WS-Trustworthy: A Framework for Web Services Centered Trustworthy Computing

Jia Zhang
Department of Computer Science
Northern Illinois University
Chicago, IL 60115
jiazhang@cs.niu.edu

Liang-Jie Zhang
IBM T.J. Watson Research

Yorktown Heights, NY 10598
zhanglj@us.ibm.com

Jen-Yao Chung
IBM T.J. Watson Research

Yorktown Heights, NY 10598
jychung@us.ibm.com

Abstract

The emerging paradigm of Web services has been gaining significant momentum in the recent years since it offers a promising way to facilitate Business-to-Business (B2B) collaboration. However, it is not clear that this new model of Web services provides any measurable increase in computing trustworthiness. In this paper we propose a generic framework to control the trustworthiness of computing in the domain of Web services. A layered model is established to highlight four key elements: resources, policies, validation processes, and management. The robustness of this model exhibits its flexibility and extensibility. Examples utilizing our framework are reported.

1. Introduction

On January 15, 2002, Bill Gates delivered a company wide email that coined a concept known as "Trustworthy Computing" [5]:

"…Trustworthy Computing is computing that is as available, reliable and secure as electricity, water services and telephony…."

As Bill Gates billing it as the highest priority to the entire Microsoft workforce [5], this concept of trustworthy computing has been significantly changing the way that Microsoft designs and builds software [12]. Moreover, this concept has been leading the whole IT industry to a complete new level of trustworthiness in computing.

As Microsoft's follow-up white paper indicates, the concept of trustworthy computing has been bringing a "sea change" not only in the way how software is developed and delivered, but also in the way how the whole society views computing in general [7]. Since we are still at the infant stage of this new revolution, enormous amount of research issues, either immediate or fundamental, are open for resolutions. Our research is initiated and excited by this challenge.

This research focuses upon trustworthy computing in the domain of Web services. The emerging paradigm of Web services has been obtaining significant momentum in both academia and industry in recent years. Simply put, a Web service is a programmable Web application that is universally accessible through standard Internet protocols [4], such as Simple Object Access Protocol (SOAP) [10]. By means of each organization exposing its software services on the Internet and making them accessible via standard programmatic interfaces, this model of Web services offers a promising way to facilitate Business-to-Business (B2B) collaboration. In addition, Web services technology largely increases cross-language and cross-platform interoperability of distributed computing [4]. Furthermore, this paradigm of Web services opens a new cost-effective way of engineering software to quickly develop and deploy Web applications by dynamically integrating other independently published Web services components to conduct new business transactions.

However, it is not clear that this new model of Web services provides any measurable increase in computing trustworthiness. Among other aspects, the essential feature of "dynamic discovery and integration" of Web services model raises new challenges to software trustworthiness. In a traditional software system, all of its components and their relationships are pre-decided before the software runs. Therefore, each component can be thoroughly tested, and the interactions among components can be fully tested, before the system starts to run. Web services extend this paradigm by providing a more flexible approach to dynamically locate and assemble distributed Web services in an Internet-scale setting. In detail, when a system requires a service component, the system will search a public registry [11] where Web services providers publish their services, choose the optimal Web service that fulfils its requirements, bind to the service's Web site, and invocate the service. In other words, in this dynamic invocation model, it is likely that users may not even know which Web services they will use [6], much less those Web services' trustworthiness. Even worse, since a Web service is potentially a dynamic entity controlled and hosted by its provider, there are no guarantees that the code underlying the Web service is not being updated; therefore, the inherent trustworthiness of the Web service may be varied with time.

In summary, the flexibility of Web services-centered

Proceedings of the 2004 IEEE International Conference on Services Computing (SCC'04)
0-7695-2225-4/04 $ 20.00 IEEE

computing is not without penalty since the value added by this new paradigm can be largely defeated if: (1) the selected Web service component does not thoroughly fulfill the requirements (i.e., functional and nonfunctional), (2) the hosts of Web services components act maliciously or errantly at invocation times, (3) erratic Internet behaviors or resource scarcity pose unendurable time delays, or (4) the selected Web services components act errantly in the composed environment.

Therefore, our research seeks to explore trustworthiness in the domain of Web services. Throughout this paper, from this point on, *trustworthy computing* refers to trustworthy computing in the domain of Web services. In addition, although computing has broad scope that includes software, hardware, system, services, etc. [7], we focus on software computing only. In this paper, we propose a generic framework safeguarding Web services-centered trustworthy computing. We achieve our goal in the following ways. First, we identify trustworthy entities or resources that are needed to ensure the trustworthiness. Second, we identify and formally represent trustworthy policies. Third, we propose a validation process for trustworthiness. Fourth, we present an environment for trustworthiness management.

The remainder of this paper is organized as follows. In Section 2 we discuss related work. In Section 3, we propose a generic trustworthy framework. In Section 4, we apply our framework to an example. In Section 5, we discuss trustworthy resource layer. In Section 6, we discuss trustworthy policy layer. In Section 7, we discuss trustworthy validation process layer. In Section 8, we discuss trustworthy management layer. In Section 9, we present self-assessments on our framework. In Section 10, we draw conclusions and describe future work.

2. Related work

In recent years, researchers have been doing much work on modeling Web services-oriented system architecture. Generally, all of the proliferating work is built upon eXtensible Markup Language (XML) [16] technology. Among them, Web Services Description Language (WSDL) [13] is the basis of other work. Intending to formally and precisely define a web service, WSDL from W3C (http://www.w3c.org) is becoming the *ad hoc* standard for web services publication.

However, WSDL can only specify limited static information of a web service, such as its abstract interface, bindings to specific message formats and protocols, and the location the service [17]. In recognition of this problem, researchers from both academia and industry have been developing other description languages to extend the power of WSDL to depict Web service architecture. Among the efforts, BEA, IBM, and Microsoft's Business Process Execution Language for Web Services (BPEL4WS) [2] is an outstanding example. BPEL4WS is a programming abstraction that allows developers to compose multiple synchronous and asynchronous Web services into an end-to-end business flow. In other words, it is a language that can be used to specify business processes and business interaction protocols. With built-in support for asynchronous interactions, flow control, and compensating business transactions, BPEL4WS specifies an interoperable integration model aiming at facilitating the automatic integration of Web services components. Collaxa BPEL Server [3], among others, provides a platform to model, connect, deploy and manage BPEL processes.

IBM, Globus, and HP propose the WS-Resource framework to facilitate the universal access of stateful resources contained in Web services [14]. WS-Resource framework not only defines a set of syntax specifications for developers to define stateful resources and associate resources with Web services, but also defines a set of syntax specifications for users of the Web services to access their associated resources.

WS-Security [15] standard proposes a family of protocols that enhances SOAP [10] messaging technique to solve three basic problems about the quality of protection of Web services: authentication and authorization of users, message integrity, and message encryption. Focusing on secure communication, these mechanisms can be used to accommodate a wide range of security models and encryption technologies.

Based upon WS-Security, six enhanced models help to establish secure interoperable Web services [15]: (1) WS-Policy provides a syntax-wired model to specify Web services endpoint policies; (2) WS-Trust defines methods to request and issue security tokens for establishing trust relationships; (3) WS-Privacy specification describes a model for expressing privacy claims inside of WS-Policy descriptions and associating privacy claims with messages; (4) WS-Authorization defines how Web services manage authorization data and policies; (5) WS-SecureConversation defines a security context based upon security tokens for secure communication; and (6) WS-Federation defines mechanisms to enable identity, account, attribute, authentication, and authorization federation across different trust realms.

However, WS-Security and related techniques and languages only address the security issue of Web services-centered computing, while trustworthiness is a holistic property that encompasses many more attributes beyond security, such as reliability, availability, safety, survivability, performance, fault tolerance, etc. In addition, although these WS-Security related languages define a basic set of constructs that can be used and extended by other Web Services specifications to describe a broad range of service requirements, preferences, and

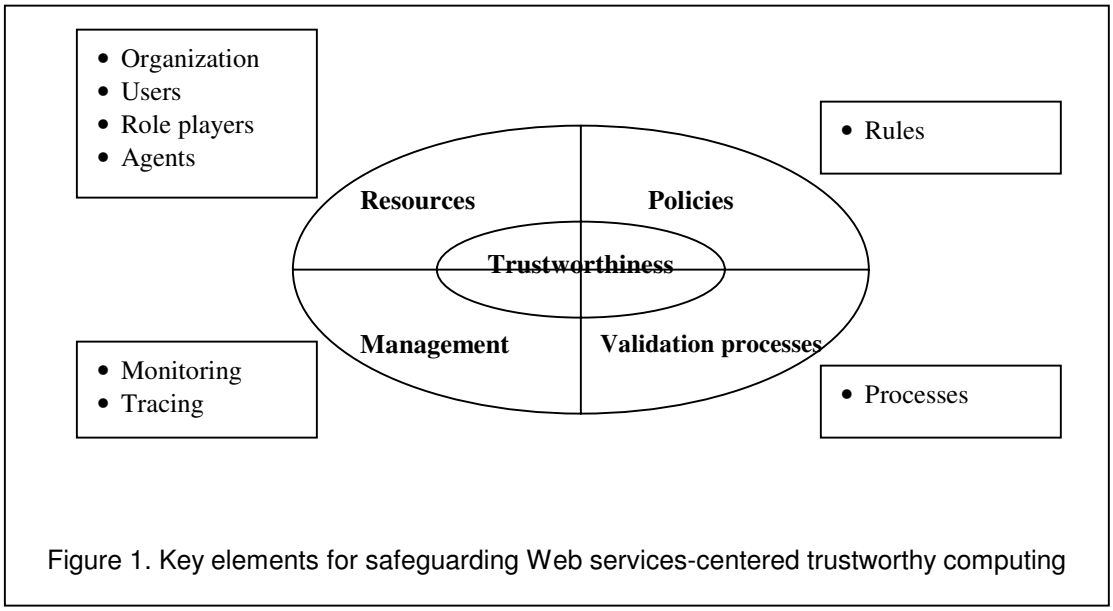

Figure 1. Key elements for safeguarding Web services-centered trustworthy computing

capabilities, none of these languages provides any assertion for Web services endpoint properties.

In contrast with these related work and standards, our research intends to establish a generic framework that proposes a layered model to assure trustworthy computing. This model is oriented to Web services-centered computing, in the sense that layers except the policy layer are equipped with an *ad hoc* Web services standard format or protocol or product that are described above.

3. Framework

In this section, we propose a generic framework to control trustworthiness of Web services-centered computing. We believe that four key elements are imperative to safeguard trustworthy computing, namely, resources, policies, validation processes, and management, as shown in Figure 1:

- **Resources:** The process of computing involves different types of participant entities, such as the organizations, users, people who engage in the software life cycle by acting in different roles (e.g., developers, testers, analysts, etc.), and other entities (e.g., agents if agents technology [9] is adopted). Every entity needs to take responsibility to assure the trustworthiness. Different roles and their responsibilities need to be identified and clearly delineated.
- **Policies:** Policies identify the factors that are likely to compromise the trustworthiness, in other words, what constitute trustworthiness or how these factors can best be measured. Policies should also explicitly address roles and their responsibilities and expected behaviors.
- **Validation processes:** Trustworthiness control involves addressing each factor in order to enhance or control it. These are the procedures that document how policy objectives are to be achieved and verified.
- **Management:** Trustworthiness should be traced and monitored as a programmatic entity throughout the whole life cycle of a Web services-centered project.

Based upon these four elements, a layered framework is proposed to assure trustworthy computing, as illustrated in Figure 2. The framework is composed of four trustworthy layers: resource layer, policy layer, validation process layer, and management layer. Meanwhile, this model is considered to be oriented to Web services-centered computing, in the sense that each layer is equipped with an *ad hoc* Web services standard language or product. In the rest of this section, we briefly describe each layer. More details will be discussed in the following four sections.

The highest layer is the trustworthy resource layer. As shown in Figure 2, different roles are identified in this layer to represent different types of entities involved in the computing; each has its dedicated responsibility to assure trustworthy computing. In this layer, a set of basic roles is predefined, such as organization, user, role player, agent, etc. WS-Resource [14] is associated with the resource layer to formally describe each role.

The second layer is the trustworthy policy layer. As illustrated in Figure 2, multiple polices are identified to express different aspects of trustworthiness requirements. A set of high-level policies is predefined in this layer, such as security, reliability, safety, survivability, etc. Each

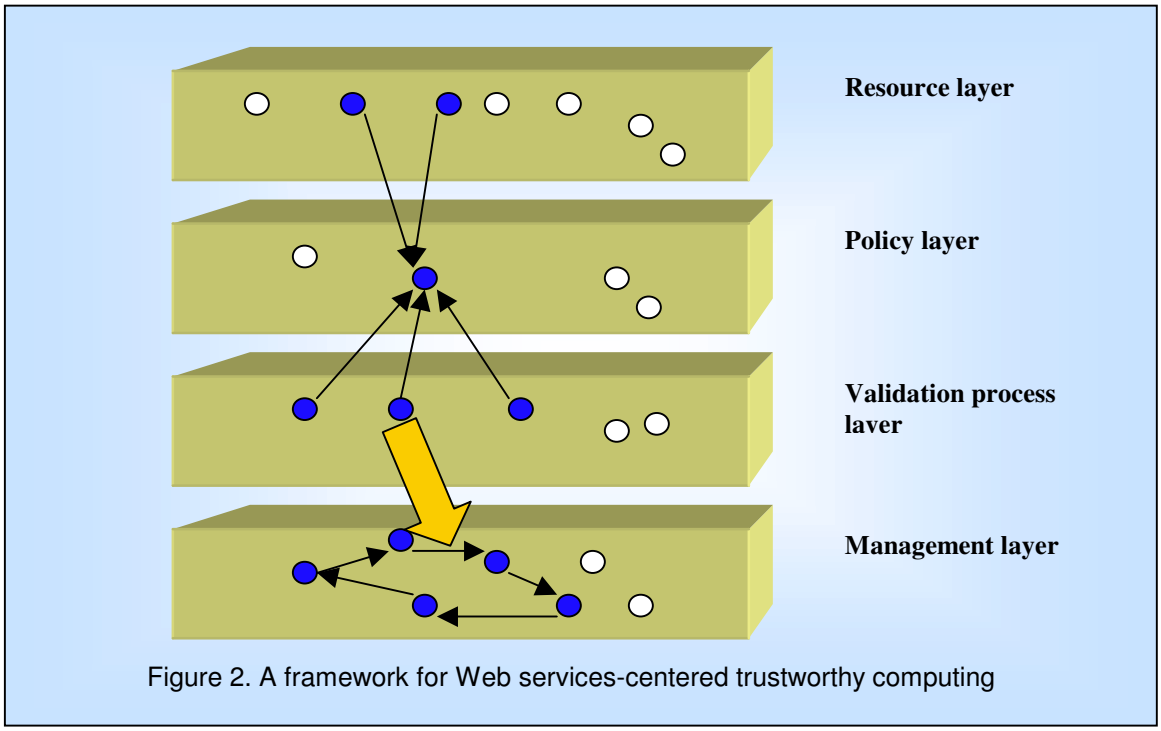

Figure 2. A framework for Web services-centered trustworthy computing

policy may involve multiple roles from the resources layer.

The third layer is the validation process layer, which defines procedures that validate the corresponding policies. As shown in Figure 2, a policy may be validated by different procedures, which may be either independent with each other or related to each other. BEPL4WS [2] is associated with this layer to formally define each validation process.

The fourth and the lowest layer is the management layer, which provides an environment to monitor and track the execution of the validation procedures defined in the validation process layer. In other words, it determines how the trustworthiness processes are to be monitored and logged. As shown in Figure 2, this layer runs each validation process to test the trustworthiness of the system. A check is also enabled to ascertain the effectiveness of the validation processes. The management layer should also include the assessment to ensure that policies and procedures meet the requirements. BPEL Integration Development Environment (IDE) such as Collaxa BPEL Server [3] can be associated with the management layer to execute the validation processes defined using BPEL4WS.

4. Applying the Framework

In the last section, we presented our framework to control trustworthy computing. In this section, we discuss how to apply our framework to assure trustworthy computing.

As Neumann states, the term trustworthiness always refers to the fact that a particular component is worthy of being trusted to fulfill some critical requirements [8]. Therefore, for a specific Web services-centered application system, above all, the set of trustworthiness requirements need to be identified. Then for each identified requirement, a stepwise routine as described below can be followed to apply our framework to establish a trustworthiness assurance measurement. In order to help readers understand our method, we will walk users through an example explained as follows.

Let us consider an application system that utilizes mobile agents technology [9] to facilitate dynamic selection of flight reservation Web services. The policies and validation process and the rationale underneath them are discussed in another paper. Due to the page limitation, here we mostly simplify the system just to illustrate how to apply our framework. Assume that all the candidate flight reservation Web services will return a number at invocation, and the trustworthiness requirement is to only select the candidate services that will not return negative numbers. With the trustworthiness requirement identified, now we are ready to apply our framework.

First, we need to identify trustworthy resources involved with the requirement, and model each resource using WS-Resource. The details of the procedure will be discussed in Section 5.

Second, we need to identify trustworthy policies. The details of the procedure will be discussed in Section 6.

Third, we need to define trustworthy validation processes, and model the processes using BPEL4WS [2].

The details of the procedure will be discussed in Section 7.

Fourth, we need to apply the constructed trustworthiness control model to the Collaxa server platform [3]. The details of the procedure will be discussed in Section 8.

It should be noted that the extent, timing, and documentation of the executable trustworthiness measurement developed for every specific application to satisfy the requirements of our framework will vary and will depend upon many factors, including the size and the nature of the application system.

5. Trustworthy Resource Layer

The process of computing involves different types of participant entities, such as the organizations, users, people who engage in the software life cycle by acting in different roles (e.g., developers, testers, analysts, etc.), and other entities (e.g., agents if agents technology [9] is adopted). Every entity needs to take responsibility to assure the trustworthiness. Different roles and their responsibilities need to be identified and clearly delineated. Earlier studies, such as those by Boehm [1], have revealed that accountability structure (i.e., proper definition of roles and responsibilities) is imperative to ensure the success of a software project. Therefore, this layer will facilitate a role-based trustworthiness assurance.

In our framework, four types of resources are predefined as follows:

- Organizations: This resource refers to both the organizations that are involved with the application system and the ones that provide Web services as components.
- Users: This resource refers to the users of the application system.
- Role players: This resource refers to the people who engage in the software life cycle by acting in different roles (e.g., developers, testers, analysts, etc.), and
- Other entities: This resource refers to other entities involved. For example, if the agents technology [9] is adopted, agents are introduced into the system; thus, agents should be identified as resources.

Considering our example, we can see that all of the four types of resources are needed. Furthermore, we need to further identify the role of trustworthiness testers. As a minimum, three roles are in turn predefined: (1) the program manager (PM) who is accountable for delivering a system that meets the trustworthiness expectation, (2) the technical project officer who is accountable for delivering a system to the PM that meets the PM's stated trustworthiness requirements, and (3) quality assurance (QA) manager who is responsible for developing the QA Plan and for measuring, assessing, and reporting trustworthiness performance against objectives.

It can be easily seen that roles can be considered as WS-Resources due to three characteristics that roles possess: (1) uniqueness: each role has a distinguishable identity and lifetime; (2) Statefulness: each role maintains a specific state that can be materialized using XML; and (3) accessibility: the information of each role should be accessed through one or more Web services to provide another dimension of trust.

We will walk through the definition of a QA manager role *QualityController* using WS-Resource specifications as follows. The definition of a *QualityController* is shown in Figure 3. The state of a *QualityController* is composed of four resource property components: (1) its unique identification number, (2) its responsibility related to trustworthiness, (3) whether the role is mandatory, and (4) the skill set that the role requires. Then its resource properties document, named *QualityControllerproperties*, is defined as shown in Figure 3.

Figure 4 shows how the defined *QualityController* role can be published as part of the Web service by embedding code into the WSDL description of the Web service. Then the WS-Resource properties document declaration of *QualityController* is associated with the WSDL portType definition via the use of the Resource Properties attribute, as highlighted in Figure 4. Later service requestors may

```
<xs: schema

targetNamespace="http://wstc.com/QualityControllerPro
pertiesExample"
   xmlns:tns="http://wstc.com/QualityControllerProperti
esExample"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   …>

   <xs:element name="RoleID" type=…/>
   <xs:element name="Responsibility" type=…/>
   <xs:element name="Mandatory" type=…/>
   <xs:element name="RoleRequirements" type=…/>

   <xs:element name="QualityControllerProperties">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="tns:RoleID"/>
            <xs:element ref="tns: Responsibility"/>
            <xs:element ref="tns: Mandatory"/>
            <xs:element ref="tns: RoleRequirements"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element name="QualityControllerProperties">
```

Figure 3. Code piece 1

```
<wsdl:definitions

targetNamespace="http://wstc.com/QualityControllerProperties"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:wsrp="http://www.ibm.com/xmlns/stdwip/web-services/ws-resourceproperties"
   xmlns:tns="http://wstc.com/QualityControllerProperties"
   …>
   …
      <wsdl:types>
         <xs:schema>
            <xs:import namespace=http://wstc.com/QualityControllerProperties schemaLocation="…"/>
         </xs:schema>
      </wsdl:types>
   …
      <wsdl:portType name="QualityControllerInfo"
   wsrp:ResourceProperties="tns:QualityControllerResourceProperties">
      <operation name="…
   …
      </wsdl:portType>
   …
   </wsdl:definitions>
```

Figure 4. Code piece 2

obtain and examine this XML schema definition of the WS-Resource properties document, which represents the type of stateful resource *QualityController*. A sample SOAP request to the *QualityController* resource is shown in Figure 5. The SOAP message requests two properties of the specific *QualityController* of a system*:* his/her identification number and his/her skill set.

6. Trustworthy Policy Layer

Policies should explicitly address roles and their responsibilities and expected behaviors. Policies identify the factors that are likely to compromise the trustworthiness. Trustworthiness control involves addressing each factor in order to enhance or control it. It is apparent that the second step cannot proceed without the first step being completed successfully. However, there does not appear to be a clear consensus in practice or in the literature as to what constitutes trustworthiness or how these factors can best be measured.

```
<QualityControllerProperties>
   <QCID></QCID>
   <RoleID>QualityController</RoleID>
   <Responsibility>Certify Quality</Responsibility>
   <Mandatory>yes</Mandatory>
   <RoleRequirements></RoleRequirements>
</QualityControllerProperties>
```

Figure 5. Code piece 3

It should be noted that the policy layer does not contain detailed technical information. For example, a policy may require that all SOAP messages sent to a service provider over the Internet be protected. How to realize this policy is a validation process issue though, whether the SOAP message will be encrypted first before being sent to the service provider, or the SOAP message will not be encrypted but will be sent through an encrypted channel.

A set of high-level policies is predefined in this layer, such as security, reliability, safety, survivability, etc. Due to the page limitation, we will not discuss in detail each predefined policy. Instead, we will focus on the specific policy our example requires. In our example, the policy can be informally specified as that, if a candidate flight reservation Web service returns a negative value, the mobile agent should alert the service requestor to abandon the candidate.

7. Trustworthy Validation Process Layer

Validation processes are meant to provide reasonable assurance that the system of trustworthiness control is relevant, adequate, and complied with in practice. The validation processes normally include the development of a general strategy and the preparation of a detailed approach to the corresponding policies and may also outline the supervision and review responsibilities and other trustworthiness control procedures specific to the trustworthiness requirement.

Compared to the policy layer, the validation process layer is less stable. It is unlikely that the policies will change radically oftentimes. On the other hand, however, due to the ever-evolving technologies and products, the validation process layer may change on a regular basis to adapt to new technological changes. It should be noted that the different validation processes that are associated with the same policy should achieve the same objective.

As a language that can be used to specify business processes and business interaction protocols, BPEL4WS [2] can be used to model the validation process. Let us walk through our example to illustrate how to use BPEL4WS to model validation processes. If the remote flight reservation Web service generates a negative number as return, it should be considered as an exception. The agent needs to handle the *NegativeNo* fault then. As shown in Figure 6, the agent uses the <invoke> section to invoke the Web service *FlightRESerivce*. A

```
<scope containerAccessSerializable="no">
 <faultHandlers>
  <catch faultContainer="frError"
faultName="frs:NegativeNo">
    <!—send assertion to the service requestor -->
…
  </catch>
 </faultHandlers>
 <sequence>
  <invoke inputContainer="frInput" name="flightRE"
operation="process" outputContainer="frOutput"
partner="flightREService"
portType="frs:FlightREService"/>
…
 </sequence>
</scope>
```

Figure 6. Code piece 4

<faultHandlers> section is embedded to handle the exception. The fault handlers define the activities that must be performed in response to *NegativeNo* faults resulting from the invocation of the assessment and approval services. The WSDL fault *NegativeNo* is identified by a qualified name formed by the target namespace of the WSDL document in which the relevant portType and fault are defined.

8. Trustworthy Management Layer

The purpose of monitoring and tracking the application of trustworthiness control policies and procedures is to obtain reasonable assurance that the system of trustworthiness control is suitably designed and effectively applied. Monitoring involves an ongoing consideration and evaluation of: (1) the relevance and adequacy of the trustworthiness control policies and validation procedures, (2) the appropriateness of the resources provided, (3) compliance with trustworthiness control policies and validation procedures, and (4) the consistency of the policies and validation procedures with the developments.

BPEL Integration Development Environment (IDE), such as Collaxa BPEL Server [3], can be used to execute the validation processes defined using BPEL4WS. In more detail, as a validation process written in BPEL4WS is inputted into a Collaxa server, the Collaxa server has the built-in ability to (1) test validation process by examining the state of BPEL process instances, (2) track execution and capture the history of the validation process, and (3) monitor the validation process by aggregating statistical information.

It should be noted that the responsibility for monitoring the application of trustworthiness control policies and procedures is different from the overall responsibility for trustworthiness control. Therefore, whenever possible, it is desirable that the two responsibilities be assigned to different roles and individuals.

Monitoring and tracking can also reveal deficiencies of trustworthiness control policies and procedures. Thus further investigations or corrective actions can be performed based upon the execution of the validation processes.

9. Discussions

In this section we discuss the merits and limitations of our framework.

9.1 Merits of the framework

The robustness of this model is its layered approach where each layer is built upon an organization foundation and *ad hoc* standards.

Web services-centered computing normally involves a federation of organizations and technologies; therefore, it is nearly impossible to deliver a solitary set of polices and validation processes that cover a multiplicity of diverse organizations, requirements, skills, resources, and technologies. Instead, in our model, policies and validation processes are separated. The same policy can be implemented and validated by different processes. It should be noted that the policy layer does not contain detailed technical information; instead, this layer describes what needs to be achieved but not how.

Our framework can be adapted and extended to suit the needs of adopting trustworthiness requirements. As we illustrated in the example, on one hand, the framework is general and comprehensive enough to be used as a generic guidance to construct an executable model to assure trustworthiness requirements; on the other hand, the framework is made as lean as possible while still fulfilling its mission to help safeguard predictably trustworthy software.

Since Web services-centered computing seeks interoperability between components on a worldwide basis, information data are defined based upon standard data formats and protocols. Seamlessly incorporating the most recent standards and typical tools, in our framework, each layer (except trustworthy policy layer) is associated with an *ad hoc* standard language or platform. Therefore, our framework provides a practical guidance of establishing trustworthiness assurance measurement. It should be noted that the language or tool associated with each layer can be replaced by other products, while the concept of our framework still applies. Therefore, our framework will neither impinge upon software vendors' flexibility nor thwart enterprise autonomy.

Finally, our framework should interest organizations

that pursue formal trustworthiness control, since our framework defines a robust system of trustworthiness control that should result in a number of benefits to an organization as it:

- Promotes consistency amongst staff within an organization by formalizing trustworthiness control policies and making the work more effective and efficient;
- Provides guidance on the assignment of an assurance team that collectively possesses the necessary competencies and skills to complete the trustworthiness requirement;
- Improves the quality of performance of building trustworthiness measurement;
- Establishes professional standards that can be used as a benchmark by those who monitor the performance of organizations and practitioners;
- Enables the identification and evaluation of threats to software trustworthiness; therefore the organization can take appropriate actions to eliminate those threats or reduce them to an acceptable level by the application of safeguards, and monitor the compliance with the applicable trustworthiness requirements; and
- Provides additional confidence amongst those who use practitioner reports issued by an organization.

9.2 Limitations of the framework

It should be noted that our framework provides a high level guidance of establishing trustworthiness control. Each layer of the model for a specific application system needs to be manually created. The quality of the model to be built is fully dependent on the experience of the practitioners. In order to make our framework more practical, we need an integrated development environment tailored to the framework.

In addition, our current work only provides some high level instructions to apply our framework to build an infrastructure of trustworthiness control. A methodology is needed to provide more detailed guidance for practitioners.

Furthermore, in our current framework we do not formally define trustworthy policies. We know that WS-Policy [15] is fully based upon XML and does not place limits on the types of requirements and capabilities of Web services that can be described. Therefore, the specification syntax should be extended to identify other attributes, such as the trustworthy policies.

10. Conclusions and Future Work

In this paper we propose a generic framework to assure Web services-centered trustworthy computing. Our framework exhibits robust flexibility and extensibility to guide the measurement of software trustworthiness. Our future work will include: (1) constructing framework tailored integrated development environment to help build the model, (2) exploring extending WS-Policy to formally define trustworthy policies, and (3) conducting more case studies.

11. References

[1] B.W. Boehm, "R&D Trends and Defense Needs", *Research Directions in Software Engineering,* Ed. by P. Wegner, 1979, Cambridge, MA: MIT Press.

[2] IBM Corporation, "Business Process Execution Language for Web Services (BPEL4WS)," Version 1.0, 2002.

[3] http://www.collaxa.com.

[4] C. Ferris and J. Farrell, "What Are Web Services?", *Communications of the ACM*, 46(6), Jun. 2003, pp. 31.

[5] Bill Gates, "Trustworthy Computing", email, http://www.wired.com/news/business/0,1367,49826,00.html.

[6] N. Gold, C. Knight, A. Mohan, and M. Munro, "Understanding Service-Oriented Software", *IEEE Software*, Mar./Apr. 2004, pp. 71-77.

[7] C. Mundie, P. ve Vries, P. Haynes, and M. Corwine, "Trustworthy Computing", Microsoft White Paper, revised version, http://www.microsoft.com/mscorp/innovation/twc/twc_whitepaper.asp.

[8] Peter Neumann, "Principled Assuredly Trustworthy Composable Architectures", emerging draft of the final report for DARPA's Composable High-Assurance Trustworthy Systems (CHATS) program, http://www.csl.sri.com/users/neumann/chats4.pdf.

[9] A. Pham and A. Karmouch, "Mobile Software Agents: An Overview", *IEEE Communications magazine*, 36(7), Jul. 1998, pp. 26-37.

[10] http://www.w3.org/TR/SOAP.

[11] http://www.uddi.org/specification.html.

[12] http://www.microsoft.com/windowsserver2003/developers/top10fordevs.mspx.

[13] http://www.w3.org/TR/wsdl.

[14] http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf.

[15] http://www-106.ibm.com/developerworks/webservices/library/ws-secure.

[16] .B. McLaughlin and M. Loukides, *Java and XML*, O'Reilly Java Tools, 2001.

[17] L.-J. Zhang, "Challenges and Opportunities for Web Services Research", *International Journal of Web Services Research (JWSR)*, 1(1), 2004.