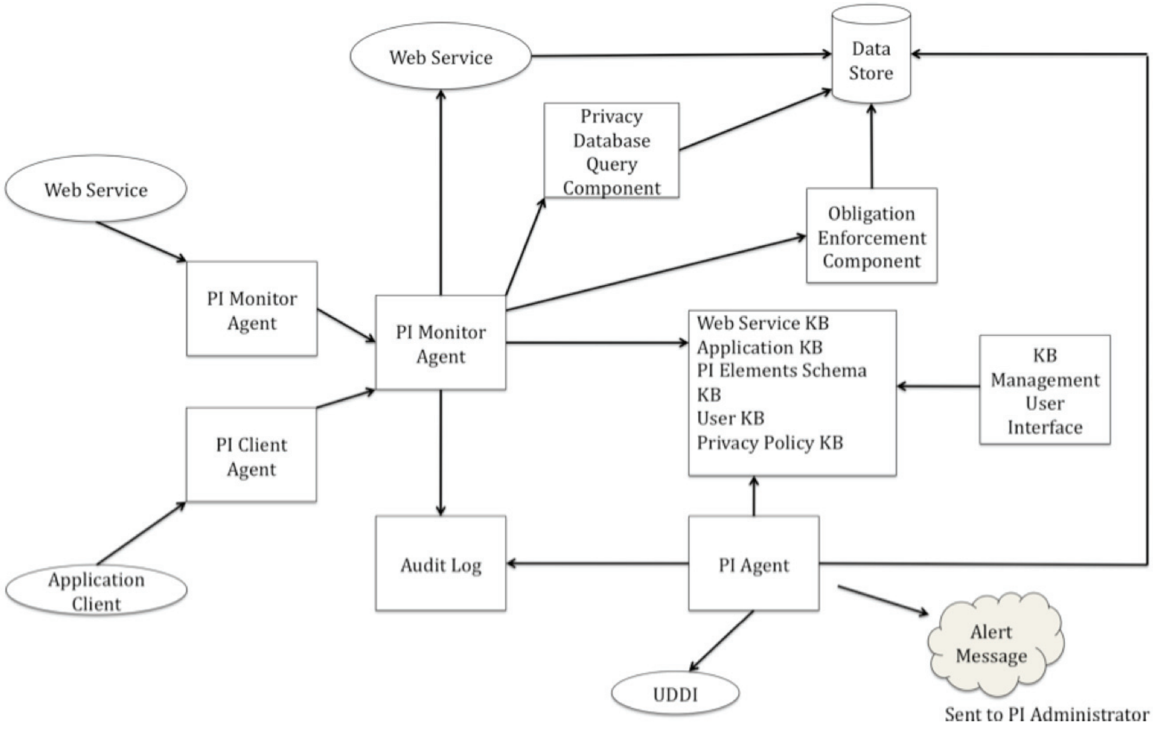


Figure 2 The enhanced architecture.

The following sections in this chapter describe briefly these enhancements and why they are needed. Further details regarding the architecture can be found in later chapters.

3.2 PI CLIENT AGENT

As implied by EPAL, the rules in a privacy monitoring and enforcement architecture should be able to express whether a user category is allowed to access a data category for a certain action with a certain purpose when some optional conditions are satisfied and the promised obligation/s is achievable. To construct an architecture that fully supports privacy monitoring and enforcement, a mechanism to acquire user information and some information about the application client is required in the architecture – hence the need for the PI Client Agent. The PI Client Agent is designed to collect, from the user or the operating system services, the following information about the client that invokes the web service:

- ID of the user who is using the application;
- Name of the process that invokes the web service.

Details of how this information is collected on the client are discussed in the next chapter.

3.3 USER KB

To evaluate privacy rules, the PI Monitor Agent needs to know about the user who caused the invocation of the web service and to which user category he or she belongs. User category is a group of users who share the same role in an organization's computing system from the privacy perspective. After acquiring the user ID from the client side, the PI Monitor Agent can determine the user category from the user ID using a mapping of the user ID's to user categories (for definition of user category please see section 4.1.1). A proposed User KB holds such mapping information.

During the creation of a User KB, user IDs are created with categories as their tags. A user ID may have multiple user category tags. After user ID is acquired, the PI Client Agent will insert the user ID into the header of the SOAP request message and when the PI Monitor Agent on the server side receives the SOAP request, the PI Monitor Agent will extract user ID from the SOAP header and then query the User KB about the user category tags that are associated with that user ID.

3.4 PRIVACY DATABASE QUERY COMPONENT

We now discuss the need for the Privacy Database Query Component, which is shown in Figure 2. As specified in EPAL, a privacy rule may contain some conditions that need to be satisfied before the PI Monitor Agent applies the rule (IBM, 2003). A condition, which is defined by the <condition> element in the <epal-policy> element of an EPAL policy, determines the conditions under which a <rule> element in an EPAL policy should be applied. The <condition> element is not an essential element in the <rule>

element. When a <rule> element does not have the <condition> element, the <rule> element does not have any pre-condition to be applied and evaluated. The <condition> element is returned to the PI Monitor Agent by Privacy Policy KB when the applicable <rule> element has a <condition> sub-element. For instance, consider a scenario where a customer's information is allowed to be accessed for marketing purposes only when the customer is over 18 years old and the customer agrees to allow his/her PI to be used for marketing purposes.

To check the conditions stated in privacy rules, The PI Monitor Agent needs to query the customer database to get the values of the attributes required in the condition/s. To do so, a component called the Privacy Database Query Component is added to query the customer databases to get the required attributes. How the Privacy Database Query Component interacts with the PI Monitor Agent will be discussed in the next chapter.

3.5 Obligation Enforcement Component

Figure 2 also shows the Obligation Enforcement Component. As specified in EPAL, privacy rules should also be able to specify obligations that need to be satisfied/executed by an organization after a certain action is permitted by the rule on the PI elements (IBM, 2003). Examples of obligations include retention of the data for a specified period of time or, perhaps, notifying the user. In an a-synchronized implementation of the architecture, whether the obligation is achievable needs to be checked before the PI Monitor Agent applies the applicable privacy rule.

In an EPAL policy, the obligation, which is required to be checked or executed before the application of the privacy rule, is stated in the <obligation> sub-element of the <rule> element. In our proposed architecture, the checking/execution of the obligation is done by the Obligation Enforcement Component.

3.6 PROPOSED SOAP HEADER FORMAT FOR PI MONITORING

As part of the proposed method for monitoring private data in a WSA environment, a recommended header format for SOAP messages is proposed. Private data monitoring and privacy enforcement systems use headers of SOAP messages to store information about the invoking side that might be used by the PI Monitor Agent of the invoked monitored web service. The invoking side could be an application client or another monitored web service. Depending on the types of the invokers, information inserted into the headers might be different. For instance, if the request is from an application client with a PI Client Agent, the header contains the user ID, process name, and the invoked web service. If, on the other hand, the request comes from another monitored web service the header contains: the invoking web service name; the invoked web service; information indicating whether the request has been checked by a PI Monitor Agent on the invoking side; information on whether the response is static or dynamic. Note that static and dynamic web services will be discussed later in section 4.1.2.

The information acquired on the invoking side is used by the PI Monitor Agent to make a decision on whether to authorize the access to private data. How the PI Monitor Agent uses the header information will be discussed in detail~~s~~ in Chapter 5. Figure 3 shows the format of the header.

<table>
<tr><td colspan="2">User-ID</td><td colspan="2">Process-Name</td><td colspan="2">Invoking-Web-Service</td></tr>
<tr><td colspan="2">Invoked-Web-Service</td><td colspan="4">Flag-Request-Checked</td></tr>
<tr><td colspan="6">Flag-Static-Response</td></tr>
<tr><td colspan="6">SOAP Body</td></tr>
</table>

▨ SOAP Header

□ SOAP Body

Figure 3   A proposed SOAP header format for monitoring PI.

User-ID: The ID of the user of the application client.

Process-Name: The name of the process that made the request.

Invoking-Web-Service: The name of the web service that made the request.

Invoked-Web-Service: The name of the web service that is invoked.

Flag-Request-Checked: Set to "1" if this message has been checked by the PI Monitor Agent.

Flag-Static-Response: Set to "1" if the response of the invocation is static.

The following is a template for the PI monitoring SOAP header

```
<?xml version="1.0"?>
<soap:Envelope
 xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:header>
      <User-ID/>
      <Process-Name/>
      <Invoking-Web-Service/>
      <Invoked-Web-Service/>
      <Flag-Request-Checked/>
      <Flag-Static-Response/>
</soap:header>
<soap:Body>
………………
</soap:Body>
</soap:Envelope>
```

3.7 MONITORED WEB SERVICE LIST

The Monitored Web Service List (MWSL) is contained in the Web Service KB and is maintained by the PI administrator. Figure 4 shows a typical WSA environment, where web services and application clients are distributed across the network domain.

Application client in this thesis means the client software used for an application of a web service. Data flows travel in the form of SOAP messages on the network among web services and application clients. MWSL is a list of web services that need to be monitored for compliance with the organization's privacy policy rules. To have full coverage of monitoring of PI flows in an organization's network domain, it is recommended all the web services of an organization be included in the MWSL. As a result, PI monitors will be deployed to all the web services of the organization. The PI Monitor Agent runs as an add-on component to the web service (by using either SOAP extensions or Java Axis) inspecting requests and responses to or from their resided MWSL web services.

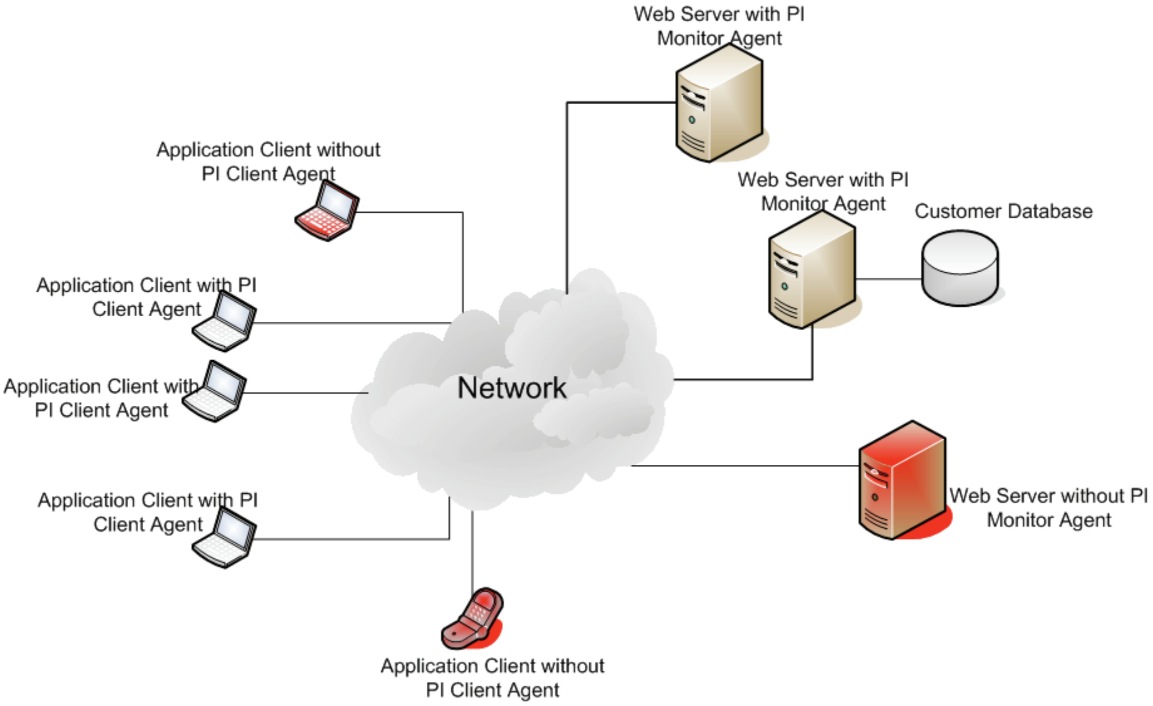

Web Server with PI Monitor Agent
Application Client without PI Client Agent
Web Server with PI Monitor Agent
Customer Database
Application Client with PI Client Agent
Application Client with PI Client Agent
Network
Application Client with PI Client Agent
Web Server without PI Monitor Agent
Application Client without PI Client Agent

Figure 4 A typical WSA environment with the PI Monitor Agent and the PI Client Agent deployed.

The MWSL is encoded in XML and is pushed to the PI Monitor Agents and the PI Client Agents from the Web Service KB component. Once there is a change in the MWSL in Web Service KB (such as a new web service is added to the MWSL or a web service is removed from the MWSL), an update from the Web Service KB is pushed to the PI Monitor Agents and the PI Client Agents. For efficiency considerations, a differential update is used. In later discussions in this thesis, a monitored web service means a web service with a PI Monitor Agent and an unmonitored web service means a web service without a PI Monitor Agent.

There are scenarios where application clients without PI Client Agents and unmonitored web services are involved in data communications/exchanges with monitored web services. These scenarios are also explored in this thesis.

CHAPTER 4 THE DETAILS OF ARCHITECTURAL COMPONENTS

Based on the findings regarding the information that is required by the PI Monitor Agent, an enhanced version of the architecture in the paper Privacy Compliance with Web Services (Bodorik et al., 2009) has been developed for monitoring the PI flows and enforcing of privacy policies in a WSA environment. The core component of this enhanced architecture is a PI Monitor Agent. A PI Client Agent is also added to the architecture to provide the PI Monitor Agent with information regarding the client. In this Chapter, we first describe the information required by the PI Monitor Agent to make decisions; then talk about how the architecture works; then describe details about the PI Monitor Agent and PI Client Agent; and, finally, describe the information and services other supporting components in the architecture provide to the PI Monitor Agent. We also define interfaces of these components to the PI Monitor Agent.

How other supporting components work is out of the scope of this thesis.

4.1 INFORMATION REQUIRED BY THE PI MONITOR AGENT

EPAL (Enterprise Privacy Authorization Language) is an inter-system interoperability language developed by IBM to define enterprise privacy policies on the private data handling practices according to fine-grained positive and negative authorization rights (IBM, 2003). The goal of EPAL is to provide a means of encoding an enterprise's privacy-related data policies and practices in an XML format document, which can be imported and enforced by an enterprise's privacy enforcement system. As a result of selecting EPAL, its properties determine what information is needed in a privacy monitoring and enforcement system. This section is going to describe the information that will be needed by a PI Monitor Agent to make decisions based on privacy policies represented using EPAL.

4.1.1 Implications on the Architecture Due to EPAL

To achieve the goal of monitoring PI flows and enforcing privacy policies in a WSA environment, EPAL contains the following information expressed in a policy rule. An EPAL's policy rule expresses whether a User Category, to which the user of PI belongs, is allowed to access a Data Category, for a certain action, with a certain purpose, when some specified, but optional, conditions are satisfied and the obligation/s, specified by the rule, are achievable. In an EPAL privacy policy, this is expressed in a rule element by defining the following six sub-elements: <user-category>, <data-category>, <purpose>, <action>, <condition> and <obligation>.

To evaluate a rule, the PI Monitor Agent will work with other supporting components in the proposed architecture to find the following five properties regarding a web service request/response:

- User Category
- Data Category
- Purpose
- Action
- Condition

Thus, when PI data is referenced in a web services request/response, the PI Monitor Agent determines the above properties for that request/response. Based on these properties, the PI Monitor Agent is able to retrieve the rule element applicable to this request/response. The above properties are described below:

User Category

The User Category is a group of users who share the same role in an organization's computing system from a privacy perspective. As mentioned in Chapter 3, the User Category is determined by user KB by a mapping of user IDs to User Categories.

Data Category

The Data Category is defined from a privacy perspective in that it is a set of PI elements that have the same privacy implications, in that the same privacy rule applies to the data elements in a category. Mappings of sets of data objects to a Data Category are stored in PI Elements Schema KB. The knowledge in PI Elements Schema KB in an enterprise is acquired by using a PI Agent to mine data stores used by that enterprise. When mining of data stores is finished, private data objects are abstracted from data stores. PI elements are defined in a PI elements dictionary that maps data objects to PI elements. The privacy administrator then defines different Data Categories and maps the sets of PI elements to the defined Data Categories.

Purpose

Purpose defines the objective of the usage of PI elements. The Purpose can be obtained from Web Service KB or User KB based on different scenarios.

Action

Action defines the action taken on the PI elements by the recipient of the PI elements. The Action can be obtained from Application KB or Web Service KB.

Condition

Condition refers to the pre-conditions defined by the <condition> element in <epal-policy> element of an EPAL policy. It determines the conditions under which a <rule> element in an EPAL policy should be applied. The <condition> element is an optional element in the <rule> element. When a <rule> element does not have a <condition> element, it means this <rule> element does not have any pre-condition to be applied. A <condition> element is returned to the PI Monitor Agent by Privacy Policy KB when the applicable <rule> element has a <condition> sub-element referencing to a <condition> element. If the <condition> sub-element in the applicable <rule> element is empty, which means there is no precondition to the application of the <rule> element, the Privacy Policy KB will not return any <condition> element. The PI Monitor Agent will receive an empty <condition> sub-element in the returned <rule> element and apply it without checking any precondition.

4.1.2 Static or Dynamic

Another piece of information that is required by the PI Monitor Agent is whether the request/response is static or dynamic. It might be known to the architecture that for a certain web service the PI elements contained in the request to or response from that web service will not change over different invocations. In such situations, KBs contains knowledge regarding the data category/ies that are contained in the request or response of the web service. Therefore, the PI Monitor Agent need not parse through the SOAP message body to find the PI elements the message contains. The Data Category can be acquired immediately by querying Web Service KB. To support this feature, we define ***static*** requests/responses and ***dynamic*** requests/responses.

If the request/response between a requester and an invoked web service contains no PI elements or the same PI elements every time the requester invokes the web service, the request/response is considered to be ***static***; otherwise, it is considered to be ***dynamic***.

The information on whether the request/response is static or dynamic is stored in Web Service KB.

4.2 ARCHITECTURAL OVERVIEW

In paper Privacy Compliance with Web Services (Bodorik et al., 2009), an agent-based architecture is presented to enforce privacy policies on private information in a WSA environment. Based on the development of the prototype of the PI Monitor Agent, the architecture is enhanced to fully support the goal of monitoring the PI flow and enforcing the privacy policy in a WSA environment. Among these enhancements, some components have been added to the architecture. Recall that the newly added components include:

1. A PI Client Agent is added to the architecture to collect required information from the client side, extending the functionality of the architecture to the client side;

2. User KB is added to provide mappings from user IDs to User Categories;
3. A privacy database query component is added to the architecture for the purpose of getting the required context data by the condition element in EPAL policy; and
4. An obligation enforcement component is added to the architecture to provide enforcement mechanism to the architecture.

This section provides an overview of the modified architecture and describes how the PI Monitor Agent in this architecture works and where it finds the information that is required to fulfill its tasks. Other supporting components and their interactions with the PI Monitor Agent are also discussed in Section 4.3.

As shown in Figure 2, the proposed private data monitoring and privacy policy enforcement architecture consists of the following twelve components: PI Monitor Agent, PI Client Agent, Web Service KB, Application KB, PI Elements Schema KB, User KB, Privacy Policy KB, Audit Log, PI Agent, Privacy Database Query Component, Obligation Enforcement Component and KB Management User Interface. Among these components, Web Service KB, Application KB, PI Elements Schema KB, User KB, Privacy Policy KB, Privacy Data Base Query Component and Obligation Enforcement Component provide support to the PI Monitor Agent. They provide the PI Monitor Agent with the information it needs to make decisions or provide an obligation mechanism to the architecture. Details regarding these components and how they interact with the PI Monitor Agent will be discussed in Section 4.3.

In WSA infrastructures, PI flows through requests and responses to/from web services. Inspections of PI elements in requests to and responses from web services are handled by PI Monitor Agents. These Agents run as an add-on component on the monitored web services, working with KBs and the Privacy Database Query component to determine whether the request or response of a web service should be allowed through or dropped based on the organization’s privacy policy rules.

After the PI Monitor Agent gets the applicable <rule> element from Privacy Policy KB, if there is any <obligation> sub-element in the returned <rule> element, the PI Monitor Agent will send the required obligation/s to the Obligation Enforcement Component to fulfill the stated obligations in the privacy policy rule. When the obligations are completed by the Obligation Enforcement Component, the Obligation Enforcement Component will return a "success" or "failure" message to the PI Monitor Agent indicating whether the Obligation Enforcement Component is successful or unsuccessful. If the Obligation Enforcement Component is successful, the PI Monitor Agent will take the action (allow or drop) as stated in the ruling attribute of the returned <rule> element.

After the PI Monitor Agent takes an action on the request/response message, it will log the message in the Audit Log component. Logs generated by PI Monitors or Monitor Agents are stored in the Audit Log for the following purposes:

1. The PI Agent in the architecture data mines logs in the Audit Log to generate KBs. This process is needed when the system first gets initiated in an organization, so the system is able to learn the WSA environment around it and generate the knowledge that will be stored in KBs. This knowledge, generated from data mining, is stored in KBs and will be needed by the PI Monitor Agent to make decisions. The generated knowledge includes:
    a. Data Categories contained in invocations of web services;
    b. Purposes of recipients of PI elements. By data mining logs, the PI Agent is able to find User Categories, process names, web services that are involved in the PI flows (messages from/to web services). With human PI Administrator's involvement, purposes of PI flows are determined and stored in KBs;
    c. Actions taken on PI elements by the recipients of the PI elements. By data mining logs, the PI Agent will find User Categories, process names and web services that are involved in the flow of PI elements. With human PI administrator's involvement, actions of the application client or web service on private data will be determined and stored in KBs; and

d. Whether a request/response is static or dynamic. the PI Agent data mines logs in the Audit Log to determine if the request and response between a requester and provider is static or dynamic in the perspective of private data contained in the request and response. Once this information is generated, it will be stored in KBs.

2. The PI Agent compares logs in the Audit Log with the information in KBs to verify and correct knowledge in KBs. If the PI Agent finds the knowledge contained in KBs is not accurate against the information in logs or if it needs to correct KBs to reflect the latest status, the PI Agent will issue a notification email to the privacy administrator so that the privacy administrator can investigate and manage the correction.

Both how to mine logs contained in the Audit Log to generate KBs and how to compare logs with knowledge in KBs to verify and correct knowledge in KBs are out of the scope of this thesis. In this thesis we assume that the KBs are already available and the mechanism to compare logs in the Audit Log with the knowledge in KBs to verify and correct knowledge in the KBs is also already available.

PI Client Agents are extensions of the enhanced architecture on the application client side. They are deployed to workstations with application clients that need to access monitored web services (web services with PI monitors). Details about the PI Client Agent will be discussed in section 4.5 PI Client Agent.

The PI Agent in the architecture is designed to assist the PI Administrator by performing tasks that can be automated:

- Mine logs in the Audit Log to generate knowledge in KBs;
- Compare logs in the Audit Log with knowledge in KBs to verify and correct information in KBs; if there is any difference, the PI Agent will send a notification email to the PI Administrator;

- Analyze UDDI to acquire information regarding web services and store them in the Web Service KB; and
- Mine data stores of an organization to generate a dictionary for mapping the data objects that are found in data stores to PI elements and stores the PI elements dictionary in the PI Element Schema KB.

The details of how the PI Agent works to fulfill its tasks are out of the scope of this thesis.

The KB Management User Interface is designed to provide a GUI for the PI Administrator to manage KBs in the architecture. On some occasions, the PI Administrator needs to get involved in the creation of KBs. For instance, when receiving a notification that a difference between the logs and KBs has been found by the PI Agent or when there is a requirement from the senior management to change the privacy policies, the PI Administrator will get involved to investigate and modify KBs accordingly. The details of how the KB Management User Interface is designed are out of the scope of this thesis.

The following sections are going to focus on the PI Monitor Agent and PI Client Agent. Details about KBs and how they interact with the PI Monitor Agent can be found in section 4.3 Architectural Components and Interactions.

4.3 PI MONITOR AGENT

In this thesis, we assume that all the databases in an organization are accessed through web services, that the execution of a web service is requested by sending the web service a SOAP request message and that once the execution of the requested web service is finished, the requested web service will return a SOAP response message.

In a WSA environment, data flows from/to web services through SOAP requests and responses. Among the data transferred on the wire, some are PI elements. PI Monitor Agents are deployed to web services that need to be monitored according to Web Service

KB. They run as an add-on component in the monitored web service to check requests and responses from/to the monitored web services. When there is a SOAP message going to or coming from a monitored web service, the PI Monitor Agent of the monitored web service will parse the SOAP message, find required information for evaluation of the EPAL rule elements and make a decision on what action to take according to applicable EPAL rule elements.

Based on the architecture proposed in this thesis, the working mechanism of the PI Monitor Agent is also designed to achieve the goal of monitoring private data, authorizing private data access and enforcing obligations in WSA. When a SOAP message comes to or is sent from a monitored web service, the PI Monitor Agent, on the monitored web service platform, will perform the following tasks:

1. If the message has been checked by another PI Monitor Agent, as determined by the header, the PI Monitor Agent will pass it along without any further action. If not, the PI Monitor Agent will find the following five properties, needed to evaluate EPAL policy rules, regarding the SOAP message:

    - User Category
    - Data Category
    - Purpose
    - Action
    - Condition

    To assist the PI Monitor Agent to find the above information, the PI Client Agent or another PI Monitor Agent on the invoking side might provide the following information in the SOAP header:

    - The user ID of the user that uses the application client of the web service;
    - The name of the process that invoked the web service;

   - The invoked web service;
   - The invoking web service;
   - A flag indicating that this request message has been checked; and
   - A flag indicating the response is static or dynamic.

   Based on the information in the SOAP header and the information contained in the SOAP body, the PI Monitor Agent will query KBs to find the required five properties for the evaluation of EPAL rule elements. More details can be found in Chapter 5.

2. Queries Privacy Policy KB about the applicable EPAL rule element. Privacy Policy KB will return the found EPAL <rule> element together with the <condition> element if the <condition> sub-element is not empty.

3. If there is any condition that needs to be evaluated before the application of the returned EPAL rule element, the PI Monitor Agent will check with the Privacy Database Query Component to find the needed data attribute values for evaluating <condition> elements.

4. Make a decision based on the returned EPAL rule element. Possible actions include allow and drop.

5. This step is optional. If the PI Monitor Agent is on the invoking side, it will put the following information in the header of the SOAP request:
   - A flag indicating the response is static or dynamic.
   - The invoked web service.
   - The invoking web service.
   - A flag indicating that the request has been checked.

6. Log SOAP messages. For system benchmarking and auditing purposes, the PI Monitor Agent will log SOAP requests/responses in the Audit Log component.

> Information about how to log is contained in the Web Service KB. It depends on what web service is being called. By default, the PI Monitor Agent will log the request or response message with the following details: the SOAP message (with headers), the time when the message is received, the source of the message, the destination of the message, the User Category, the Data Category, the purpose, the action, the PI Monitor Agent's decision and the applied EPAL rule elements name.

A more detailed description on how the PI Monitor Agent works will be discussed in several scenarios in Chapter 5.

In Chapter 6, multiple example cases were examined under the following different circumstances of PI flows:

1. An Application Client with a PI Client Agent invokes a monitored web service (web services with a PI Monitor Agent);
2. A monitored web service invokes another monitored web service;
3. A monitored web service invokes an unmonitored (web services without a PI Monitor Agent) web service;
4. An unmonitored web service invokes a monitored web service; and
5. An application client without PI Client Agent invokes a monitored web service.

In ASP.NET infrastructure, the PI Monitor Agent is implemented by using SOAP Extension Classes. More details regarding how the PI Monitor Agent is implemented in ASP.NET can be found in Chapter 7.

4.4 MULTIPLE PI MONITOR AGENTS

As shown in Figure 5, in WSA environments there are cases in which, in order to finish invoker A's request, the invoked Web Service B needs to invoke another Web Service C. In these cases, there might be more than two PI Monitor Agents involved in the invocation and PI elements may vary among each invocation. In the architecture presented in this thesis, in cases when more than two PI Monitor Agents are involved in invocations, the PI Monitor Agent on each web service only needs to be aware of the PI elements coming in or going out of the resided web service. Inspections of PI flows on the subsequent or parental web services will not be handled by the PI Monitor Agent on the current web service.

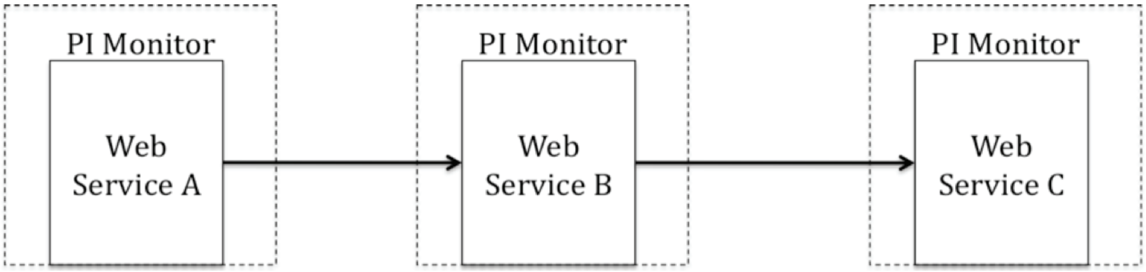


Figure 5 Multiple PI Monitor Agents.

For instance, the PI Monitor on Web Service A need not to be aware of the PI flows in the invocation between Web Service B and Web Service C.

4.5 PI CLIENT AGENT

As described in the section on the PI Monitor Agent, some information is required by the PI Monitor Agent to make a decision on a SOAP request/response. When the invocation of the monitored web service is from an application client, to monitor private data and authorize private data access between this monitored web service and the invoking application client, the PI Monitor Agent needs to acquire some information regarding the client side such as the user ID of the user of the application client and the name of the process that made the invocation. This information is not available in KBs and can only be acquired from the client side. PI Client Agents are extensions of the private data monitoring and privacy policy enforcement architecture on the client side. They gather information from an application client and insert the gathered information into the header of the SOAP request message. PI client agents are deployed to work stations with application clients to complete the following functions:

1. Authenticate users of the application client and insert user IDs into headers of SOAP requests. PI Client Agents keep a copy of the monitored web service list, which is synchronized with the copy in Web Service KB. If an application client is invoking a web service that is in the monitored web services list, the PI Client Agent on that workstation will prompt a user authentication. Once the authentication is finished, the PI Client Agent will insert the user ID into the header of the SOAP request message.

2. Find the name of the process that made the SOAP request and insert the found process name in the header of the SOAP request message. If an application client is invoking a monitored web service, the PI Client Agent on this application client will find out the process that made this call and insert the process name in the header of the SOAP request message.

4.6 ARCHITECTURAL COMPONENTS

As shown in Figure 2, for the PI Client Agent and the PI Monitor Agent to work properly, supporting components need to be established in the architecture providing the needed information and services to the PI Monitor Agent and the PI Client Agent. The supporting components are:

- Web Service KB
- User KB
- Application KB
- PI Elements Schema KB
- Audit Log
- Privacy Policy KB
- Obligation Enforcement Component
- Database Query Component

These components provide web service interfaces to the PI Monitor Agent and the PI Client Agent. PI Monitor Agents and PI Client Agents can interact with them through

SOAP messages. Following sub-sections are descriptions of the supporting components and their interfaces provided to PI Monitor Agents and the PI Client Agents.

4.6.1 Web Service KB

Web Service KB contains information to map web services to user categories. The Web Service KB also contains information to map pairs of requesters and web services to purposes and actions. Requesters could be processes or web services. When web services are recipients of PI elements, web services become users of these PI elements. In order to evaluate EPAL, PI Monitor Agents need to know the User Categories that these web services belong to, the purposes of these web services and their actions.

Web Service KB also contains information to determine whether a request or response between a requester and a web service is static or dynamic. The requester could be a process or another web service. If the request or response is static, Web Service KB also contains information about the Data Categories contained in this static request or static response. For unmonitored web services (web services without a PI Monitor Agent) invoking monitored web services (web services with a PI Monitor Agent), Web Service KB contains information to determine if the request/response is static based only on the invoked web service.

For unmonitored services and applications without PI Client Agents invoking monitored web services, Web Service KB contains information to determine User Categories, purposes and actions for both request and response based only on the invoked monitored web services.

Web Service KB also contains information of what to log regarding the invocation of a specific web service.

As shown in Figure 6, Web Service KB interacts with PI Monitor Agents and PI Client Agents. When the PI Monitor Agent needs to get information from Web Service KB, it will request Web Service KB by calling “GetWSKBProperty” web method that Web Service KB provides to the PI Monitor Agent.

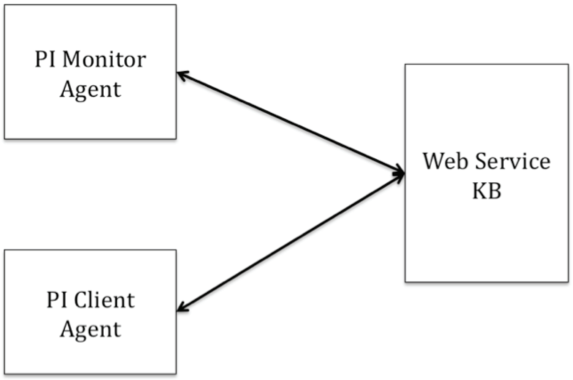


Figure 6 Web Service KB's interaction with the PI Monitor Agent and the PI Client Agent.

Web Method GetWSKBProperty
xmlns: http://example.com/WSKB/

Service Name: GetWSKBProperty

Web method GetWSKBProperty requires the following input data:

- <invoking_web_service_name or process_name>
- <invoked_web_service_name>
- <request_or_response> 0 for request, 1 for response

Web method GetWSKBProperty returns the following output data in a SOAP message:

- <user_category>
- <purpose>
- <action>
- <whether_request_is_static> if true, values 0
- <data_category_for_request> Null if request is not static
- <whether response is static> if true, values 0
- <data_category_for_response> Null if response is not static

A template of the request and reply between a PI Monitor Agent and the Web Service KB can be found in Appendix A.

Push of the Monitored Web Services List from Web Service KB
When there is a change in the monitored web service list in the Web Service KB, Web Service KB will push an update to PI Monitor Agents and PI Client Agents to reflect the change. However, Web Service KB also provides a web service interface for PI Monitor Agents and PI Client Agents to download the monitored web services list. Before the update or download happens both parties will exchange a "shared secret" to make sure each one of them is a legitimate and authorized partner.

4.6.2 User KB

With the PI Client Agent, the private data monitoring and privacy policy enforcement architecture is able to identify users on the application client side by authenticating users. The PI Monitor Agent will need support from User KB, which contains information to map user IDs to User Categories. Also, purposes of the usage of PI elements can be determined based on cases of a specific user using a specific process to retrieve PI elements. The User KB contains information to map a specific user using a specific process to purposes. Based on the user ID and the process the user is using, User KB is able to find the purpose of the user's action on the private data.

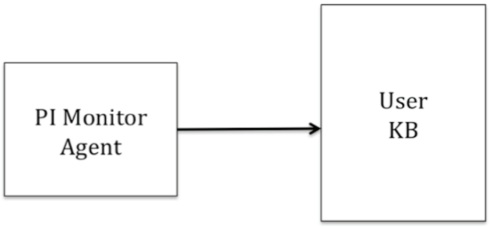


Figure 7 User KB's interaction with the PI Monitor Agent.

As shown in Figure 7, User KB interacts with the PI Monitor Agent through web method "GetUKBProperty".

Web Method GetUKBProperty

xmlns: http://example.com/UKB/

Service Name: GetUKBProperty

Web method GetUKBProperty requires the following input data:

- <user_ID>
- <process_name>

Web method GetUKBProperty returns the following output data:

- <user_category>
- <purpose>

A template of the request and reply between a PI Monitor Agent and the User KB can be found in Appendix B.

4.6.3 Application KB

When a user is using an application client to retrieve private information from a monitored web service, the PI Monitor Agent needs to find out the action the application client takes on the private information. Application KB contains information to identify the action based on the invoked monitored service and process name.

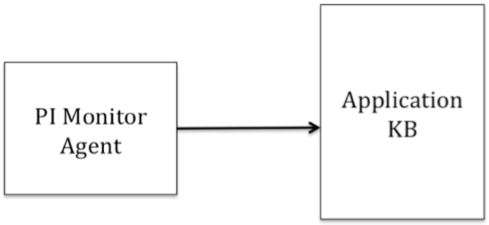


Figure 8 Application KB's interaction with the PI Monitor Agent.

As shown in Figure 8, Application KB can be queried by calling web method “GetAKBProperty”.

Web Method GetAKBProperty

xmlns: http://example.com/AKB/

Service Name: GetAKBProperty

Web method GetAKBProperty requires the following input data:

- <process_name>
- <invoked_web_service>

Web method GetAKBProperty returns the following output data:

- <action>

A template of the request and reply between a PI Monitor Agent and the Application KB can be found in Appendix C.

4.6.4 PI Elements Schema KB

When a request/response is dynamic, the PI Monitor Agent will parse the SOAP message to abstract the data objects contained in the SOAP message body, send them to PI Elements Schema KB to get the applicable data category/ies. After receiving this query, the PI Elements Schema KB is able to identify the PI elements from the queried data objects set and based on the found PI elements identify the applicable data category/ies. To fulfill these tasks, the PI Elements Schema KB contains the following information:

1. A dictionary for private data; and
2. Information to map sets of private data elements to data categories.

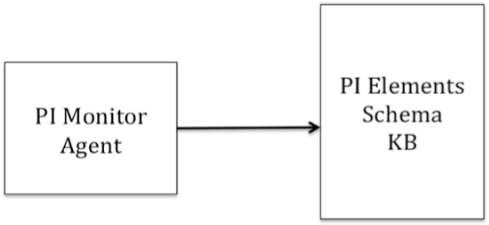


Figure 9 PI Elements Schema KB's interaction with the PI Monitor Agent.

As shown in Figure 9, the PI Monitor Agent requests the PI Elements Schema KB by calling the following web method.

Web Method GetPIESKBProperty

xmlns: http://example.com/PIESKB/

Service Name: GetPIESKBProperty

Web method GetPIESKBProperty requires the following input data:

- A set of <data_item> that are found in SOAP message body

Web method GetPIESKBProperty returns the following output data:

- <data_category>

A template of the request and reply between a PI Monitor Agent and the PI Elements Schema KB can be found in Appendix D.

4.6.5 Audit Log

Audit Log stores logs generated by the PI Monitor Agents. When SOAP requests and replies pass through a PI Monitor Agent, the PI Monitor Agent makes a decision on whether to allow or drop this message. After the decision is made, the PI Monitor Agent will send the Audit Log a SOAP message, which contains the following information to be logged in Audit Log:

- The SOAP message:
  SOAP messages with their headers are logged.

- The time when the message is received:
  This is the time when the message is received by the PI Monitor Agent. Time will be converted by the PI Monitor Agent into UTC time to eliminate the difference between time zones that might be caused by the geographic distribution of the organization’s web services.

- The source of the message:

This is the name of the source of the message. If the message is a request, this field is the requester's name. If the message is a response, this field is the provider's name.

- The destination of the message:
  This is the name of the destination of the message. If the message is a request, this field is the provider's name. If the message is a response, this field is the requester's name.

- User Category:
  User Category/ies that is found by the PI Monitor Agent regarding the usage of the message.

- Data Category:
  Data Category/ies that is found by the PI Monitor Agent regarding the usage of the message.

- Purpose:
  Purpose/s that is found by the PI Monitor Agent regarding the usage of the message.

- Action:
  Action/s that is taken by the recipient of the message to the PI elements.

- PI Monitor Agent's decision:
  Action taken by the PI Monitor Agent on the message. It is either drop or allow.

- Applied EPAL rule element:
  The rule element returned from the Privacy Policy KB to the PI Monitor Agent.

After receiving the SOAP request from a PI Monitor Agent, the log information contained in the SOAP message will be abstracted by Audit Log and stored in a log database. Then the Audit Log will return success or failure to the PI Monitor Agent.

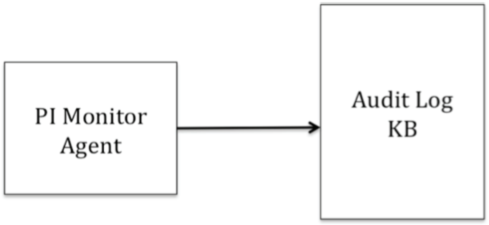


Figure 10 Audit Log KB's interaction with the PI Monitor Agent.

As shown in Figure 10, the Audit Log KB can be queried by the PI Monitor Agent by calling the following web method.

Web Method GetALKBProperty
xmlns: http://example.com/ALKB/
Service Name: GetALKBProperty

Web method GetALKBProperty requires the following input data:

- the SOAP message (with headers),
- the time when the message is received,
- the source of the message,
- the destination of the message,
- the User Category,
- the Data Category,
- the Purpose,
- the Action,
- the PI Monitor Agent's decision, and
- the applied EPAL rule element.

Web method GetALKBProperty returns the following output data:

- Log is success

SOAP templates of the request and reply between a PI Monitor Agent and the Audit Log KB can be found in Appendix E.

4.6.6 Privacy Policy KB

Privacy Policy KB stores the EPAL privacy policies. After the PI Monitor Agent acquires all the information that is required to evaluate EPAL rule elements, PI Monitor Agent will send all the following information to Privacy Policy KB:

- User category/ies
- Data category/ies
- Purpose/s
- Action/s

After receiving the query from a PI Monitor Agent, Privacy Policy KB will look up in its rule bases and return the applicable <rule> element in EPAL policy and <condition> elements referenced in <rule> element.

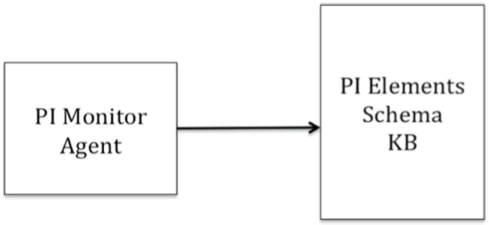


Figure 11 Privacy Policy KB's interaction with the PI Monitor Agent.

As shown in Figure 11, the Privacy Policy KB can be queried by the PI Monitor Agent by calling the following web method.

Web Method GetPPKBPolicy
xmlns: http://example.com/PPKB/
Service Name: GetPPKBPolicy

Web method GetPPKBPolicy requires the following input data:

<user_category>

<data_category>

<purpose>

<action>

Web method GetPPKBPolicy returns the following output data:

<rule> element in EPAL policy

<condition> element referenced in <rule> element

SOAP templates for the request and reply between a PI Monitor Agent and the Privacy Policy KB can be found in Appendix F.

4.6.7 Obligation Enforcement Component

The obligation element in the EPAL policy specifies the additional required actions the enterprise is obligated to take when a certain action is taken on the private data. After the PI Monitor Agent determines which rule element to use, it will find in the rule element the required obligation/s to take if there is any. The PI Monitor Agent will send the required obligation/s to Obligation Enforcement Component, which then will execute the required obligation/s and return “success” or “failure” messages to the PI Monitor Agent. If the execution of the obligations is successful, the PI Monitor Agent will take the action stated in the <action> sub-element in rule element. If the execution of the obligation/s is not successful, the PI Monitor Agent will drop the message.

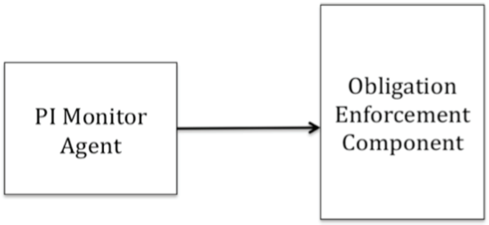


Figure 12 Obligation Enforcement Component's interaction with the PI Monitor Agent.

As shown in Figure 12, the PI Monitor Agent invokes the Obligation Enforcement Component by calling the following web method.

Web method RequestOEC
xmlns: http://example.com/OEC/
Service Name: RequestOEC

Web method RequestOEC requires the following data as input:

- <obligation> element in <rule> element

Web method RequestOEC returns the following data to PI monitor

- Success or failure

SOAP templates for the request and reply between a PI Monitor Agent and the Obligation Enforcement Component can be found in Appendix G.

4.6.8 Privacy Database Query Component

The Privacy Database Query Component keeps the information on which database contains the data variable/s required to evaluate the condition element in EPAL policy. In EPAL policies, the condition element references to the container element that defines the data structure of context data that will be used in the condition element. The Privacy Database Query Component provides an interface for the PI Monitor Agent to query data that is required to evaluate the condition element. Once receiving a query from the PI Monitor Agent, the Privacy Database Query Component determines which data base needs to be queried for the context data and gets the data from the database.

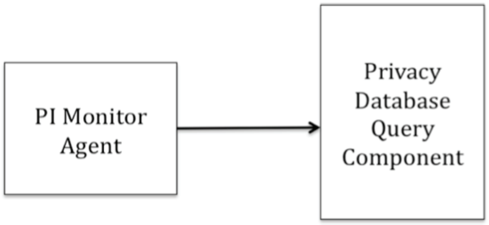


Figure 13 Privacy Database Query Component's interaction with the PI Monitor Agent.

As shown in Figure 13, the PI Monitor Agent invokes the Privacy Database Query Component by calling the following web method.

Web method RequestDQC
xmlns: http://example.com/PDQC/
Service Name: RequestPDQC

Web method RequestPDQC requires the following input:

- <container refid=
  attribute refid=> Data attribute/s in <condition> element
- <Unique_data_subject_identifier> such as CustomerID

Web method RequestPDQC returns the following output:

- <container refid= attribute refid= >data value</container refid= attribute refid= >

The unique identifier for the data subject is required to identify the data subject for which the context data is queried. If the unique identifier is missing in the SOAP message, then the condition cannot be satisfied and, therefore, the PI Monitor Agent will ignore the rule element in question.

SOAP templates for the request and reply between a PI Monitor Agent and the Privacy Database Query Component can be found in Appendix H.

CHAPTER 5 SYSTEM WORKING SCENARIOS

This chapter illustrates how the PI Monitor Agent and the PI Client Agent work together, with the support of the other architectural components, to achieve the tasks of monitoring private data and enforcing privacy policies in a WSA environment. The scenarios are designed to reflect the possible PI flows in a typical WSA environment.

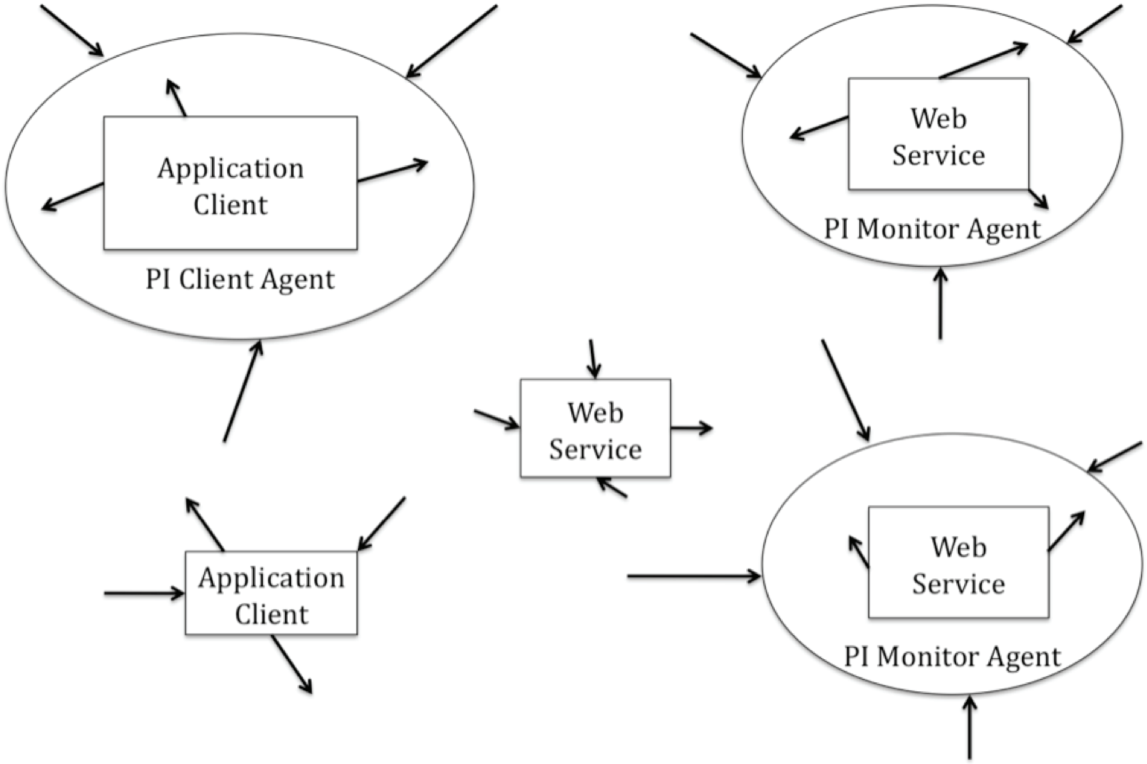


Figure 14 Possible PI flows in a typical WSA environment.

Figure 14 shows the possible PI flows in a typical WSA environment for an enterprise's business. It has application clients and web services distributed across the network domain. Some web services are monitored web services and some are not while some application client platforms have PI Client Agents and some have not. There are five possible scenarios regarding PI flows in such a web services and application environment:

1. An application client with a PI Client Agent invokes a monitored web service;
2. A monitored web service invokes another monitored web service;
3. A monitored web service invokes an unmonitored web service;
4. An unmonitored web service invokes a monitored web service; and
5. An application client without a PI Client Agent invokes a monitored web service.

5.1 SYSTEM WORKING SCENARIO 1

As shown in Figure 15, this working scenario demonstrates how the PI Monitor Agent and the PI Client Agent work in the situation where an application client with the PI Client Agent invokes a monitored web service.

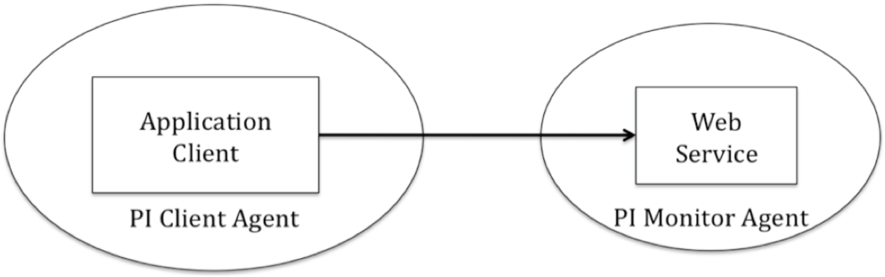


Figure 15 Application client with PI Client Agent invokes monitored service.

When an application client initiates a SOAP request, the PI Client Agent on the application client platform will kick in and detect the invoked web service in the SOAP message body. If this invoked web service is a monitored web service, the PI Client Agent will prompt a user authentication and find the process that made the call. Once the user authentication is finished and the process is found, the PI Client Agent will insert the user ID, the process name and the invoked web service name in the header of the SOAP request and pass the message along. If the invoked web service is not a monitored web service, the PI Client Agent will pass the message along without user authentication and without finding the process that invoked the web service.

After receiving this SOAP request, the PI Monitor Agent on the invoked web service will find the following information in the SOAP header: user ID, process name and the invoked web service. For the request, based on the process name and the invoked web service, the PI Monitor Agent will query Web Service KB about the User Category/ies, Purpose/s, Action/s and whether the request or response is static. If the request or response is static, the Web Service KB will return Data Category/ies for the static request or static response. The PI Monitor Agent will cache the found information for the response (static or dynamic; including the Data Category if it is static). At this point, if the request is static, the PI Monitor Agent has all the information it needs to evaluate an EPAL policy and the EPAL Policy Evaluation Process will start. If the request is dynamic, the PI Monitor Agent will need to parse though the SOAP request message to find the data items contained in the message. After finding the data items in the SOAP request message, the PI Monitor Agent will send all the data items to PI Elements Schema KB to get Data Category/ies for the request. Then, the EPAL Policy Evaluation Process will start.

The EPAL Policy Evaluation Process is as follow for this particular scenario: the PI Monitor Agent sends the User Category/ies, Data Category/ies, Purpose/s, and Action/s to the Privacy Policy KB. Based on the enterprise's Privacy Policy, the KB will return the applicable <rule> element together with any <condition> element that is referenced in the applicable <rule> element. If the <rule> element is to drop, the PI Monitor Agent will drop the request without processing the returned information further. If the <rule> element is to allow the message, based on the returned <condition> element, the PI Monitor Agent will query the Privacy Database Query Component to get the data value required to evaluate the <condition> element. If the <obligation> sub-element in <rule> element is not null, the PI Monitor Agent will send the <obligation> element to the Obligation Enforcement Component to enforce the obligation stated in the <obligation> element. Once the Obligation Enforcement Component finishes the enforcement, it will return a success or failure message to the PI Monitor Agent. If the obligation fails, the PI Monitor Agent will drop the request since the obligations cannot be honored. If the obligation is successful, the PI Monitor Agent will allow the request to go through. If there is neither a <condition> sub-element nor an <obligation> sub-element in the <rule> element, the PI Monitor Agent will allow the request through.

When the response is returned from the invoked web service, the PI Monitor Agent will find the cached information regarding whether the response is static and, if it is static, the Data Category. For the response, the PI Monitor Agent will query User KB about User Category/ies and Purpose/s based on user ID and process name. The PI Monitor Agent will also query the Application KB about Action/s based on the invoked web service and process name. If a response is static, the PI Monitor Agent already has the required information to evaluate the EPAL policy. The PI Monitor Agent will start the EPAL Policy Evaluation Process. If the response is dynamic, the PI Monitor Agent will need to parse through the SOAP response message and find the data items in the message. Then, the PI Monitor Agent will send the found data items to the PI Elements Schema KB and get the Data Category/ies for the response. Finally, the PI Monitor Agent will start the EPAL Policy Evaluation Process.

5.2 System Working Scenario 2

As shown in Figure 16, this system working scenario demonstrates how the PI Monitor Agent works in the situation where a monitored web service invokes another monitored web service.

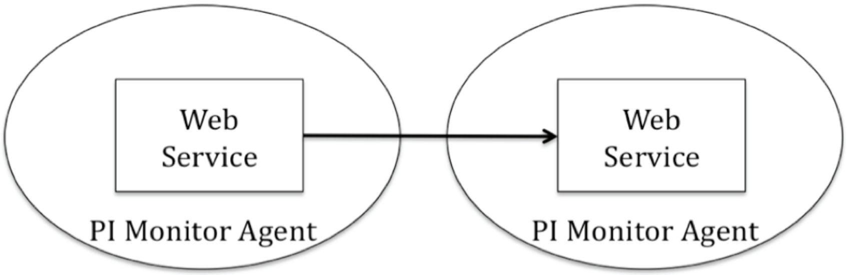


Figure 16 A monitored service invokes another monitored service.

When a monitored web service initiates a SOAP request, the PI Monitor Agent hooked with this invoking web service will find the invoked web service in the SOAP request body. If this invoked web service is monitored, the PI Monitor Agent will find the invoking web service's name. Once the invoking web service and invoked web service are found, the PI Monitor Agent will query Web Service KB based on the invoking web service and the invoked web service to find the User Category/ies, Purpose/s, Action/s, and whether the request or response is static. If any of the request and response is static, Web Service KB will also return the Data Category/ies for the static request or static response. The PI Monitor Agent will insert a flag indicating the response is static or dynamic to the SOAP request header. If the request is static, the PI Monitor Agent already has the information needed to evaluate the EPAL policy for the request and it will start the EPAL Policy Evaluation Process. If the request is dynamic, the PI Monitor Agent will need to parse through the request message, find data items contained in the request message and send them to the PI Elements Schema KB to collect the Data Category/ies for the request. Then, the PI Monitor Agent will start the EPAL Policy Evaluation Process. The PI Monitor Agent will also insert the following information to the header of the request: the invoked web service, the invoking web service, a flag indicating that the request has been checked.

After receiving this SOAP request, the PI Monitor Agent on the invoked web server will find a flag in the header indicating that this request message has been checked. The PI Monitor Agent will cache the information that is found in the header (the invoked web service, the invoking web service, a flag indicating the response is static or dynamic) and pass the request along to the invoked web service. When the SOAP response is returned from the invoked web service, the PI Monitor Agent on the invoked web service will check the cached flag indicating the response is static or dynamic. If it is a static response, the PI Monitor Agent will query Web Service KB based on the invoking web service and the invoked web service to find User Category/ies, Data Category/ies, Purpose/s and Action/s. At this point of time, the PI Monitor Agent already have all the information needed for the EPAL Policy Evaluation Process. The PI Monitor Agent will start the EPAL Policy Evaluation Process. If the response is dynamic, the PI Monitor Agent will parse through the response message, find the data items contained in the message and send them to the PI Elements Schema KB to get Data Category/ies for the response. The PI Monitor Agent will also query Web Service KB based on the invoking web service and the invoked web service to find User Category/ies, Purpose/s and Action/s. The PI Monitor Agent will then start the EPAL Policy Evaluation Process. If the EPAL Policy Evaluation Process results in an “allow” action, the SOAP response message will arrive at the invoking web service. The PI Monitor Agent on the invoking web service will allow this response through to the business logic of its monitored web service.

5.3 SYSTEM WORKING SCENARIO 3

As shown in Figure 17, this system working scenario demonstrates how the PI Monitor Agent works in the situation where a monitored web service invokes an unmonitored web service.

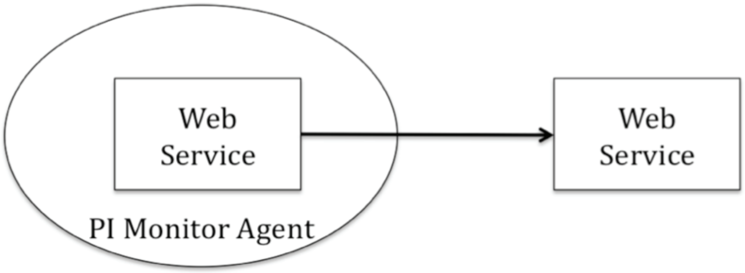


Figure 17 A monitored service invokes an unmonitored service.

When a monitored web service initiates a SOAP request, the PI Monitor Agent on this invoking web service will find the invoked web service in the SOAP request message. The PI Monitor Agent will verify if the invoked web service is a monitored web service or not. If not, the PI Monitor Agent will find the invoking web service. For the request, the PI Monitor Agent will query Web Service KB based on the invoking web service and the invoked web service to find User Category/ies, Purpose/s, Action/s and whether the request and response are static. If any of the request or response is static, the Web Service KB will return the Data Category/ies for the request or response. The PI Monitor Agent will cache the found information: whether the response is static or dynamic. If the request is static, the PI Monitor Agent already has the information needed to evaluate the EPAL policy and the EPAL Policy Evaluation Process will start. If the request is dynamic, the PI Monitor Agent will parse through the request message, find all the data items in the request message and send them to the PI Elements Schema KB to get Data Category/ies for the request. The EPAL Policy Evaluation Process for request will then start.

When the response comes back from the unmonitored web service, the PI Monitor Agent will check the following cached information: whether the response is static or dynamic. If the response is static, the PI Monitor Agent will query Web Service KB based on the invoking web service and the invoked web service to find User Category/ies, Data Category/ies, Purpose/s, Action/s and then start the EPAL Policy Evaluation Process. If response is dynamic, the PI Monitor Agent will need to parse through the response message and get Data Category/ies for the response from the PI Elements Schema KB. The PI Monitor Agent then queries the Web Service KB based on the invoking web service and the invoked web service to find User Category/ies, Purpose/s and Action/s. Then the EPAL Policy Evaluation Process will start for the response.

5.4 System Working Scenario 4

As shown in Figure 18, this system working scenario demonstrates how the PI Monitor Agent works in the situation where a monitored web service gets invoked by an unmonitored web service or an application client without the PI Client Agent.

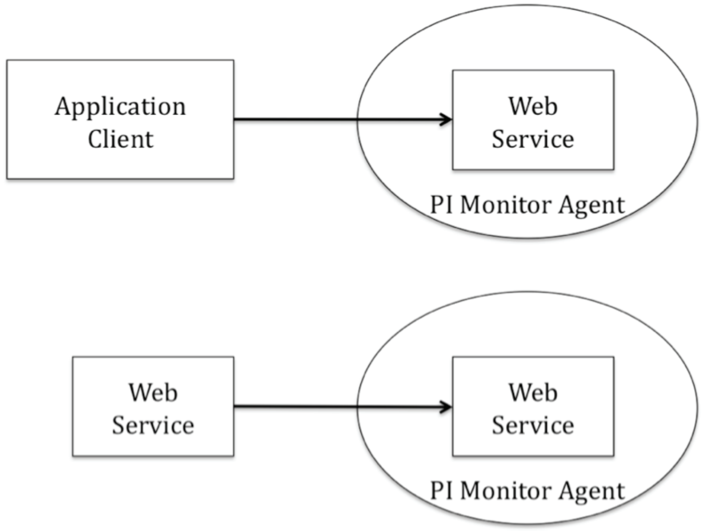


Figure 18 An unmonitored service or an application client without PI Client Agent invokes a monitored service.

When receiving a request from an unmonitored web service or an application client without a PI Client Agent, the PI Monitor Agent on the invoked monitored web service will notice that the PI Monitoring Header is missing. The PI Monitor Agent will parse the SOAP message body for the invoked web service. Based on the invoked web service, the PI Monitor Agent will find Purpose/s, Action/s and User Category/ies in the Web Service KB for both the request and response (default values will be used if these variables cannot be found in KBs). Also, the web service will return whether or not the request or response is static. If the request or response is static, the Web Service KB will return the Data Category/ies for either the static request or the static response. If the request is static, the PI Monitor Agent will already have all the information needed to evaluate the EPAL policy. The EPAL Policy Evaluation Process will start. If the request is dynamic, the PI Monitor Agent will parse through the request message and collect the Data Category/ies in the PI Elements Schema KB. The PI Monitor Agent will start the EPAL Policy Evaluation Process.

When the response comes back from the unmonitored invoked web service, if the response is static, the PI Monitor Agent already has the required information to evaluate the EPAL policy. The PI Monitor Agent will start the EPAL Policy Evaluation Process. If the response is dynamic, the PI Monitor Agent will parse through the response message and get Data Category/ies for the response in the PI Elements Schema KB. Then, the EPAL Policy Evaluation Process will start.

CHAPTER 6 DEMONSTRATION CASES

This chapter presents four sample cases to demonstrate how the architecture works to provide private data monitoring and privacy policy enforcement in a WSA environment.

6.1 SELECTION OF CASES

These demonstration cases are chosen to reflect the working scenarios for the system as described in Chapter 5. These scenarios are designed to reflect typical applications of WSA in an e-Commerce business. Also, the following ideas are demonstrated in these scenarios:

- How the system deals with cases when multiple privacy rules apply;
- How the <condition> element in EPAL policy is evaluated.

In the four cases, demonstration Case 2 is selected to reflect the two scenarios described in Section 5.4. To the PI Monitor Agent on the monitored web service, the SOAP request from an unmonitored web service or the SOAP request from an application client without the PI Client Agent are the same; therefore, the way the PI Monitor Agent handles the incoming requests and departing responses in these two scenarios is the same.

6.2 DEMONSTRATION CASES

In this section, we are going to present four demonstration cases.

6.2.1 Demonstration Case 1

In this scenario, an application client with a PI Client Agent invokes a monitored web service, where PI elements are found in both the request and the response. The request is static and the response is dynamic.

To accomplish services provided to customers, an e-Commerce web site “example.com” collects certain privacy information from its customers. With users’ consent, users’ PI collected by “example.com” can be disclosed to an entity other than the one who collected the PI. Another entity here could be another internal division or a third party organization. This scenario also demonstrates the following concepts:

- How the <condition> element in EPAL is evaluated;
- Which policy will be returned when multiple EPAL rules apply.

For marketing purposes, Mike Smith (UserID: HO5MTS), a Marketing Manager from the Marketing Department, wants to retrieve private information that “example.com” collected from customer John Crane (CustomerID: T56333492). To retrieve the customer’s PI information, the Marketing Manager will need to use an application client called ICAM (Integrated Customer Account Management). This client is developed for users from different departments of “example.com” to retrieve different kinds of information regarding a specific user (such as contact information, billing information and marketing information). In this case, Mike, the Marketing Manager, is going to use this client to retrieve marketing information about a customer.

According to “example.com’s” privacy policy, users from the Marketing Department are only allowed to retrieve the following private information regarding a customer for marketing purposes if the customer agrees to let his/her information be used for marketing purposes:

- First Name
- Middle Name
- Family Name
- Address
- Email
- Phone Number
- Fax
- Consent
- Preferences

Web Service account_info_retrieval

During user data retrieval, the web service ***account_info_retrieval*** expects the following data items from an application client:

- Customer_ID
- Data_Group (The kind of information requested; for example: marketing, billing or contact)

If marketing data is requested, web service returns the following PI elements among other non-PI data to the application client.

<FirstName>
<MiddleName>
<FamilyName>
<Address>
<Email>
<PhoneNumber>
<Fax>
<Consent>
<Preferences>

PI Elements Schema KB

This KB is where we maintain a PI element dictionary and define Data Categories. In PI Elements Schema KB, <FirstName>, <MiddleName>, <FamilyName>, <Address>, <Email>, <PhoneNumber>, <Fax>, <Consent> and <Preferences> are defined as PI elements and are categorized as ***marketing_data***.

User KB

This is one of the places where we define User Categories. In this scenario for the response, the Marketing Manager who invoked the web service ***account_info_retrieval*** is the user of the PI elements. In User KB, the Marketing Manager, Mike Smith, belongs to User Categories ***marketing_processor*** and ***fraud_processor***.

In User KB it also shows that if user Mike Smith (UserID: HO5MTS) is using application ***ICAM.exe***, the purposes of the usage of PI in the response of web service ***account_info_retrival*** are ***marketing_processing*** and ***fraud_processing***.

Web Service KB

Web Service KB contains the following information regarding this invocation:
Based on the invocation parties (process name: ***ICAM.exe*** and the invoked web service: ***account_info_retrival***) the purpose of the request of web service ***account_info_retrieval*** is mapped to ***user_information_retrieval_input*** and the resultant action is mapped to ***process***. For the request, web service ***account_info_retrieval*** is the user of the PI elements in the request and web service ***account_info_retrieval*** is mapped to User Category ***user_information_retrieval_interface.***

Also, in Web Service KB, based on the invocation parties (invoker: ***ICAM.exe*** and invoked: ***account_info_retrival***) it shows that the request of this invocation is static with the Data Category ***Customer_Identifier*** and that the response is dynamic. The Data Category ***Customer_Identifier*** only has one PI element, which is ***Customer_ID.***

Application KB

Application KB contains the following information: based on the invocation parties, the Action of the response of web service ***account_info_retrieval*** is mapped to ***retrieval.***

PI Client Agent

PI Client Agents are installed on the workstations at the Marketing Department. When Mike Smith initiates a service request to the web service ***account_info_retrieval*** from the ICAM client on his workstation, the PI Client Agent on his workstation will notice that a SOAP message has been initiated. By looking at the SOAP message, the PI Client Agent finds that this request is going to a monitored service. The PI Client Agent will then prompt an authentication window and try to find the name of the process that made this request. When the authentication is finished and process name is found, the PI Client Agent inserts the following headers into the SOAP message and passes the message along.

```
<user-id>HO5MTS<user-id/>
<process-name>ICAM.exe<process-name/>
<invoked-web-service>account_info_retrieval<invoked-web-service>
```

How the PI Monitor Agent Works

After receiving this SOAP request, the PI Monitor Agent on the invoked web service ***account_info_retrieval*** detects the following headers in the SOAP request message: user ID, process name and the invoked web service. For the request, the PI Monitor Agent queries Web Service KB based on the process name ***ICAM.exe*** and the invoked web service ***account_info_retrival*** to find out that the Purpose of this request is ***user_information_retrieval_input,*** the Action of this request is ***process,*** the User Category of this request is ***user_information_retrieval_interface*** and that the request is static with Data Category ***Customer_Identifier*** and that the response is dynamic. At this point of time, the PI Monitor Agent has all the information to evaluate the EPAL Policy for the request. The PI Monitor Agent will start the EPAL Policy Evaluation Process and get the following applicable <rule> element from Privacy Policy KB.

```
<rule id=”1” ruling=”allow”>
                <short-description>rule for case 1</short-description>
                <long-description>rule for case 1</long-description>
                <user-category refid=”user_information_retrieval_interface”/>
                <data-category refid=”Customer_Identifier”/>
                <purpose refid=”user_information_retrieval_input”/>
                <action refid=”process”/>
</rule>
```

** The above rule element in the EPAL Policy states that any user information retrieval interface is allowed to receive the customer ID to process retrieval request for user information.**

Since there is no <condition> and <obligation> sub-element in the applicable <rule> element and the <rule> element is to allow the request, the PI Monitor Agent will allow the request through and log the message.

When the response is returned from the invoked web service, the PI Monitor Agent on the invoked web service queries User KB based on the user ID ***HO5MTS*** and process name ***ICAM.exe*** to get the User Category/ies and Purpose/s for the response. In this case the PI Monitor Agent gets multiple User Categories for ***HO5MTS***: ***marketing_processor*** and ***fraud_processor***. The PI Monitor Agent also gets multiple purposes: ***marketing_processing*** and ***fraud_processing***. For the response, the PI Monitor Agent queries Application KB to find out the action of ***ICAM.exe*** invoking ***account-info-retrieval*** (also based on invocation parties) and gets ***retrieval*** as action. The PI Monitor Agent already knows that the response is dynamic. The PI Monitor Agent will parse through the response message and find the following data items: <FirstName>, <MiddleName>, <FamilyName>, <Address>, <Email>, <PhoneNumber>, <Fax>, <Consent> and <Preferences>. The PI Monitor Agent will send these items to the PI Elements Schema KB. The PI Elements Schema KB finds that these data items are all PI elements and they form the Data Category ***marketing_data***. At this point of time, the PI Monitor Agent will evaluate EPAL Policy for the response. During the EPAL Policy Evaluation Process, Privacy Policy KB will find the following applicable <rule> elements in EPAL policy:

```
<rule id=”2” ruling=”allow”>
        <short-description>rule for case 1</short-description>
        <long-description>rule for case 1</long-description>
        <user-category refid=”marketing_processor”/>
        <data-category refid=”marketing_data”/>
        <purpose refid=”marketing_processing”/>
        <action refid=”retrieval”/>
        <condition refid=”condition1”/>
</rule>

<rule id=”3” ruling=”allow”>
        <short-description>rule for case 1</short-description>
        <long-description>rule for case 1</long-description>
        <user-category refid=”fraud_processor”/>
        <data-category refid=”marketing_data”/>
        <purpose refid=”fraud_processing”/>
        <action refid=”retrieval”/>
</rule>
```

Privacy Policy KB will return the first applicable <rule> element together with the <condition> element referenced in the <rule> element. The referenced <condition> element is as follows:

```
<condition id="condition1">
                <predicate
                  refid="http://www.research.ibm.com/privacy/epal#string-equal">
                        <function
                          refid="http://www.research.ibm.com/privacy/epal#string-bag-to-value">
                                <attribute-reference
                                  container-refid="Customer"
                                  attribute-refid="Consent"/>
                        </function>
                        <attribute-value
                          simpleType="http://www.w3.org/2001/XMLSchema#string">true</attribute-
value>
                </predicate>
        </condition>
```

After receiving the returned <rule> element and <condition> element, the PI Monitor Agent will send the referenced data attribute <attribute-reference container-refid=”Customer” attribute-refid=”Consent”/> and the CustomerID **T56333492** to the Privacy Database Query Component. The Privacy Database Query Component, based on the information it has regarding which database holds the required data attribute, accesses the database to get the value for data attribute Consent under CustomerID **T56333492.** In this case, since the customer has agreed to share privacy information for marketing purpose, data attribute Consent has value “1” meaning it has been agreed upon. Since there is no obligation required in the policy, the PI Monitor Agent will allow the response through and log the response message.

6.2.2 Demonstration Case 2

In such a scenario, an application client without a PI Client Agent invokes a monitored web service, where PI elements are found only in the request and the request is static.

A customer is trying to apply for an account on an e-Commerce site “example.com” from Internet Explorer on his workstation. According to “example.com’s” privacy policy, when a customer applies for an account, it only collects the following privacy information from the customer:

- First Name
- Middle Name
- Family Name
- Gender
- Date of Birth
- Address
- Email
- Phone Number
- Fax
- Consent: an option that users choose to receive promotions from “example.com”

- Preferences: commodity categories for which users choose to receive promotions in the future (The available values are: Apparel, Automotive, Baby, Electronics, Grocery & Pets, Health & Wellness, Home & Office, Jewelry, Movies, Music & Books, Outdoor Living, Sports, Toys, Video Games. Customers can select any combination of the above options as values in Preferences.)

Web Service account_apply

When a customer applies for an account on “example.com”, the monitored web service ***account_apply*** on “example.com” is called. The client (in this case it is Internet Explorer browser) on customer side will send the required data items to web service ***account_apply,*** which then writes the collected data into a customer database.

To invoke ***account_apply***, the application needs to send the following data to the web interface of web service ***account_apply***:

<FirstName>
<MiddleName>
<FamilyName>
<Gender>
<DateofBirth>
<Address>
<Email>
<PhoneNumber>
<Fax>
<Consent>
<Preferences>

Web Service KB

In this case, for the request, web service ***account_apply*** is the user of the PI elements. In Web Service KB, it contains information that the User Category for the request of

account_apply is ***application_processor.*** In Web Service KB it shows that ***anonymous*** is the User Category for the response.

Web Service KB contains information to map web services to actions and purposes. In this scenario, the request of web service ***account_apply*** is mapped to Action ***store*** and Purpose ***user_application***. The response of the web service ***account_apply*** is mapped to Action ***none*** and Purpose ***none***.

Also, in Web Service KB, it shows that the request and response of the web service account_apply are static: the Data Category of request is ***initial_user_data*** which contains “First Name”, “Middle Name”, “Family Name”, “Gender”, “Date of Birth”, “Address”, “Email”, “Phone Number”, “Fax”, “Consent” and “Preferences”; the Data Category of the response is ***none***, which means there is no PI element in response.

How the PI Monitor Agent Works

The PI Monitor on the web service ***account_apply*** notices that the SOAP request that just came has no PI monitoring header. Therefore, it is from an unmonitored web service or application client without a PI Client Agent. The PI Monitor Agent detects in the request message that the invoked web service is ***account_apply***. For both the request and the response, the PI Monitor Agent will query the Web Service KB based on the invoked web service ***account_apply*** for User Category/ies, Purpose/s, Action/s and whether the request and response are static. If any request or response is static, the Data Category/s for the static request or static response will be returned. After checking the knowledge it has, the Web Service KB returns the following to the PI Monitor:

For request:

- User Category ***application_processor***
- Purpose ***user_application***
- Action ***store***
- request is static
- Data Category ***initial_user_data***

For response:

- User Category ***anonymous***
- Purpose ***none***
- Action ***none***
- response is static
- Data Category ***none***

At this point in time, the PI Monitor Agent already has all the information required to evaluate the EPAL policy for the request. The EPAL Policy Evaluation Process will begin. It finds the following <rule> element for the request:

```
<rule id=”4” ruling=”allow”>
        <short-description>rule for case 2</short-description>
        <long-description>rule for case 2</long-description>
        <user-category refid=”application_processor”/>
        <data-category refid=”initial_user_data”/>
        <purpose refid=”account_apply”/>
        <action refid=”store”/>
</rule>
```

According to the privacy policy, the SOAP request message of this invocation is allowed through and the PI Monitor Agent will log the request.

When the response comes back from the invoked web service, because the Data Category for the response is ***none***, response in this invocation is also allowed through. The PI Monitor Agent will log the response.

6.2.3 Demonstration Case 3

In this scenario, a monitored web service invokes another monitored web service, where PI elements are found only in the request and the request is static.

When web service ***account_apply*** is processed, child web service ***welcome_email_notify*** will be invoked. ***welcome_email_notify*** is hosted on another web server. The invocation of ***welcome_email_notify*** is to send the new customer a welcome email, which includes all the information the user just provided so the user can keep a record of what has been collected by “example.com”. According to the privacy policy at “example.com”, web service ***account_apply*** is allowed to forward all the information it collected to web service ***welcom_email_notify***.

Web services (methods) involved:

- account_apply
- welcome_email_notify

Web Service welcome_email_notify

The interface of ***welcome_email_notify*** expects the following data from an invoker:

<FirstName>

<MiddleName>

<FamilyName>

<Gender>

<DateofBirth>

<Address>

<Email>

<PhoneNumber>

<Fax>

<Consent>

<Preferences>

Web Service KB

In this case, web service ***welcome_email_notify*** is the user of request and web service ***account_apply*** is the user of response. In Web Service KB, the User Category for web service ***welcome_email_notify*** is ***email_processor***. The User Category for web service ***account_apply*** is ***application_processor***.

In this case, the Web Service KB shows that if the invoker is ***account_apply*** and the invoked is ***welcome_email_notify***, the Action of the request is ***disclose*** and the Purpose of the request is ***user_notification***.

In Web Service KB, it also shows that the request and response of web service ***account_apply*** invoking web service ***welcome_email_notify*** are static: the data category of request is ***initial user data***; the data category of response is ***none***, which means that there is no PI element in the response.

How the PI Monitor Agent Works

When the SOAP request message is initiated by web service ***account_apply***, the PI Monitor Agent on the web service ***account_apply*** will find out in the SOAP message that the invoked web service is ***welcome_email_notify***, which is in the PI Monitor Agent's Monitored Web Service List. The PI Monitor Agent then figures out that ***account_apply*** is the invoking web service. Based on ***account_apply*** as the invoker and ***welcome_email_notify*** as the invoked, the PI Monitor will find in Web Service KB the following information about the request:

- User Category ***email_processor***
- Purpose ***user_notification***
- Action ***disclose***
- request is static
- Data Category ***initial user data***

It also finds the following information about the response:

- response is static

The PI Monitor will insert the following information to the SOAP request header: a flag indicating the response is static.

Since the request is static and the Data Category for the request has been determined, the PI Monitor Agent now has all the information needed to evaluate the EPAL for the request. EPAL Policy Evaluation Process will begin. The PI Monitor Agent will get the following <rule> element from Privacy Policy KB:

```
<rule id=”5” ruling=”allow”>
        <short-description>rule for case 3</short-description>
        <long-description>rule for case 3</long-description>
        <user-category refid=”email_processor”/>
        <data-category refid=”initial_user_data”/>
        <purpose refid=”user_notification”/>
        <action refid=”disclose”/>
</rule>
```

According to the privacy policy, the SOAP request message of this invocation is allowed through. Before the PI Monitor passes the request message along, the request message will be logged in the Audit Log module and the following headers will be inserted into the header of the SOAP request message:

```
<invoking-web-service>account_apply< invoking-web-service/>
<invoked-web-service>welcome_email_notify< invoked-web-service/>
<flag-request-checked>yes< flag-checked/>
<flag-static-response>yes< flag-static-response/>
```

After receiving this request message, the PI Monitor Agent on the invoked web service ***welcome_email_notify*** will find a flag in the header indicating that this request has been checked. The PI Monitor Agent will cache other information found in the header, such as, the invoked web service, the invoking web service and a flag indicating the response is static. The PI Monitor Agent will query the Web Service KB based on the invoking web service ***account_apply*** and the invoked web service ***welcome_email_notify*** for the User Category/ies, Purpose/s, Action/s and Data Category/ies for the response. The PI Monitor Agent will get data category "none" for the response, which means there are no PI elements in the response. The PI Monitor will log the response message in the Audit Log module and allow the response message through.

6.2.4 Demonstration Case 4

In Case 4, a monitored web service invokes an unmonitored web service, where PI elements are found only in the request. The request is static.

To complete an online order made by a customer, "example.com" will need to send the customer's billing information to a bank to process the transaction. When a customer makes an order online, he/she also agrees to let "example.com" disclose his/her PI information to a third party financial organization for transaction purposes. According to "example.com's" privacy policy, only the information that is needed for a transaction will be disclosed to the financial organization.

According to the privacy policy at "example.com", the following billing information will be allowed to be disclosed to the financial organization to finish a transaction:

- First Name
- Middle Name
- Family Name
- Billing Address
- Credit Card Number
- Credit Card Expiration Date

Web Service transaction_invoke

During the transaction, the web service ***transaction_invoke*** on “example.com” invokes web services provided by other financial organizations to finish transactions. For our demonstration, we can assume that TD Canada’s web service for online transactions is ***customer_transaction*** and it requires the following input:

<Requester’s Organization ID>
<First Name>
<Middle Name>
<Family Name>
<Billing Address>
<Credit Card Type>
<Credit Card Number>
<Credit Card Expiration Date>
<Amount of money>
<Purpose for transaction>

Among the above data, we can find a billing information Data Category. Since the web service ***customer_transaction*** is located at TD Canada, the PI Monitor Agent does not monitor it.

PI Elements Schema KB

In PI Elements Schema KB, the following data are defined as PI elements: <First Name>, <Middle Name>, <Family Name>, <Billing Address>, <Credit Card Number>, <Credit Card Expiration Date>. They are categorized as the ***billing_data*** Data Category.

User KB

Since the User of the PI information is a web service, there is nothing in the User KB that will be provided to the PI Monitor.

Web Service KB

In this case, the web service ***customer_transaction*** is the recipient of the request and ***transaction_invoke*** is the recipient of the response. In Web Service KB, both the web service ***transaction_invoke*** and the web service ***customer_transaction*** are mapped to User Category ***transaction_proccessor***. In this case, Web Service KB shows that if the invoker is ***transaction_invoke*** and the invoked is ***customer_transaction***, both the request and the response are static with Data Category ***billing_data*** for the request and no private data (data category ***none***) for the response. The Action of the request is ***disclose*** and the Purpose of the request is ***transaction_process***. The Purpose for the response is ***none*** and the Action for the response is ***none***.

How the PI Monitor Agent Works

When the SOAP request message is initiated by the web service ***transaction_invoke*** , the PI Monitor Agent on the web service ***transaction_invoke*** will find out in the SOAP message that the invoked web service is ***customer_transaction***. The PI Monitor Agent finds out that the ***customer_transaction*** is not a monitored web service. The PI Monitor then figures out that ***transaction_invoke*** is the invoking web service. Based on ***transaction_invoke*** as the invoker and ***customer_transaction*** as the invoked, the PI Monitor Agent will find in Web Service KB the following information for the request:

User Category ***transaction_proccessor***
Purpose ***transaction_process***
Action ***disclose***
request is static
Data Category ***billing_data***

For the response, the PI Monitor Agent will find in the Web Service KB that the response is static and the Data Category for the response is ***none***. This information will be cached for later use.

The PI Monitor Agent is now ready to evaluate the EPAL Policy for the request. The EPAL Policy Evaluation Process will begin. The PI Monitor Agent will get the following <rule> element from Privacy Policy KB:

```
<rule id=”5” ruling=”allow”>
        <short-description>rule for case 4</short-description>
        <long-description>rule for case 4</long-description>
        <user-category refid=”transaction_processor”/>
        <data-category refid=”billing_data”/>
        <purpose refid=”transaction_process”/>
        <action refid=”disclose”/>
</rule>
```

According to the privacy policy, the SOAP request message of this invocation is allowed through. The PI Monitor Agent will log the request message and pass the request message along. When the response message comes back, the PI Monitor Agent will notice that the Data Category for the response is ***none*** and, therefore, allow it through and log the response message.

6.3 Conclusion

These four examples demonstrate the proof that the concept of the PI Monitor Agent for privacy monitoring and enforcement in a WSA environment. The agent is effective to facilitate support for privacy in organizations’ computer systems. The efficiency of the PI Monitor Agent will be evaluated in next Chapter.

CHAPTER 7 PROTOTYPE AND EXPERIMENTATION

As a proof of concept, a prototype of the proposed architecture has been implemented in ASP.NET. Based on the prototype, experiments were conducted to evaluate the performance of the PI Monitor Agent. In this Chapter, we will first describe the implementation of the PI Monitor Agent and the PI Client Agent in ASP.NET and then we will describe the objective of our experiment, the set up of the experiment and the platform of the experiment. We will also describe the method we used to measure the performance of the PI Monitor Agent. Finally, the results of the experiment will be analyzed to explore the performance of the PI Monitor Agent and, based on the analysis, we will make suggestions on the implementation of the PI Monitor Agent for a real operational environment.

7.1 IMPLEMENTATION OF PROTOTYPE

Technology that can be used to implement the PI Monitor Agent and the PI Client Agent varies in ASP.NET when compared to a Java-based platform. Within the ASP.NET infrastructure, PI Monitor Agents and PI Client Agents can be implemented by using ASP.NET SOAP extensions. In Java infrastructure, AXIS is the technology to use. The prototype of the PI Monitor Agent and the PI Client Agent was implemented in ASP.NET using C# language. This section describes how to implement the PI Monitor Agent and the PI Client Agent in ASP.NET. We tested the prototype on selected web services with an application client to simulate the WSA environment in which the privacy architecture might be working.

ASP.NET provides a unified pipeline for web services to communicate globally. This pipeline is using SOAP as a common information communication format and HTTP as the connection protocol (Shepherd, 2003). When a SOAP message crosses this pipeline, from a client to an ASP.NET web service and also when it returns from the web service to the client, it needs to pass through several stages. Figure 19 illustrates these stages of SOAP message processing for both the client and the server.

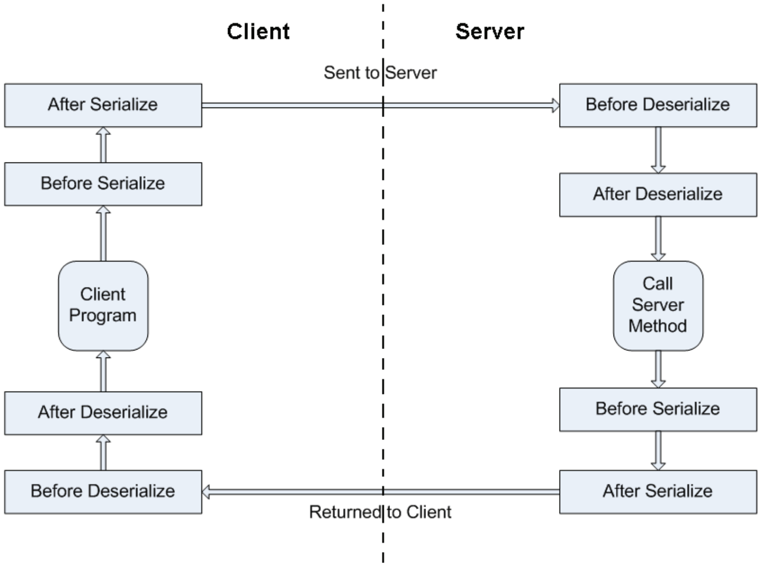


Figure 19 Life cycle of a SOAP message (Shepherd, 2003).

Although ASP.NET automates much of this process, we can also use SOAP extensions to hook up our own code into the pipelines and manipulate SOAP messages along the pipelines. Figure 20 illustrates how SOAP extensions fit into the overall ASP.NET architecture.

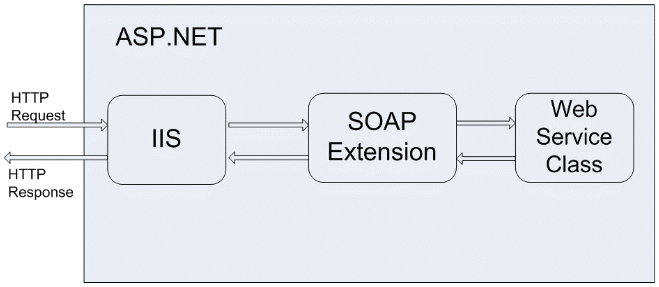


Figure 20 SOAP Extensions in ASP.NET infrastructure (Shepherd, 2003).

ASP.NET SOAP extensions are able to inspect or modify a SOAP message at specific stages in the message processing on either the client or the server. Through this SOAP extension architecture, we are able to access a SOAP message as it is de-serialized into objects and as it is serialized from a common language runtime (CLR) object into a

SOAP message, which enables us to implement the PI Monitor Agent and the PI Client Agent (Shepherd, 2003).

7.1.1 How to Find the Called Web Service Method

The PI Monitor Agent and the PI Client Agent need to find the called web service method in the SOAP request message. In SOAP 1.1, when a client sends a SOAP request message to a web service, the name of the called web service method is placed in two locations: the SOAPAction HTTP header and the Request Element's Name in the SOAP envelope. This second location is noted because, in the SOAP 1.2 version, the called web service method can only be found in the Request Element's Name location. In order to support both SOAP 1.1 and 1.2 in this implementation, we chose to use the Request Element's Name location to identify the called web service.

When a SOAP message gets initiated from a monitored web service or an application client with the PI Client Agent, the PI Monitor Agent or the PI Client Agent on the initiator needs to find out what web service has been invoked. We suppose the following example: a web site provides their customers with service to query prices of stocks. The web service name is "GetStockPrice". A sample SOAP request message to invoke that web service is shown below. In the sample SOAP message the invoked web service name is highlighted in the color gray. For this SOAP request message, the PI Monitor Agent or the PI Client Agent need to find the gray-highlighted web service and also the name space, which is located in a line previous to the one containing the web service name. The two, the web service name and the name space, form a unique identifier for the invoked web service:

http://www.example.org/stock/GetStockPrice

A sample SOAP request:

```
POST /InStock HTTP/1.1
Host: www.example.org
```

```
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

7.1.2 PI Monitor Agent

The PI Monitor Agent is implemented using the SOAP extension class. The core method of the SOAP extension class is ProcessMessage. The ProcessMessage method provides interfaces to hook up our own code to the eight stages in a SOAP message’s life cycle. The PI Monitor Agent intercepts the process of SOAP messages at two stages on the server side: “before deserialize” and “after serialize”. The following shows a sample code to hook the PI Monitor Agent into the two stages of processing a SOAP message on the server side. The method PIMonitorAgentIncoming() carried out during the stage “before deserialize” is to process the incoming message while the method PIMonitorAgentOutgoing() carried out during the “after serialize” stage is to process the outgoing message.

```
public override void ProcessMessage(SoapMessage message)
  {
    switch (message.Stage)
    {
      // message coming
      case SoapMessageStage.BeforeDeserialize:
        PIMonitorAgentIncoming(message);
        break;

      // about to call methods
      case SoapMessageStage.AfterDeserialize:
        break;

      // after method call
      case SoapMessageStage.BeforeSerialize:
        break;

      // message outgoing
      case SoapMessageStage.AfterSerialize:
        PIMonitorAgentOutgoing(message);
        break;
    }
  }
```

If the PI Monitor Agent works on the invoked web service, the PIMonitorAgentIncoming() method deals with the request message from the invoker and the PIMonitorAgentOutgoing() method deals with the response message being sent out to the invoker. If the PI Monitor Agent works on the invoking web service, the PIMonitorAgentIncoming() method deals with the returned response from the invoked web service and the PIMonitorAgentOutgoing method deals with the request message being sent to the invoked web service.

To evaluate the performance of the PI Monitor Agent, timers are put into the PI Monitor Agent’s code to log the time at different points of processing the SOAP request/response message.

7.1.3 PI Client Agent

The PI Client Agent works on the client side to process the SOAP request message sent out from the client. The PI Client Agent intercepts the SOAP request message at the stage “after serialize”. A piece of sample code to hook the PI Client Agent into the stage “after serialize” on the client side is shown as follows.

```
public override void  ProcessMessage(SoapMessage message)
   {
      switch (message.Stage)
      {
        // message coming from client
        case SoapMessageStage.BeforeDeserialize:
          break;

        // about to call methods
        case SoapMessageStage.AfterDeserialize:
          break;

        // after method call
        case SoapMessageStage.BeforeSerialize:
          break;

        // outgoing to client
        case SoapMessageStage.AfterSerialize:
          PIClientAgent(message);
          break;
      }
```

```
  }
```

The PI Client Agent() method carried out at the stage “after serialize” is the major body of the PI Client Agent. This method only deals with the request message sent out from the client. When the response message comes back from the invoked web service, the PI Client Agent does not need to get involved.

7.1.4 The Supporting Components

The focus of this thesis is the PI Monitor Agent; therefore, how other supporting components work is beyond [there is another instance where this should be changed in another chapter] the scope of this paper. For the purpose of the proof of concept and our experiment, we also implemented the supporting components in our experiment environment. The supporting components are implemented with web services that take a SOAP request from the PI Monitor Agent, deserialize the SOAP request, return data object results, serialize the returned data objects into a SOAP response message and send the response back. Since how each supporting component works is beyond the scope of this thesis, the returned data objects are programmed into the code of each component in advance. The code of each supporting component still performs the deserialization and serialization of SOAP requests/responses.

7.2 EXPERIMENTATION

In order to test the efficacy of the developed prototype and to explore the potential performance cost of the PI Monitor Agent, we set up a series of experiments to reflect the demonstration cases presented in Chapter 6. Results are measured to evaluate the efficiency of the PI Monitor Agent and conclusions are made on the feasibility of a PI Monitor Agent in an operational environment.

In this section, we will describe how to measure the performance of the PI Monitor Agent. Then we will provide an overview of the set-up of the experiments, describe the platforms and instrumentation of the experiments and, finally, we will present and analyze the results of the experiments.

7.2.1 Evaluation Method

To evaluate the performance cost of the PI Monitor Agent, we will use the time delays that are introduced to the legacy system during the processing of SOAP messages by the PI Monitor Agent. Time delays caused by the PI Monitor Agent are represented by the following function.

$$T = t_{parse} + \sum_{i=1}^{n} t_i + t_{iner}$$

Where

T denotes the total delay caused by the PI Monitor Agent;

t_{parse} denotes the time spent by the PI Monitor Agent parsing the SOAP message;

t_i denotes the time spent to invoke the supporting component i (of n components); and

t_{iner} denotes the time spent within the logic function of the PI Monitor Agent itself.

There are n=8 components that could be invoked by the PI Monitor Agent during the processing of a SOAP message. They include:

- Web Service KB
- Application KB
- User KB
- PI Elements Schema KB
- Privacy Policy KB
- Privacy Database Query Component
- Obligation Enforcement Component
- Audit Log

To calculate the time delays that are related to the PI Monitor Agent, timer programs have been put into the code of the PI Monitor Agent to log the following time points during the processing of a SOAP message by the PI Monitor Agent:

1. When the PI Monitor Agent starts processing a SOAP message T_1;
2. When the PI Monitor Agent starts parsing through the SOAP message T_2;
3. When the PI Monitor Agent finishes parsing through the SOAP message T_3;
4. When the PI Monitor Agent starts invoking the supporting components T_4;
5. When the PI Monitor Agent finishes invoking the supporting components T_5;
6. When the PI Monitor Agent finishes processing the SOAP message T_6.

By checking the log file that is generated by the PI Monitor Agent after it finishes processing a SOAP message, we are able to get T_1- T_6 and calculate T, t_{parse}, $\sum_{i=1}^{8} t_i$ and t_{iner} in the following way:

- $T = T_6 - T_1$;
- $t_{parse} = T_3 - T_2$;
- $\sum_{i=1}^{8} t_i = T_5 - T_4$;
- $t_{iner} = T - t_{parse} - \sum_{i=1}^{8} t_i$.

Time Spent to Parse the SOAP Message t_{parse}

In this thesis, we define the time spent by the PI Monitor Agent to parse the SOAP message to be the time span between the PI Monitor Agent's receipt of the SOAP message and PI Monitor Agent's completion of the deserialization of the SOAP into objects. t_{parse} depends on two major factors: the length of the SOAP message and whether the SOAP request/response is static. If the SOAP request/response is static and the Web Service KB contains information regarding whether the SOAP request/response is static, there are two outcomes:

1. the PI Monitor Agent does not need to parse the SOAP body to find the data objects contained in the SOAP message, and
2. the PI Monitor Agent only needs to find the invoked web service if the invoked web service is not contained in the header of the SOAP message.

Time Spent to Invoke A Supporting Component t_i

We define the time spent by the PI Monitor Agent to invoke a supporting component to be the time span between the PI Monitor Agent's start of serialization of the objects into a SOAP request message to invoke a supporting component and the receipt of the returned data from the invoked supporting component. The time spent on serialization of input data objects of the invocation into a SOAP request message and the time spent on deserialization of the SOAP response message into returned data objects are included in t_i. This delay depends primarily on two major factors: how the supporting component is implemented and the size of the information base contained in the supporting component. For example, a KB implemented using a database to store information would be faster than a KB implemented using XML to store information.

Time Spent within the Logic Function of the PI Monitor Agent t_{iner}

We define the time spent within the logic function of the PI Monitor Agent t_{iner} to be the time the agent spends on processing the returned data objects from other supporting components. t_{iner} depends on the number of logic functions that the PI Monitor Agent needs to go through to make a decision.

7.2.2 Experiment Setup Overview

To test the PI Monitor Agent, we set up our experiment environment to simulate a real WSA environment. Demonstration Cases 1 and 3 presented in Chapter 6 were chosen as the set up for our experiments. The reason for choosing these two cases is because they represent two typical scenarios a PI Monitor Agent will confront from the perspective of workload: one request is from an application client with the PI Client Agent and the other request is from another monitored web service.

Case 1 and Case 2 in Chapter 6 are scenarios where the application client invokes a monitored web service. They differ in that in Case 1 the user ID and process name will be inserted into the SOAP header by the PI Client Agent, while in Case 2 there is no such information in the header due to the absence of the PI Client Agent. For the PI Monitor Agent on the invoked web service, the workload would be almost the same for these two cases, except in case 2 the PI Monitor Agent on the invoked web service will need to parse the SOAP request message to find the invoked web service while in case 1 the invoked web service is already in the SOAP header. In Case 3, the monitored web service is invoked by another monitored web service, which is a different workload expectation from that of Cases 1 and 2. In Case 3, two PI Monitor Agents work together to check the request and response messages between two web services. As a result, the workload on each PI Monitor Agent is reduced compared to the workload in Cases 1 and 2. Case 4 is similar to Case 1 and Case 2 from the perspective of the workload assigned to the PI Monitor Agent during the invocation of a web service.

Therefore, we chose Case 1 in Chapter 6 to be the setup of Experiment 1 while Case 3 in Chapter 6 was chosen to be the setup of Experiment 2.

Experiment 1

For Experiment 1, from Case 1 in Chapter 6, the experiment environment was set up as follows.

On the client side we created an application client called ICAM (Integrated Customer Account Management), which is developed to retrieve customer information from a web service called *account_info_retrieval*.

The user interface of ICAM is designed to work in the following way: in the windows form, the user needs to type in the customer ID and choose the retrieved data type from a drop-down list. There are three options in the drop-down list: marketing, billing and contact. When both the customer ID is typed in and the retrieved data type is chosen, the user will click on the “OK” button to retrieve the data by invoking the web service and passing it the customer ID and the data type as parameters. This results in a SOAP request message being sent to the web service *account_info_retrieval*. The web service will retrieve the customer information and return it in a SOAP response message, which will result in the customer data being displayed in a separate window.

The web service *account_info_retrieval* is designed to work in the following way: after receiving the request from the ICAM client, the web service accesses the customer database to retrieve the requested customer information and return it back to the ICAM client in a SOAP response message. An MS SQL database is used to implement the customer database, which is populated with random customer data. Details about the design of the customer database can be found in Appendix I.

The PI Client Agent on the client intercepts the SOAP request sent out from ICAM, processes it and passes it along.

On the server side, the PI Monitor Agent intercepts the SOAP request/response to/from the web service *account_info_retrieval* and queries other supporting components to fulfill its tasks.

Experiment 2

For Experiment 2, from Case 3 in Chapter 6, the experiment environment was set up as follows.

For this experiment, we created two web services: *account_apply* and *welcome_email_notify*. When the web service *account_apply* is processed, the child web service *welcome_email_notify* will be invoked. The invocation of *welcome_email_notify* is to send the new customer a welcome e-mail, which includes all the information the user just provided so the user can keep a record of what has been collected by example.com. According to the privacy policy at example.com, the web service *account_apply* is allowed to forward all the information it collected to the web service *welcome_email_notify*.

The PI Monitor Agents are installed on the two web services.

7.2.3 Experiments Platform

In the experiment, to eliminate other factors that may affect the performance such as network bandwidth, all components were placed on one test machine, including: ICAM, web service *account_info_retrieval*, web service *account_apply*, web service *welcome_email_notify,* customer database, PI Client Agent, PI Monitor Agent and other supporting components.

The test machine is a Windows Server 2003 SP2 virtual machine mounted on VMware Fusion. The Windows Server 2003 SP2 virtual machine is configured with the following resources:

- 1 processor core at 2.66 GHz
- 1468 MB memory
- 6MB L2 Cache
- 1.07 GHz Bus Speed

7.2.4 Experiment Results and Analysis

In this section, we present the results of our experiments, analyze these results and, based on the analysis, we make conclusions regarding the performance of the PI Monitor Agent.

Experiment 1

The results of Experiment 1 are presented in Table 1.

Table 1 Results of experiment 1.

	T	t_{parse}	$\sum_{i=1}^{8} t_i$
SOAP Request	3.5ms	0.3ms	3.0ms
SOAP Response	7.9ms	0.8ms	6.7ms

In Case 1, since the SOAP request is from a client with a PI Client Agent, the header of the SOAP request message contains the following information: user ID, process name and the invoked web service. Based on the information contained in the SOAP message header, the PI Monitor Agent is able to find the User Category, Purpose, and Action in the Web Service KB. Since the request is static and the Data Category contained in the request is also stored in the Web Service KB, the PI Monitor Agent will be able to get the Data Category for the request without parsing through the body of the SOAP request message to get the data objects and then get the Data Category by querying the PI Elements Schema KB based on the acquired data objects. The PI Monitor Agent then queries the Privacy Policy KB to get the applicable rule element. As a result, the parse time t_{parse} for the request is a small fraction of the total time delay T because the PI Monitor Agent only needs to parse the header of the SOAP request message. The time spent within the logic function of the PI Monitor Agent $t_{iner} = T - t_{parse} - \sum_{i=1}^{8} t_i$ is, therefore, $t_{iner} = 0.2ms$. t_{iner} is only a small fraction of the total time delay T. The majority of the time delay T is the time spent on invoking other supporting components $\sum_{i=1}^{8} t_i$.

For the response message, the PI Monitor Agent queries the User KB and the Application KB to get the User Categories, Purpose and Actions. Since the PI Monitor Agent already knows that the response is dynamic, it needs to parse through the body of the SOAP response message to collect the data objects and then query the PI Elements Schema KB to determine the Data Category. The PI Monitor Agent also queries the Privacy Policy KB to determine the applicable rule element. After receiving the returned rule element from the Privacy Policy KB, the PI Monitor Agent also needs to invoke the Privacy

Database Query Component to acquire the data attribute referenced in the condition sub-element in the returned rule element. The PI Monitor Agent will then match the data attribute with the condition sub-element in the returned rule element. As a result, for the response message, the time spent to invoke other supporting components ($\sum_{i=1}^{8} t_i$=6.7ms) increases while the time to parse the SOAP response message (t_{parse}=0.8ms) also increases due to the fact that the PI Monitor Agent needs to parse the whole message to get the data objects contained in the message. t_{iner} also increases from 0.2ms to 0.4ms.

From the results of Experiment 1, the following conclusions can be made:

1. Including the information about whether the request/response of a web service is static or dynamic in the Web Service KB is helpful to improve the performance of the PI Monitor Agent by reducing the time spent on parsing t_{parse}.

2. Based on the results of the experiment, we suggest that multiple threading programming or concurrent programming should be used to implement the PI Monitor Agent in order to cut down on the time spent on invoking the supporting components. In our implementation, the time spent on invoking other supporting components $\sum_{i=1}^{8} t_i$ is the sum of the time delay that is related to invoking each of the supporting components. This is because, in our implementation, the invocations of supporting components by the PI Monitor Agent are programmed for serial executions. However, invocations that are independent from each other can be divided into several threads or parallel programs. For invocations whose inputs depend on the outputs of other invocations, these invocations will be carried out in a serial manner. For example, the invocation of the Privacy Policy KB depends on the invocations of other KBs such as the Web Service KB, the Application KB and the User KB. Therefore, the invocation of the Privacy Policy KB must be carried out after the completion of invocations of other KBs.

3. Using web services to implement the supporting components gives us flexibility. However, using web services is not the most efficient way of implementing supporting components as the associated parsing, serializing, and deserializing incurs substantial overhead delays. The supporting components can be implemented through dynamic link libraries in the architecture system. This will tremendously improve the performance due to the fact that DLL call is a much more efficient method than calling a web service through a SOAP message.

4. Since t_{parse} is a major factor to the total time delay T, it would benefit the performance if the parsing can be shared with the business logic of the legacy system. When a SOAP message comes to the monitor web service, the data objects abstracted by the PI Monitor Agent from the message can be used by the legacy system's business logic so the legacy system does not need to parse the SOAP message again.

5. t_{iner} is a small fraction of the total time delay T for both request and response in Experiment 1. Therefore, in scenarios that are represented by Experiment 1, t_{iner} can be ignored in the analysis of the PI Monitor Agent's performance.

Experiment 2

The results of Experiment 2 are presented in Tables 2 and 3.

Table 2 Results of experiment 2 regarding the PI Monitor Agent on web service *account_apply*. SOAP request is the request sent out from web service *account_apply*. SOAP response is the response that comes back from the invoked web service *welcome_email_notify*.

	T	t_{parse}	$\sum_{i=1}^{8} t_i$
SOAP Request	4.1ms	0.3ms	3.3ms
SOAP Response	0ms	0ms	0ms

Table 3 Results of experiment 2 regarding the PI Monitor Agent on web service *welcome_email_notify*. SOAP request is the request from web service *account_apply*. SOAP response is the response sent out from the invoked web service *welcome_email_notify*.

	T	t_{parse}	$\sum_{i=1}^{8} t_i$
SOAP Request	0.3ms	0.3ms	0ms
SOAP Response	2.4ms	0.3ms	2.1ms

In Experiment 2, the PI Monitor Agent on the invoking web service has already checked the SOAP request message. Consequently, when the request message arrives at the invoked web service, the PI Monitor Agent on the invoked web service does not need to go through the whole checking process again because a flag in the request message's header will reveal that this request message has been checked by another PI Monitor Agent. Similarly, the PI Monitor Agent on the invoked web service has already checked the SOAP response, so when the response comes back to the invoking web service the PI Monitor Agent on the invoking web service does not need to go through the whole checking process again because a flag in the response header will reveal that this response message has been checked by another PI Monitor Agent. Furthermore, when processing the request message the PI Monitor Agent on the invoking web service will also receive information regarding the response, if there is any, in the KBs. As this information is passed along to the PI Monitor Agent on the invoked web service, it could save time spent on processing the response message by the PI Monitor Agent on the invoked web service.

In Experiment 2, on both the invoker side and the invoked side, the PI Monitor Agents do not need to parse the SOAP message body to get the Data Category since both the request and the response are static and their Data Category information is contained in the Web Service KB. The time spent within the logic function of the PI Monitor Agent is $t_{iner} = T - t_{parse} - \sum_{i=1}^{8} t_i$, therefore for the PI Monitor Agent on web service *account_apply* processing the request t_{iner} is 0.5 ms while for the PI Monitor Agent on web service *welcome_email_notify* processing the response t_{iner} is 0 ms. Compared to t_{iner} in Experiment 1, t_{iner} for the PI Monitor Agent on web service *account_apply* processing the request increases. This is because the PI Monitor Agent on *account_apply* needs to match the invoked web service name with the Monitored Web Service List and insert information into the header of the SOAP request message it processes.

A noticible result is that t_{iner} for the PI Monitor Agent on web service *wecome_email_notify* processing the response is 0 ms. This is because after receiving the response the PI Monitor Agent soon finds out that the response does not have any private data, as a result the PI Monitor Agent allows the response message through without going further through its logic function.

Another noticible result is that T for the PI Monitor Agent on web service *account_apply* processing the response from web service *welcome_email_notify* is 0 ms. This is because once the PI Monitor Agent on the invoking web service sees a response comes back from another monitored web service, it assumes that the response message has been checked and will allow it through right away.

Based on the results of Experiment 2, the following conclusions can be made:

1. Using headers of SOAP messages to carry information that is required by the child web service does create a performance cost. However, this cost could be compensated for in regard to the time that is saved by the PI Monitor Agent on the invoked web service due to the fact that the PI Monitor Agent on the invoked side does not need to go through the process the PI Monitor Agent went through to get the information contained in the header. To prove this point, more experimentation needs to be done.

2. The introduction of the Monitored Web Service List to the system does create performance cost.

3. The distribution of the workload to multiple PI Monitor Agents in a WSA environment could benefit the overall performance by reducing the possibility of overloading a single PI Monitor Agent.

7.3 SUMMARY

In this chapter we have proved the concept of the PI Monitor Agent we designed and evaluated the potential performance cost of the PI Monitor Agent based on two experiments. The experiments show that T can be improved by improving t_{parse}, $\sum_{i=1}^{n} t_i$ and t_{iner}. For the implementation of the PI Monitor Agent in an operational environment, we present the following conclusions:

1. The total time cost T can be reduced by including information about whether the request/response of a web service is static or dynamic in the Web Service KB.

2. The use of multiple threading programming or concurrent programming to implement the PI Monitor Agent will improve its performance. As most of the modern programming languages support concurrent programming and the wide application of multi-core processors, using concurrent programming is more achievable in recent years.

3. The total time cost T can be reduced by implementing the supporting components in the architecture by using DLL instead of using web services.

4. Sharing parsing with the business logic of the invoked web service would benefit the total performance cost. However, this requires a change to the code of the legacy system. In some cases, it is not cost efficient to change the code.

5. Distributing the workload to multiple PI Monitor Agents in a WSA environment could benefit the overall performance by reducing the possibility of overloading a single PI Monitor Agent.

6. The introduction of the Monitored Web Service List to the system does create performance cost.

7. Using headers of SOAP messages to carry information that is required by the child web service does create a performance cost. This cost could be balanced by the time that is saved on the PI Monitor Agent on the invoked web service due to the fact that the PI Monitor Agent on the invoked side does need to go through the process that the PI Monitor Agent went through to get the information contained in the header. To prove this point, more experimentation needs to be done.

CHAPTER 8 CONCLUSION

In this thesis we have designed an agent for privacy monitoring and enforcement in a WSA environment and created a prototype as a proof of concept. The agent is based on a specific multi-agent architecture for privacy compliance. To design the agent, the information that is required for monitoring privacy and enforcing privacy policy in a WSA environment has been studied. This intensive examination led to an extension of the architecture to bring out its full potential in monitoring privacy information flows and enforcing privacy policies in a WSA environment. Based on the prototype, experiments were completed to evaluate the potential performance cost of the PI Monitor Agent. The research outcomes lead us to suggestions on the implementation of a PI Monitor Agent for a real operational environment.

The added components in the architecture are:

- The PI Client Agent
- The User KB
- The Obligation Enforcement Component
- The Privacy Database Query Component

A format of the SOAP header has also been proposed to facilitate privacy monitoring and enforcement in WSA.

Besides the above enhancements, in the extended architecture we also proposed the concept of static and dynamic request/response of a web service. Our experiments show that the total time delay T caused by the PI Monitor Agent can be reduced by including information about whether the request/response of a web service is static or dynamic in the Web Service KB. On the other hand, the experiment results also show that the introduction of the Monitored Web Service List to the system does create performance cost.

By analyzing the results of the experiments, we also suggested that using multiple threading programming or concurrent programming to implement the PI Monitor Agent will improve its performance. In our implementation, the time spent on invoking other supporting components $\sum_{i=1}^{8} t_i$ is the sum of the time delay that is spent to invoke each of the supporting components. The reason for this is that, in our implementation, the invocations of supporting components from the PI Monitor Agent are programmed for serial executions. However, invocations that are independent from each other can be divided into several threads or parallel programs. For invocations whose inputs depend on the outputs of other invocations, these invocations will be carried out in a serial manner. For example, the input of the invocation of the Privacy Policy KB depends on the outputs of invocations of other KBs such as the Web Service KB, the Application KB and the User KB. Therefore, the invocation of the Privacy Policy KB must be carried out after the completion of invocations of other KBs.

Another technology that can be used to improve the performance of the PI Monitor Agent is to implement the supporting components through Dynamic Link Library (DLL) instead of web services. Using web services to implement the supporting components does have some advantages over the method of using DLL. One of the major advantages is that implementing the supporting components as web services gives the architecture flexibility for integration with other systems due to the interoperability nature of WSA. There is, however, a performance cost to this advantage.

Since the time spent by the PI Monitor Agent on parsing (t_{parse}) SOAP messages is a major factor in the total time delay T, it would benefit the performance if the parsing can be shared with the business logic of the legacy system. When a SOAP message comes to the monitored web service, the data objects abstracted by the PI Monitor Agent from the message can be used by the legacy system so the legacy system does not need to parse the SOAP message again.

The analysis of the results of our experiments also shows that the distribution of the workload of monitoring and enforcing privacy to multiple PI Monitor Agents in a WSA environment could benefit the overall performance by reducing the possibility of overloading a single PI Monitor Agent.

For future work, further research is recommended in the following areas:

1. Using the SOAP header to carry information that is used for the support of privacy monitoring and enforcement does create an extra performance cost. However, this cost could be compensated by the time that would be saved by the PI Monitor Agent on the invoked web service due to the fact that the PI Monitor Agent on the invoked side does not need to go through the process the PI Monitor Agent went through to get the information contained in the header. To prove this point, more experiments need to be done.
2. The distribution of workload over PI Monitor Agents on networked web services could affect the performance of the privacy architecture as a whole. More research work needs to be done to measure the exact performance gain due to the distribution of the workload.

Overall, the research and experiments undertaken and presented in this thesis prove that the PI Monitor Agent methodology adds both efficiency and effectiveness to the important requirements for monitoring privacy information flows and enforcing privacy policy in WSA.

BIBLIOGRAPHY

[1] Adam, N.R., Worthmann, J.C. (1989). Security-control methods for statistical databases: a comparative study. *CSUR 21*(4), 515–556.

[2] Adams, C. and Barbieri, K. (2006). Privacy Enforcement in E-Services Environments. *Privacy Protection for E-Services*, Idea Group, Inc.

[3] Agrawal, R., Kiernan, J., Srikant, R., Xu, Y. (2002). Proceedings of the 28th *International Conference on Very Large Databases (VLDB).* Hippocratic databases.

[4] Ashley, P., Powers, C.S., Schunter, M. (2002). *Third International Symposium on Electronic Commerce*. Privacy promises, access control, and privacy management.

[5] Barth, A., Mitchell, J. C. and Rosenstein, J. (2004). Proceedings of the 2004 *Workshop on Privacy in the Electronic Society*. Conflict and combination in privacy policy languages. ACM Press.

[6] Bodorik, P., Jutla, D.N., Dhillon, I. (2009). Privacy Compliance with Web Services. *Journal of Information Assurance and Security*, *4* (5), 412-421.

[7] Byun, J.W., Bertino, E., Li, N. (2004). Purpose Based Access Control for Privacy Protection in Relational Database Systems. Purdue University.

[8] Byun, J.W., Bertino, E., Li, N. (2005). *Symposium on Access Control Model And Technologies (SACMAT).* Purpose based access control of complex data for privacy protection.

[9] Byun, J.W., Bertino, E. (2006). Micro-views, or on how to protect privacy while enhancing data usability: concepts and challenges. *SIGMOD Rec. 35*(1), 9–13.

[10] Cranor, L., Egelman, S., Hogben, G., Humphrey, J., Langheinrich, M., Marchiori, M., Presler-Marshall, M., Reagle, J., Schunter, M., Stampley, D. (2006). The Platform for Privay Preferences 1.1 (P3P1.1) Specification. Retrieved from: http://www.w3.org/TR/P3P11/

[11] Davis, J.C (2000). Protecting privacy in the cyber era. IEEE Technology and Society Magazine, 10-22.

[12] Dragovic, B., Crowcroft, J. (2004). Proceedings of *New Security Paradigms Workshop*. Information Exposure Control through Data Manipulation for Ubiquitous Computing.

[13] Gandon, F. and Sadeh, N. (2004). Semantic Web Technologies to Reconcile Privacy and Context Awareness. *Web Semantics Journal, 1*(3), 241-260.

[14] Garcia, D.Z.G., Toledo, M.A. (2008). *11th IEEE International Conference on Computational Science and Engineering Workshops*. Web Service Privacy Framework Based on a Policy Approach Enhanced with Ontologies.

[15] Hung, P., Chiu, D., Fung, W., Cheung, W., Wong, R., Choi, S., Kafeza, E., Kwok, J., Pun, J., Cheng V. (2005). *ICEC '05 Proceedings of the 7th international conference on Electronic commerce.* Towards end-to-end privacy control in the outsourcing of marketing activities: a web service integration solution.

[16] IBM (2003). Enterprise Privacy Authorization Language [EPAL 1.2]. Retrieved from http://www.zurich.ibm.com/security/enterprise-privacy/epal/Specification/index.html

[17] International Security Trust and Privacy Alliance [ISTPA] (2009). Privacy Management Reference Model. Retrieved from http://xml.coverpages.org/ISTPA-PrivacyManagementReferenceModelV20.pdf

[18] Iyengar, V.S. (2002). Proceedings of *SIGKDD'02*. Transforming Data to Satisfy Privacy Constraints. Edmonton, Alberta.

[19] Jajodia, S., Sandhu, R. (1991). *ACM International Conference on Management of Data (SIGMOD)*. Toward a multilevel secure relational data model. 50–59. New York: ACM Press.

[20] Jutla D.N., Bodorik P, Zhang Y. (2006). PeCAN: An Architecture for Privacy-aware Electronic Commerce User Contexts. *Information Systems Journal*, *31*(4-5), 295-320.

[21] Karjoth, G., Schunter, M. and Waidner, M. (2002). Privacy-enabled services for enterprises. IBM Research.

[22] Kobsa, A. and Schreck, J. (2003). Privacy through Pseudonymity in User-Adaptive Systems. *ACM Transactions in Internet Technology*, *3*(2), 149-183.

[23] LeFevre, K., Agrawal, R., Ercegovac, V., Ramakrishnan, R., Xu, Y., DeWitt, D. (2004). *The 30th International Conference on Very Large Databases (VLDB).* Disclosure in Hippocratic Databases.

[24] Leino-Kilpi, H., Valimaki, M., Dassen, T., Gasull, M., Lemonidou, C., Scott, A., Arndt, M. 2001. Privacy: A Review of the Literature. *International Journal of Nursing Studies, 38,* 663-671.

[25] Li, M., Sun, X., Wang, H., Zhang, Y., Zhang, J. (2011). Privacy-aware access control with trust management in web service. *World Wide Web Journal*.

[26] Parker, R.G. (2005). Privacy Issues: Business Impacts and Responsibilities.

CAAA/SAP AG Technology and Accounting Education Seminar Series. Canadian Academic Accounting Association (CAAA).

[27] Rao J., Dimitrov D., Hofmann P. and Sadeh N. (2006). In Proceedings of *the IEEE International Conference on Web Services (ICWS 2006)*. A Mixed Initiative Framework for Semantic Web Service Discovery and composition.

[28] Rezgui, A., Ouzzani, M., Bouguettaya, A., Medjahed, B. (2002). Proceedings of the *4th international workshop on Web information and data management.* Preserving privacy in web services.

[29] Sandhu, R., Chen, F. (1998). The multilevel relational data model. *ACM Trans. Inf. Syst. Secur. 1*(1), 93–132.

[30] Schoeman, E. D. (1984). *Philosophical Dimensions of Privacy: An Anthology*. New York, NY: Cambridge University Press.

[31] Shepherd, G. (2003). Using SOAP Extensions in ASP.NET. Retrieved from: http://msdn.microsoft.com/en-us/magazine/cc164007.aspx

[32] Song, R., Korba,L., and Yee, G. (2006). Pseudonym Technology for E-Services. *In Privacy Protection for E-Services*. Idea Group, Inc.

[33] Xu, W., Sekar, R., Ramakrishnan, I., Venkatakrishnan, V. (2005). Proceedings of the *WWW '05 Special interest tracks and posters of the 14th international conference on World Wide Web.* An approach for realizing privacy-preserving web-based services.

[34] Yee, G. (2007). Proceedings of the 2007 *ACM workshop on Secure web services*. A privacy controller approach for privacy protection in web services.

APPENDIX A SOAP TEMPLATES FOR WEB SERVICE KB

SOAP REQUEST

```
<?xml version="1.0"?>
<soap:Envelope
 xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
        <m:GetWSKBProperty xmlns:m="http://www.example.com/WSKB/">
                <m:invoking_web_service_name_or_process_name/>
                <m:invoked_web_service_name/>
                <m:request_or_response/>
        </m:GetWSKBProperty>
</soap:Body>

</soap:Envelope>
```

SOAP RESPONSE

```
<?xml version="1.0"?>
<soap:Envelope
 xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
        <m:GetWSKBPropertyresponse xmlns:m="http://www.example.com/WSKB/">
                <m:user_category/>
                <m:purpose/>
                <m:action/>
                <m:whether_request_is_static/>
                <m:data_category_for_request/>
                <m:whether_response_is_static/>
                <m:data_category_for_response/>
        </m:GetWSKBPropertyresponse>
</soap:Body>

</soap:Envelope>
```

APPENDIX B SOAP TEMPLATES FOR USER KB

SOAP REQUEST

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
        <m:GetUKBProperty xmlns:m="http://www.example.com/UKB/">
                <m:user_ID/>
                <m:process_name/>
        </m:GetUKBProperty>
</soap:Body>

</soap:Envelope>
```

SOAP RESPONSE

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
        <m:GetUKBPropertyresponse xmlns:m="http://www.example.com/UKB/">
                <m:user_category/>
                <m:purpose/>
        </m:GetUKBPropertyresponse>
</soap:Body>

</soap:Envelope>
```

APPENDIX C SOAP TEMPLATES FOR APPLICATION KB

SOAP Request

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
        <m:GetAKBProperty xmlns:m="http://www.example.com/AKB/">
                <m:process_name/>
                <m:invoked_web_service/>
        </m:GetAKBProperty>
</soap:Body>

</soap:Envelope>
```

SOAP Response

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
        <m:GetAKBPropertyresponse xmlns:m="http://www.example.com/AKB/">
                <m:action/>
        </m:GetAKBPropertyresponse>
</soap:Body>

</soap:Envelope>
```

APPENDIX D SOAP TEMPLATES FOR PI ELEMENTS SCHEMA KB

SOAP REQUEST

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
        <m:GetPIESKBProperty xmlns:m="http://www.example.com/PIESKB/">
                <m:data_item/>
                …
                <m:data_item/>
        </m:GetPIESKBProperty>
</soap:Body>

</soap:Envelope>
```

SOAP RESPONSE

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
        <m:GetPIESKBPropertyresponse xmlns:m="http://www.example.com/PIESKB/">
                <m:data_category/>
        </m:GetPIESKBPropertyresponse>
</soap:Body>

</soap:Envelope>
```

APPENDIX E   SOAP TEMPLATES FOR AUDIT LOG KB

SOAP REQUEST

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
        <m:GetALKBProperty xmlns:m="http://www.example.com/ALKB/">
                <m:SOAP_message/>
                <m:receive_time/>
                <m:source_of_message/>
                <m:destination_of_message/>
                <m:user_category/>
                <m:data_category/>
                <m:purpose/>
                <m:action/>
                <m:decision/>
                <m:rule/>
        </m:GetALKBProperty>
</soap:Body>

</soap:Envelope>
```

SOAP RESPONSE

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
        <m:GetALKBPropertyresponse xmlns:m="http://www.example.com/ALKB/">
                <m:success_or_failure/>
        </m:GetALKBPropertyresponse>
</soap:Body>

</soap:Envelope>
```

APPENDIX F   SOAP TEMPLATES FOR PRIVACY POLICY KB

SOAP Request

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
        <m:GetPPKBPolicy xmlns:m="http://www.example.com/PPKB/">
                <m:user_category/>
                <m:data_category/>
                <m:purpose/>
                <m:action/>
        </m:GetPPKBPolicy>
</soap:Body>

</soap:Envelope>
```

SOAP Response

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
        <m:GetPPKBPolicyresponse xmlns:m="http://www.example.com/PPKB">
                <m:rule/>
                <m:condition/>
        </m:GetPPKBPolicyresponse>
</soap:Body>

</soap:Envelope>
```

APPENDIX G SOAP TEMPLATES FOR OBLIGATION ENFORCEMENT COMPONENT

SOAP REQUEST

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
        <m:RequestOEC xmlns:m="http://www.example.com/OEC/">
                <m:obligation/>
        </m:RequestOEC >
</soap:Body>

</soap:Envelope>
```

SOAP RESPONSE

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
        <m:RequestOECresponse xmlns:m="http://www.example.com/OEC/">
                <m:success_or_failure/>
        </m:RequestOECresponse>
</soap:Body>

</soap:Envelope>
```

APPENDIX H SOAP TEMPLATES FOR PRIVACY DATABASE QUERY COMPONENT

SOAP REQUEST

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
          <m:RequestPDQC xmlns:m="http://www.example.com/PDQC/">
                    <m:container refid= attribute refid=/>
                    <m:unique_data_subject_identifier/>
          </m:RequestPDQC>
</soap:Body>

</soap:Envelope>
```

SOAP RESPONSE

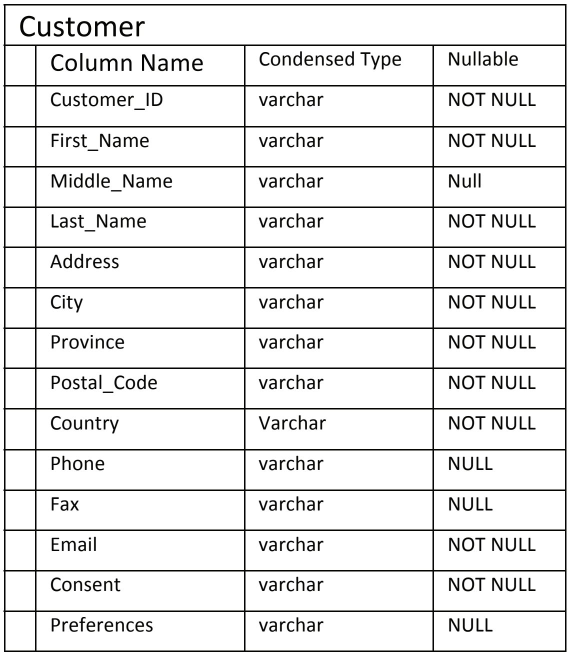
```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
          <m:RequestPDQCresponse xmlns:m="http://www.example.com/PDQC/">
                    <m:container refid= attribute refid=/>
          </m:RequestPDQCresponse>
</soap:Body>

</soap:Envelope>
```

APPENDIX I DESIGN FOR CUSTOMER DATABASE

There is only one table in customer database – table Customer.



Customer

	Column Name	Condensed Type	Nullable
	Customer_ID	varchar	NOT NULL
	First_Name	varchar	NOT NULL
	Middle_Name	varchar	Null
	Last_Name	varchar	NOT NULL
	Address	varchar	NOT NULL
	City	varchar	NOT NULL
	Province	varchar	NOT NULL
	Postal_Code	varchar	NOT NULL
	Country	Varchar	NOT NULL
	Phone	varchar	NULL
	Fax	varchar	NULL
	Email	varchar	NOT NULL
	Consent	varchar	NOT NULL
	Preferences	varchar	NULL