From Conflict of Interest to Separation of Duties in WS-Policy for Web Services Matchmaking Process

Patrick C. K. Hung
Commonwealth Scientific and Industrial Research Organization (CSIRO)
ICT Research Centre, GPO Box 664, Canberra, ACT 2601, Australia
Phone: +612 6216 7031 Fax: +612 6216 7111
Patrick.Hung@csiro.au

Abstract

A Web service is defined as an autonomous unit of application logic that provides either some business functionality or information to other applications through an Internet connection. Web services are based on a set of XML standards such as Simple Object Access Protocol (SOAP), Universal Description, Discovery and Integration (UDDI) and Web Services Description Language (WSDL). Web services architectures are built on an insecure, unmonitored and shared environment, which is open to events such as security threats. Security concerns are the major barrier that prevents many business organizations from implementing or employing Web services. Based on the previous research in Conflict of Interest (CIR), this paper further introduces another important security concept called Separation of Duties (SoD) for Web Services Matchmaking Process (WSMP). Next, this paper discusses the relationships between CIR and SoD in the context of matchmaking process. The paper then extends these two concepts into specifying and implementing CIR and SoD assertions in the newly developed WS-Policy. WS-Policy is an XML representation that provides a grammar for expressing Web services policies, to allow service locators to have a common interpretation of security requirements in the matchmaking process. Further, this paper presents a matchmaking algorithm for maximizing the level of SoD. Lastly, this paper also describes a prototype framework with "CIRService" and "SoDService" services for supporting CIR and SoD assertions in matchmaking process.

Keywords: Web services, conflict of interest, separation of duties, WS-Policy, WS-PolicyAttachment, matchmaking, delegation, security assertion, service locator.

1. Introduction

A Web service is defined as an autonomous unit of application logic that provides either some business functionality or information to other applications through an Internet connection. Web services are based on a set of XML standards such as Simple Object Access Protocol (SOAP), Universal Description, Discovery and Integration (UDDI) and Web Services Description Language (WSDL). Some studies [5] show that the Web services market is expected to grow to USD$28 billion in sales in the coming three years. In particular, Grid technologies and infrastructures increase the need for sharing and coordinating the use of Web services for different business processes in a loosely coupled execution environment. A business process contains a set of activities which represent both business tasks and interactions between Web services. Web services applications are growing in popularity. It is believed that early adopters of Web services may include several industries that involve a set of diverse trading partners working closely together in a highly competitive market such as insurance, financial services and high technology industries [16].

Traditional business-to-business applications connect trading partners through a centralised architecture. A major drawback is that setting up an additional connection with another trading partner is costly and time consuming. In contrast, the benefits of adopting Web services include faster time to production, convergence of disparate business functionalities, a significant reduction in total cost of development and easy to deploy business applications for trading partners [16]. Another difference between traditional business-to-business applications and Web Services is a secure environment versus an exposed environment. Although WSDL and UDDI address the issues of discovery for trading partners (e.g., Web services), they do not discuss any approach to detecting conflicts, dissatisfaction and mistrust among trading partners [16].

Security concerns are the major barrier that prevents many business organizations from implementing or employing Web services. Based on the previous research in Conflict of Interest (CIR) [7], this paper further introduces another important security concept called Separation of Duties (SoD) for Web Services Matchmaking Process (WSMP). Next, this paper discusses the relationships between CIR and SoD in the context of matchmaking process. The paper then extends these two concepts into specifying and implementing CIR and SoD assertions in the newly developed WS-Policy.

WS-Policy is an XML representation that provides a grammar for expressing Web services policies, to allow service locators to have a common interpretation of security requirements in the matchmaking process. Further, this paper presents a matchmaking algorithm for maximizing the level of SoD. The remainder of this paper is organized as follows. The next section gives an overview of related work and background knowledge. Section 3 presents the concept of CIR in WSMP. Next, Section 4 discusses the concept of SoD in WSMP and also presents an algorithm for WSMP. Then, Section 5 demonstrates specifying CIR and SoD assertions in WS-Policy and describes a related prototype framework. Lastly, Section 6 concludes with identification of further research.

2. Related Work and Background Knowledge

In the past few years, business process or workflow proposals relevant to Web services are proliferating in the business and academic world [20]. All of the proposed XML languages are based on WSDL service descriptions with extension elements. For example, Thatte [18] describes XLANG as a notation for the specification of message exchange behaviors (i.e., interactions) among participating Web services in business processes. XLANG describes the behaviour of the Web services as a part of a business process. In XLANG, the behaviour of each Web service is specified independently and the interactions between Web services is only through message exchanges expressed as operations in WSDL.

Similarity, the Web Services Flow Language (WSFL) [12] is also an XML language for the description of Web services interactions. WSFL specifies the appropriate usage pattern of a collection of Web services in order to achieve a particular business goal, and WSFL also specifies the interaction pattern of a collection of Web services. In addition to WSFL, Leymann [12] also proposes an XML language called the Web Services Endpoint Language (WSEL) for describing Web services endpoint properties such as time and contact information, where it is used to match the expectations from WSFL to the promises from WSDL.

Next, the Web Service Choreography Interface (WSCI) [17] describes the flow of messages exchanged by a Web service participating in interactions with other Web services. In particular, WSCI describes the dynamic interface of the Web service participating in a given message exchange by means of reusing the operations defined for a static interface. Further, the Business Process Execution Language for Web Services (BPEL4WS) [8] is a formal specification of business processes and interaction protocols. BPEL4WS defines an interoperable integration model that facilitates the expansion of automated process integration in an intra-corporate and inter-corporate environment. Lastly, WS-Coordination [10] defines an extensible framework for coordinating activities using a set of coordination protocols. Based on WS-Coordination, WS-Transaction [9] presents an XML language to describe an atomic transaction that is used to coordinate activities in a short period of time and also a business activity that is used to coordinate activities in a long period of time by applying business logic. In summary, all these XML languages facilitate defining Web services interacted activities in the format of a flow model.

No matter which specific XML language is used to define a flow model by a business process modeller, each of the activities in a flow model must be executed by an appropriate Web service. In this scenario, the role of service locators is to assign an appropriate Web service for each activity. This assignment process is called matchmaking. Furthermore, value-added Web services are required to be enacted by long duration multi-step activities. Thus Web services may also delegate some sub-activities that are decomposed from the assigned activities to other Web services. This assignment process is called delegation [11]. In general, both assignment processes are expected to use the service registry to find the most appropriate Web service to satisfy activities' or even sub-activities' requirements.

Web services architectures are built on an insecure, unmonitored and shared environment, which is open to events such as security threats. This may result in conflicts since the open architecture of Web services makes it available to many parties, who may have competing interests and goals [16]. For example, a party's commercial secrets may be released to another competing company via the Web services execution. As is the case in many other applications, the information processed in Web services might be commercially sensitive so it is important to protect it from security threats such as disclosure to unauthorized parties. Since security is an essential and integral part of many business processes, Web services have to manage and execute the activities in a secure way. However, the research area of Web services security is challenging as it involves many disciplines, from authentication/encryption to access management/security policies. Security concerns and the lack of security conventions are the major barriers that prevent many business organizations from implementing or employing Web services. An Evans Data Corporation survey of 400 enterprise development managers, conducted in January 2002, showed that 45.5% of the managers regarded security and authentication issues to be the biggest obstacle to Web services implementation [3].

As with the XML languages discussed above, there are also XML languages proposed for describing security assertions in Web services. These XML languages restrict access to Web services to authorized parties only and protect the integrity and confidentiality of messages

exchanged in a loosely coupled execution environment. For example, ebXML [4] is an XML language used to enable the global use of electronic business information in an interoperable, secure and consistent manner by all parties. Specifically there are two well-known formats for XML-based security tokens: the Security Assertions Markup Language (SAML) [14] and the eXtensible rights Markup Language (XrML) [2]. SAML is used to define authentication and authorization decisions in Web services. Web services providers submit SAML tokens to security servers for making security decisions. In addition, a Java-based toolkit called JSAML [13] is developed for supporting SAML in e-business applications. SAML is an XML-based framework for exchanging security credentials in the form of assertions about subjects. Similarity, XrML assists the owners of Web services to specify the rights of authorized users or parties and to identify the terms and conditions under which those rights may be exercised by those authorized users or parties. Lastly, WS-Security [11] describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality and single message authentication. From another point of view, WS-Security is the messaging language and SAML is the security language. Similar to ebXML, WS-Security mainly focuses on secure communication. Though BPEL4WS recommends that business process implementations use WS-Security for secure messaging, none of these languages discussed above provides any security assertion for Web services endpoint properties.

Based on WS-Security, WS-Policy [11] provides a grammar for expressing Web services policies. The WS-Policy includes a set of general messaging-related assertions defined in WS-PolicyAssertions [11] and a set of security policy assertions related to supporting the WS-Security specification defined in WS-SecurityPolicy [11]. In addition to the WS-Policy, WS-PolicyAttachment [11] defines how to attach these policies to Web services or other subjects such as service locators. WS-Authorization [11] defines how Web services manage authorization data and policies. The eXtensible Access Control Markup Language (XACML) [15] defines the fine-grained authorization and entitlement policies between subjects and resources. WS-Trust [11] defines methods for issuing and exchanging security tokens for establishing the presence of trust relationships. Lastly, WS-SecureConversation [11] defines a security context based on security tokens for secure communication. In conclusion, none of these languages proposes any security assertion for the matchmaking process between activities and Web services. This is the major motivation of this paper.

3. Concept of Conflict of Interest

By convention, security threats are usually thought to come from outsiders. In many cases, however, security problems arise in a well-control environment from authorized insiders. This means that a secure environment must not only ensure that Web services are trusted but must also deal with other security threats such as conflict of interest. Webster's Dictionary defines "conflict of interest" as a conflict between the private interests and the official responsibilities of a person in a position of trust. In fact, the concept of CIR has been studied in other research fields for some time. In bargaining games, CIR is a property of the preferences of the participants and the structure of the situation in which they find themselves. One of the classical security policies to deal with CIR is Chinese wall security policy [1] for financial applications. Chinese wall security policy contains a set of access control rules such that no person can ever access data on wrong side of that wall. Some other prior research also investigates CIR in different situations. For example, Nyanchama and Osborn [21] study CIR in privilege-privilege and role-role conflicts in a role graph model. Further, Ahn and Sandhu [22] describe a framework for specifying CIR policies in role-based systems. In addition, they present a RSL99 language for specifying role-based constraints. However, none of these works studies the effects of CIR in matchmaking process, especially in the context of Web services execution environment.

Though some Web services are capable to execute certain activities, there may have certain constraints that prohibit them executing a particular activity in a particular situation. Thus a matchmaking model with the consideration of security assertions is required in such a sophisticated Web services execution environment. To understand this work, this paper develops an abstract model for a business process as follows. A business process is represented into a flow model (FM) that contains a partially ordered set of activities (A) that is coordinated by a set of data/control flows. The order of activity execution is orchestrated by matching the input and output flow(s) of each activity. Each activity represents a piece of work (i.e., a sequence of operations) that needs to be done by a Web service. Each Web service may have a delegation model that contains a set of Web services (WS) in a hierarchical or peer-to-peer structure. Let sets of activities (A), Web services (WS), and flow models (FM), respectively, be:

- $A = \{a_1, a_2, \ldots, a_m\}$ is the set of *m* activities.
- $WS = \{ws_1, ws_2, \ldots, ws_n\}$ is the set of *n* Web services.
- $FM = \{fm_1, fm_2, \ldots, fm_p\}$ is the set of *p* flow models.

The relationships among different entities are the following:

- C: FM $\rightarrow$ A gives a partially ordered set of activities that is contained in a flow model. To illustrate, $C(fm_i) = \{a_{i1}, a_{i2}, \ldots, a_{ik}\}$ is the partially ordered set of *k* activities that is contained in the flow model fm_i and $\forall_{i,j}$

$C(fm_i) - C(fm_j) \neq \emptyset$, where $C(fm_i) \cup C(fm_j) \subseteq A$, fm_i, $fm_j \in FM$ and $i \neq j$.

- M: $A \rightarrow WS$ is a one-to-one mapping that gives a Web service that is assigned to execute an activity. To illustrate, $M(a_i) = ws_i$ is the Web service that is assigned to execute the activity a_i, where $M(a_i) \in WS$ and $a_i \in A$.
- D: $WS \rightarrow$ BOOLEAN (i.e., true or false) tells whether a Web service contains a delegation model or not. To illustrate, $D(ws_i)$ = "true" means that the Web service ws_i contains a delegation model, where $ws_i \in WS$.
- DM: $WS \rightarrow WS$ gives a set of Web services invoked in a delegation model if and only if the Web service contains a delegation model. To illustrate, $DM(ws_i) = \{ws_{i1}, ws_{i2}, \ldots, ws_{ik}\}$ is the set of *k* Web services invoked in the delegation model of Web service ws_i, where $D(ws_i)$ = "true" and $DM(ws_i) \subseteq WS$.
- MM: $FM \rightarrow WS$ gives a set of Web services invoked in the FM. To illustrate, $MM(fm_i) = \{ws_{i1}, ws_{i2}, \ldots, ws_{ik}\}$ is the set of *k* Web services invoked in the matchmaking model of the flow model fm_i. Note that this set does not include those Web services involved in the delegation models, i.e., DM. To illustrate, $MM(fm_i) = \{ws_{ij} \mid \forall\ ws_{ij} = M(a_{ij})$ where $a_{ij} \in C(fm_i)\}$, where $MM(fm_i) \subseteq WS$.

Service locators are required to ensure that the commercial secrets of clients do not leak via Web services execution. Figure 1 shows a business process (a flow model) called "Patient Health Records Integration Process" that includes three activities "Retrieve Patient Health Records at Hospital 1," "Retrieve Patient Health Records at Hospital 2" and "Integrate Patient Health Records." The patient health records often contain sensitive and identifiable information about patients. The sensitive or identifiable attributes from those patient health records should have been protected properly, e.g., encrypting the patient identifiers in the dataset. Assume that there exist two Web services A and B, which can provide the services to retrieve the patient health records at hospital 1 and 2 respectively. Also assume that both Web services, A and B, can provide the services to integrate the patient health records from both datasets by a common key such as Social Security Number. Note that neither Web service A nor B is allowed to release those patient health records to the other to protect patient's privacy. If Web Service A or B can access the aggregate of patient health records (from both hospital 1 and 2), it may be possible for Web service A or B to infer a patient's identity by using data mining techniques (i.e., either patient health records at hospital 1 and 2). Therefore, neither Web service A nor B is allowed to execute the consequent "Integrate Patient Health Records" activity because conflict of interest arises among these three activities. As a result, a third party is required, such as Web service C, to execute the "Integrate Patient Health Records" activity, i.e., (Web service A ≠ Web service C) ∧ (Web service B ≠ Web service C) ∧ (Web service A ≠ Web service B).

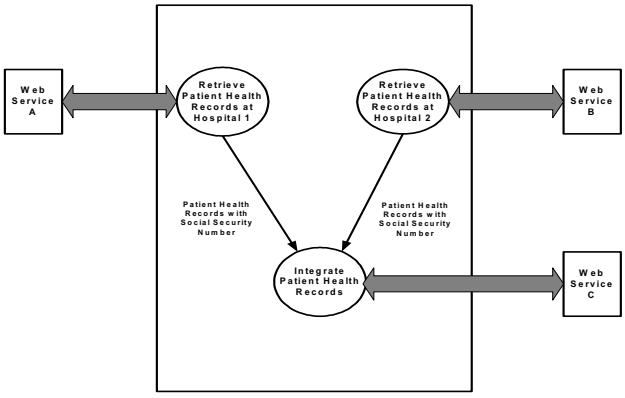

Figure 1. A Patient Health Records Integration Process Example.

In this paper, CIR is represented as a sequence of notation in the format of first order predicate calculus. This paper discusses CIR into a multi-lateral relation with exclusive-or (XOR) logic. Referring to the example of "Patient Health Records Integration Process" in Figure 1, the business process modeler can identify CIR in the matchmaking pattern as follows:

M("Retrieve Patient Records at Hospital 1"**) = ws$_i$** ⊕
M("Retrieve Patient Records at Hospital 2"**) = ws$_i$** ⊕
M("Integrate Patient Health Records"**) = ws$_i$**

where XOR is represented with the symbol "⊕" and this statement means that if a_1 is assigned to ws_a, then a_2 and a_3 cannot be assigned to ws_a and so on. However, the issues of CIR have not been widely investigated in the context of matchmaking process for Web services. In the specification phase, the business process modelers specify CIR as a set of security assertions in the Web services endpoint properties. Thus the matchmaking process should be enhanced with the security assertions in the Web services endpoint properties. In many situations, a Web service may be authorized to execute more than one activity in a flow model.

4. Concept of Separation of Duties

SoD is used as a security principle to formulate multi-person control policies, requiring that two or more different people be responsible for the completion of an activity or set of related activities [23]. The purpose of this principle is to reduce the possibility for fraud or significant errors by partitioning of activities and associated privileges [22]. For example, Nash and Poland [24] study SoD in the context of data objects within a system by separating all operations into several sub-parts

and requiring that each sub-part be executed by a different person. Moreover, a number of researchers discuss the concept of SoD in role-based systems by controlling membership in, activation of, and use of roles as well as permission assignment. For example, Kuhn [25] discusses the mutual exclusive roles in both authorization-time and run-time situations. In the authorization-time situation, the roles which have been specified as mutually exclusive cannot both be included in a user's set of authorized roles. In the run-time situation, users may be authorized for two roles that are mutually exclusive, but cannot have both roles active at the same time in a session. Next, Simon and Zurko [23] discuss the static and dynamic SoD. In the static SoD, no one person is ever allowed to perform two or more exclusive roles. However, the dynamic SoD allows users to act in roles that would be strongly exclusive in static systems, as long as some constraints for eliminating the possibility of fraud are satisfied. Similarly, Gligor et al. [26] also discusses a list of static and dynamic SoD properties. However, none of these works discusses the relationships between CIR and SoD in details and also applies these concepts in the context of Web services.

To minimize the level of security risk [27], a good WSMP process should try to constrain the number of activities executed by a Web service. SoD is one of the important concepts of fraud and error control in these situations [28]. Increasing the level of SoD in matchmaking process means that several Web services have to be involved in performing a flow model independently and no individual Web service can misuse privileges by acting alone. In a flow model, the level of SoD increases as the number of Web services involved increases. In general, the level of SoD can be simply defined as $\frac{|MM(fm)|}{|C(fm)|}$. The best case is each activity involved in the flow model is executed by a Web service separately, i.e., $\forall_{i,j}$ $M(a_i) \neq M(a_j)$, where $a_i, a_j \in C(fm)$. In this case, the level of SoD is 1 because the maximum value of $|MM(fm)|$ is $|C(fm)|$, i.e., $\frac{|C(fm)|}{|C(fm)|}$. If there exists any Web service that is matched with more than one activity in the flow model, the level of SoD is less than 1 because of $|MM(fm)| < |C(fm)|$, i.e., $\exists_{i,j}$ $M(a_i) = M(a_j)$, where $a_i, a_j \in C(fm)$. Because of the issues of incomplete information discussed above, this model does not consider the Web services involved in the delegation models. In fact, SoD is enacted whenever CIR arises in matchmaking models between Web services and activities [26]. Here is an example to demonstrate a situation where the level of SoD may increase or decrease while the instances of CIR increases. For simplicity, there is a Web service, say ws_{org}, that is assigned for two activities, say a_{c1} and a_{c2} in a flow model fm, i.e., $|M^{-1}(ws_{org}) \cap C(fm)| = 2$. There always exists a Web service ws_i, where $ws_i \in MM(fm)$ or $ws_i \notin MM(fm)$, that is available and capable for those two conflicting activities. In this case, there are two effective options that a service locator can perform:

- **Option I:** Assign one of the activities to another new appropriate Web service, either $M(a_{c1}) = ws_i$ or $M(a_{c2}) = ws_i$, where $ws_i \notin MM(fm)$ and $ws_i \in WS$.
- **Option II:** Assign one of the activities to another existing appropriate Web service, either $M(a_{c1}) = ws_i$ or $M(a_{c2}) = ws_i$, where $ws_i \in MM(fm)$.

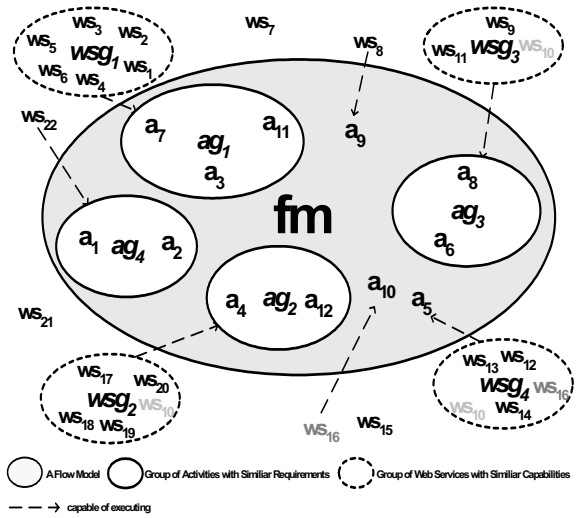

Figure 2. A Matchmaking Pattern Example.

Here an appropriate Web service means that its capabilities can satisfy the activity's requirements. In the first option, as long as the new appropriate Web service does not belong to the existing MM(fm), it will definitely increase the level of SoD. It is obvious that the number of Web services invoked will be increased by one, i.e., $MM(fm)' = MM(fm) \cup ws_i$ and $|MM(fm)'| > |MM(fm)|$. However, it is also obvious that Option II does no effect on the level of SoD because the number of Web services would be the same as before, i.e., $|MM(fm)'| = |MM(fm)|$. For a further example in more than two activities, say *n* activities, the above two options can also be applied on a bilateral basis. For example, three conflicting activities a_{c1}, a_{c2} and a_{c3} can be handled in a pattern of (a_{c1} and a_{c2}) and then ((a_{c1} or a_{c2}) and a_{c3}). To maximize the level of SoD, the service locator should assign (*n* - 1) conflicting activities to other (*n* - 1) new appropriate Web services.

Figure 2 shows an example of matchmaking pattern for a flow model. In this case, there are many possible matchmaking patterns of assigning Web services to activities. Besides those individual Web services and activities, there are two types of groups in a flow model:

- $AG = \{ag_1, ag_2, \ldots, ag_m\}$ is the set of *m* activity groups and each activity group contains a set of activities with

similar requirements, where $\forall_{i,j}\ ag_i \cap ag_j = \varnothing$, $|ag_i| > 1$, $|ag_j| > 1$, $ag_i \subseteq C(fm)$, $ag_j \subseteq C(fm)$ and $ag_i, ag_j \in AG$.

- WSG = $\{wsg_1, wsg_2, \ldots, wsg_n\}$ is the set of *n* Web services groups and each Web services group contains a set of Web services with similar capabilities, where $wsg_i \cap wsg_j \neq \varnothing$ or $wsg_i \cap wsg_j = \varnothing$, $|wsg_i| > 1$, $|wsg_j| > 1$, $wsg_i \subseteq WS$, $wsg_j \subseteq WS$ and $wsg_i, wsg_j \in WSG$.

Though the issues of grouping Web services and activities are out of scope in this paper, there exist certain emergent technologies such as Business Explorer for Web Services (BE4WS) [29] for searching Web services that can be further extended to incorporate these features. Figure 2 shows the flow model that contains 12 activities, i.e., C(fm) = $\{a_1, a_2, \ldots, a_{12}\}$, and also it shows a set of 22 Web services, i.e., WS = $\{ws_1, ws_2, \ldots, ws_{22}\}$. Note that the Web services groups *2*, *3* and *4* have a common Web service, i.e., $wsg_2 \cap wsg_3 \cap wsg_4 = \{ws_{10}\}$. On the other hand, a Web service can stand individually and also belong to a Web services group, i.e., $ws_{16} \in wsg_4$. In the flow model, the Capable of Executing (CoE) is represented in dashed arrows (Figure 2) between a Web service and an activity (Case I), a Web service and an activity group (Case II), a Web services group and an activity (Case III), and a Web services group and an activity group (Case IV), if and only if:

$$\begin{aligned}&(\exists\ wsg_i \in WSG)(\exists\ ag_i \in AG)(\exists\ ws_i \in WS)\\&(\exists\ a_i \in C(fm)):-\\&\quad((\exists\ CoE_{Case\_I}(ws_i, a_i) \Rightarrow (\exists\ ws_i \rightarrow a_i)) \vee\\&\quad(\exists\ CoE_{Case\_II}(ws_i, ag_i) \Rightarrow (\forall_j\ a_{ij} \in ag_i)(\exists\ ws_i \rightarrow a_{ij})) \vee\\&\quad(\exists\ CoE_{Case\_III}(wsg_i, a_i) \Rightarrow (\forall_j\ ws_{ij} \in wsg_i)(\exists\ ws_{ij} \rightarrow a_i)) \vee\\&\quad(\exists\ CoE_{Case\_IV}(wsg_i, ag_i) \Rightarrow (\forall_j\ ws_{ij} \in wsg_i)(\forall_j\ a_{ij} \in ag_i)\\&\quad(\exists\ ws_{ij} \rightarrow a_{ij})))\end{aligned}$$

where "capable of executing" is represented with the symbol "$\rightarrow$." Referring to Figure 2, these four cases can be illustrated as follows:

- *Case I*: $CoE_{Case\_I}(ws_8, a_9)$ and $CoE_{Case\_I}(ws_{16}, a_{10})$. This means that the Web service ws_8 and ws_{16} is capable of executing the activity a_9 and a_{10}, respectively.
- *Case II*: $CoE_{Case\_II}(ws_{22}, ag_4)$. This means that the Web service ws_{22} is capable of executing the activities a_1 and a_2 in the activity group ag_4.
- *Case III*: $CoE_{Case\_III}(wsg_4, a_5)$. This means that all Web services ws_{10}, ws_{12}, ws_{13}, ws_{14} and ws_{16} in the Web services group wsg_4 are capable of executing the activity a_5.
- *Case IV*: $CoE_{Case\_IV}(wsg_1, ag_1)$, $CoE_{Case\_IV}(wsg_2, ag_2)$ and $CoE_{Case\_IV}(wsg_3, ag_3)$. This means that any Web service ws_{1i}, ws_{2i} and ws_{3i} in the Web services groups wsg_1, wsg_2 and wsg_3 is capable of executing any activity a_{1j}, a_{2j} and a_{3j} in the activity groups ag_1, ag_2 and ag_3, respectively, i.e., $\forall\ ws_{1i} \in wsg_1$, $\forall\ ws_{2i} \in wsg_2$, $\forall\ ws_{3i} \in wsg_3$, $\forall\ a_{1j} \in ag_1$, $\forall\ a_{2j} \in ag_2$ and $\forall\ a_{3j} \in ag_3$.

Figure 3 shows a WSMP algorithm for maximizing the level of SoD in the flow model fm with a set of Web services. In the initialization section, the functions "$CoE_{Case\_I}(C(fm), WS)$," "$CoE_{Case\_II}(C(fm), WS)$," "$CoE_{Case\_III}(C(fm), WS)$" and "$CoE_{Case\_IV}(C(fm), WS)$" are used to classify the set of activities involved in the flow model (i.e., C(fm)) and the set of Web services (i.e., WS) into those four cases "Case_I," "Case_II," "Case_III" and "Case_IV" in the format of tuples (ws_i, a_i), (ws_i, ag_i), (wsg_i, a_i) and (wsg_i, ag_i), respectively. For all cases, the function "CONFLICT(ws_i, a_i)" is used to check whether there exists any CIR for each activity in the flow model fm, i.e., $a_i \in C(fm)$. If there exists CIR, this algorithm will call the function "CONFLICT_RESOLUTION(ws_i, a_i)" as an exceptional handler to tackle the issues. The details of these two functions are out of scope of this paper. Hence in order to maximize the level of SoD one needs to assign the set of activities across as many appropriate Web services as possible. In Case III and Case IV, this algorithm is trying to assign the activity to the Web service with the least number of activities already assigned to it, i.e., $MIN(|M^{-1}(ws)|)$. Note that the algorithm does not always generate an optimal solution. However, to generate an optimal solution, *e* out of the *l* links (i.e., "$\rightarrow$") need to be selected in such a way that the level of SoD is maximized, where $e = |C(fm)|$ and $l = |Case\_I| + (\sum_i |ag_i|$ where $\forall_i\ (ws_i, ag_i) \in Case\_II) + (\sum_i |wsg_i|$ where $\forall_i\ (wsg_i, a_i) \in Case\_III) + (\sum_i |wsg_i| \times |ag_i|$ where $\forall_i\ (wsg_i, ag_i) \in Case\_IV)$. An exhaustive search for the optimal assignment would search through $\binom{e}{l}$ possible assignments with the constraints of security assertions at each activity, which, in the worst case, has exponential complexity.

Algorithm WSMP-Algorithm(fm, WS):M;

INPUT:

C(fm) = $\{a_1, a_2, \ldots, a_m\}$; - the group of *m* activities involved in the flow model fm.

WS = $\{ws_1, ws_2, \ldots, ws_n\}$; - the group of *n* Web services.

INITIALIZATION:

Case_I = $CoE_{CaseI}(C(fm), WS)$; - gives a set of tuples (ws_i, a_i).

Case_II = $CoE_{CaseII}(C(fm), WS)$; - gives a set of tuples (ws_i, ag_i).

Case_III = $CoE_{CaseIII}(C(fm), WS)$; - gives a set of tuples (wsg_i, a_i).

Case_IV = $CoE_{CaseIV}(C(fm), WS)$; - gives a set of tuples (wsg_i, ag_i).

FOR i FROM 1 to m DO

$M(a_i)$ = nil;

END;

ALGORITHM:

FOR ALL $(ws_i, a_i) \in$ Case_I DO – First Loop for Case I

IF NOT CONFLICT(ws_i, a_i) THEN;

$M(a_i) = ws_i$;

ELSE

CONFLICT_RESOLUTION(ws_i, a_i);

ENDIF;

ENDFOR;

FOR ALL $(ws_i, ag_i) \in$ Case_II DO – Second Loop for Case II

FOR ALL $a_{ij} \in ag_i$ DO

IF NOT CONFLICT(ws_i, a_{ij}) THEN;

$M(a_{ij}) = ws_i$;

ELSE

CONFLICT_RESOLUTION(ws_i, a_{ij});

ENDIF;

ENDFOR;

ENDFOR;

FOR ALL $(wsg_i, a_i) \in$ Case_III DO – Third Loop for Case III

DO SELECT $ws_{ij} \in wsg_i$ WHERE $\forall_j$ MIN($|M^{-1}(ws_{ij})|$);

- the Web service with the least number of activities already assigned to it;

IF NOT CONFLICT(ws_{ij}, a_i) THEN;

$M(a_i) = ws_{ij}$;

ELSE

CONFLICT_RESOLUTION(ws_{ij}, a_i);

ENDIF;

ENDFOR;

FOR ALL $(wsg_i, ag_i) \in$ Case_IV DO – Forth Loop for Case IV

DO SELECT $a_{ij} \in ag_i$;

DO SELECT $ws_{ij} \in wsg_i$ WHERE $\forall_j$ MIN($|M^{-1}(ws_{ij})|$);

IF NOT CONFLICT(ws_{ij}, a_{ij}) THEN;

$M(a_{ij}) = ws_{ij}$;

ELSE

CONFLICT_RESOLUTION(ws_{ij}, a_{ij});

ENDIF;

UNTIL $ag_i == \varnothing$;

ENDFOR;

OUTPUT:

FOR ALL $a_i \in$ C(fm) OUTPUT $M(a_i) = ws_i$; – the Web service ws_i that is assigned to execute the activity a_i.

Figure 3. A WSMP Algorithm for Maximizing the Level of Separation of Duties.

Using the example in Figure 2, let's assume that there is no CIR occurred in this example, i.e., CONFLICT(ws_i, a_j) = false. Applying the algorithm in Figure 3 generates the following assignments. In the initialization section, all the Web services do not have any activity assigned, i.e., $\forall_{i=1,\ldots,n} |M^{-1}(ws_i)| = 0$. After the first loop for Case I, the results would be $M(a_9) = ws_8$ and $M(a_{10}) = ws_{16}$ with $|M^{-1}(ws_8)| = 1$ and $|M^{-1}(ws_{16})| = 1$. Next, after the second loop for Case II, the results would be $M(a_1) = ws_{22}$, $M(a_2) = ws_{22}$, $M(a_9) = ws_8$ and $M(a_{10}) = ws_{16}$ with $|M^{-1}(ws_{22})| = 2$, $|M^{-1}(ws_8)| = 1$ and $|M^{-1}(ws_{16})| = 1$. Then, after the third loop for Case III, the results would be $M(a_5) = ws_{10}$, $M(a_1) = ws_{22}$, $M(a_2) = ws_{22}$, $M(a_9) = ws_8$ and $M(a_{10}) = ws_{16}$ with $|M^{-1}(ws_{10})| = 1$, $|M^{-1}(ws_{22})| = 2$, $|M^{-1}(ws_8)| = 1$ and $|M^{-1}(ws_{16})| = 1$.

In the forth loop for Case IV, there are three sub-loops in it. In the first sub-loop for the tuple (wsg_1, ag_1), the results would be $M(a_3) = ws_1$, $M(a_7) = ws_2$, $M(a_{11}) = ws_3$, $M(a_5) = ws_{10}$, $M(a_1) = ws_{22}$, $M(a_2) = ws_{22}$, $M(a_9) = ws_8$ and $M(a_{10}) = ws_{16}$ with $|M^{-1}(ws_1)| = 1$, $|M^{-1}(ws_2)| = 1$, $|M^{-1}(ws_3)| = 1$, $|M^{-1}(ws_{10})| = 1$, $|M^{-1}(ws_{22})| = 2$, $|M^{-1}(ws_8)| = 1$ and $|M^{-1}(ws_{16})| = 1$. In the second sub-loop for the tuple (wsg_2, ag_2), the results would be $M(a_4) = ws_{17}$, $M(a_{12}) = ws_{18}$, $M(a_3) = ws_1$, $M(a_7) = ws_2$, $M(a_{11}) = ws_3$, $M(a_5) = ws_{10}$, $M(a_1) = ws_{22}$, $M(a_2) = ws_{22}$, $M(a_9) = ws_8$ and $M(a_{10}) = ws_{16}$ with $|M^{-1}(ws_{17})| = 1$, $|M^{-1}(ws_{18})| = 1$, $|M^{-1}(ws_1)| = 1$, $|M^{-1}(ws_2)| = 1$, $|M^{-1}(ws_3)| = 1$, $|M^{-1}(ws_{10})| = 1$, $|M^{-1}(ws_{22})| = 2$, $|M^{-1}(ws_8)| = 1$ and $|M^{-1}(ws_{16})| = 1$. In the third sub-loop for the tuple (wsg_3, ag_3), the results would be $M(a_6) = ws_9$, $M(a_8) = ws_{11}$, $M(a_4) = ws_{17}$, $M(a_{12}) = ws_{18}$, $M(a_3) = ws_1$,

$M(a_7) = ws_2$, $M(a_{11}) = ws_3$, $M(a_5) = ws_{10}$, $M(a_1) = ws_{22}$, $M(a_2) = ws_{22}$, $M(a_9) = ws_8$ and $M(a_{10}) = ws_{16}$ with $|M^{-1}(ws_9)| = 1$, $|M^{-1}(ws_{11})| = 1$, $|M^{-1}(ws_{17})| = 1$, $|M^{-1}(ws_{18})| = 1$, $|M^{-1}(ws_1)| = 1$, $|M^{-1}(ws_2)| = 1$, $|M^{-1}(ws_3)| = 1$, $|M^{-1}(ws_{10})| = 1$, $|M^{-1}(ws_{22})| = 2$, $|M^{-1}(ws_8)| = 1$ and $|M^{-1}(ws_{16})| = 1$. In conclusion, the level of SoD is $\frac{11}{12}$, where MM(fm) = $\{ws_1, ws_2, ws_3, ws_8, ws_9, ws_{10}, ws_{11}, ws_{16}, ws_{17}, ws_{18}, ws_{22}\}$ and C(fm) = $\{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}, a_{11}, a_{12}\}$. In running the algorithm from Figure 3, all other factors being equal, the service locator in this paper assigned activities to Web services in the order of ws_i first, then ws_j, then ws_k, and so on, where $i < j < k$.

5. Implementation

A Web service security model must support protocol-independent, declarative security policies that the service locator can enforce, and descriptive security policies attached to the service locator that it can use in order to securely assign activities to Web services. A security policy is a set of rules and practices that specify or regulate how a system or organization provides security services to protected resources. A security assertion is typically scrutinized in the context of security policy [6]. In general, the engineering of a security policy starts with risk analysis and ends with a set of security assertions that is ready for integration into the security architecture of a subject such as a service locator. Risk analysis identifies security threats in a business process and forms a set of security assertions, which refer to rules and practices to regulate how sensitive or activity information is managed and protected within a loosely coupled execution environment. A security policy is often formalized or semi-formalized in a security model that provides a basis for a formal analysis of security properties.

Figure 4 demonstrates specifying CIR and SoD assertions into WS-Policy. Beside those standard Web services utility (wsu) and policy (wsp) namespaces defined at http://schemas.xmlsoap.org/ws/2002/07/utility and http://schemas.xmlsoap.org/ws/2002/12/policy [11], this paper assumes that there is a schema for a Web services matchmaking process (wsmp) located at http://schemas.xmlwsmp.org/wsmp/2003/06/matchmaking, as well as a schema for the flow model of the "Patient Health Records Integration Process" stored at the http://schemas.xmlfm.org/fm/2003/06/hdi. Then, the <wsp: Policy/> element (lines 1, 2, 3, 4 and 14) is the top-level container for a policy expression with all the namespaces. In addition, the wsu:id attribute is used to indicate a fragment ID in arbitrary containing elements.

```
(01) <wsp:Policy wsu:Id="Policy1"
xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
(02)        xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy"
(03)
xmlns:wsmp="http://schemas.xmlwsmp.org/wsmp/2003/06/matchmaking"
(04)        xmlns:fm="http://schemas.xmlfm.org/fm/2003/06/hdi" >
(05)    <wsp:All wsp:Usage="wsp:Required" wsp:Preference="100">
(06)        <wsmp:ConflictOfInterest wsmp:GroupID="Group1"
(07)
wsmp:ActivityName="fm:RetrievePatientHealthRecordsatHospital1" />
(08)        <wsmp:ConflictOfInterest wsmp:GroupID="Group1"
(09)
wsmp:ActivityName="fm:RetrievePatientHealthRecordsatHospital2" />
(10)        <wsmp:ConflictOfInterest wsmp:GroupID="Group1"
(11)          wsmp:ActivityName="fm:IntegratePatientHealthRecords" />
(12)        <wsmp:SeparationOfDuties wsmp:Maximize="True" />
(13)    </wsp:All>
(14) </wsp:Policy>
```

Figure 4. An Example of CIR and SoD Assertions in WS-Policy.

The primary operator in this policy expression is <wsp:All/> (lines 5 and 13), which indicates that all of its child assertions are combined to form a policy statement. The <wsp:All/> element indicates that all of the contained security assertions must be met. The wsp:Usage attribute is used to qualify the semantics of the leaf elements as applied to a policy subject such as the service locator. In this example, the value wsp:Required means that the assertion must be applied to the service locator. If the service locator does not meet the criteria expressed in the assertion, a fault or error will occur. The wsp:preference attribute is to specify the preference of this policy statement. The higher the value of the preference is, and the greater the weighting of the expressed preference is.

This paper proposes the security assertions for specifying conflict of interest in the leaf elements (lines 6, 7, 8, 9, 10 and 11). The security assertions <wsmp:ConflictOfInterest/> are used to specify those three conflicting activities in the *Group1*:

- wsmp:ActivityName="fm:RetrievePatientHealthRecordsatHospital1" (lines 6 and 7)
- wsmp:ActivityName="fm:RetrievePatientHealthRecordsatHospital2" (lines 8 and 9)
- wsmp:ActivityName="fm:IntegratePatientHealthRecords" (lines 10 and 11)

The security assertion <wsmp:SeparationOfDuties/> (line 12) is used to indicate whether to maximize the level of SoD or not (e.g., wsmp:Maximize="True" or "False").

```
(01) <wsp:PolicyAttachment>

(02)   <wsp:AppliesTo>

(03)     <wsp:EndpointReference xmlns:services="
http://www.xmlservices123.com">

(04)       <wsp:ServiceName Name="services:LocatorService" />

(05)       <wsp:PortType Name="services:LocatorPortType" />

(06)       <wsp:Address
URI="http://www.xmlservices123.com/servicelocator" />

(07)     </wsp:EndpointReference>

(08)   </wsp:AppliesTo>

(09)   <wsp:PolicyReference
wsp:URI="http://www.xmlpolicies123.com/wsmp#Policy1" />

(10) </wsp:PolicyAttachment>
```

Figure 5. An Example of Related WS-Policy Attachment to Figure 4.

This paper assumes that the policy expression is stored at http://www.xmlpolicies123.com/wsmp#Policy1. The mechanism for associating policy with one or more subjects (e.g., service locators) is referred to as policy attachment. Figure 5 now demonstrates how to attach the policy to the service locator. Figure 5 shows a policy expression to be associated with a resource independent of its definition and representation, using a <wsp:PolicyAttachment/> element (lines 1 and 10). The <wsp:AppliesTo/> element (lines 2 and 8) defines one domain expression by the <wsp:EndpointReference/> (lines 3 and 7) element. In this example, this paper assumes that the service locator is located at http://www.xmlservices123.com. Then, the child assertions contain three elements:

- <wsp:ServiceName/> (line 4) indicates the name of the service locator.
- <wsp:PortType/> (line 5) indicates the type of the service locator.
- <wsp:Address/> (line 6) indicates the Uniform Resource Identifier (URI) of the service locator.

The <wsp:PolicyReference/> element (line 9) references the policy expression that is being applied to the service locator. The wsp:URI attribute references a policy using its URI.

Figure 6 presents a technical framework for supporting the matchmaking process in a loosely coupled Web services execution environment. The Web service "WS-Policy and WS-PolicyAttachment Editor" is used to specify related XML documents for the matchmaking process. The WS-Policy document is submitted to the Web service "WS-Policy Parser" and then the WS-PolicyAttachment document is used to bind the WS-Policy document to a particular entity, i.e., the service locator. The Web service "WS-Policy Parser" classifies different policy assertions from the WS-Policy document into different groups and then distributes each group to the relevant Web services such as the Web service "Conflict of Interest Service (CIRService)" and "Separation of Duties Service (SoDService)." On the other side, the Web service "Service Locator" interacts with the Web services "Flow Model Generator" (e.g., BPEL4WS) and "Service Registry" (e.g., UDDI) for matching each activity from the flow model to an appropriate Web service. During the matchmaking process, the "Service Locator" has to consult those services such as the "CIRService" and "SoDService." As a result, the "Service Locator" returns an appropriate matchmaking pattern for each flow model. A matchmaking pattern is defined as a set of assignments between activities and Web services. Note that all the interactions between Web services are driven by SOAP messages. The first version of a prototype Web service "CIRService" has been implemented and the 'SoDService" is currently under development by using C# on .NET framework.

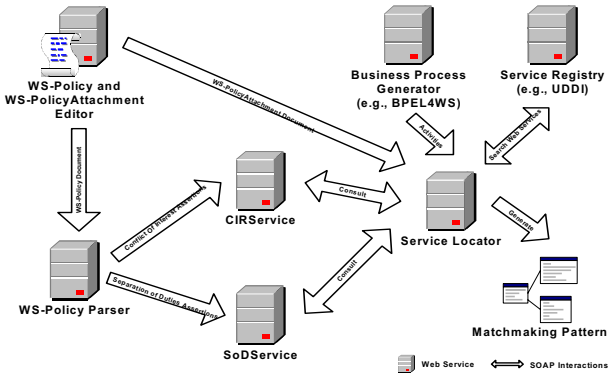

Figure 6. A Technical Framework for WSMP.

6. Conclusions and Future Research

In comparison to traditional business-to-business applications that connect trading partners through a centralised architecture, Web services have the advantages of faster time to production, convergence of disparate business functionalities, a significant reduction in total cost of development and easy to deploy business applications for trading partners. Despite the numerous benefits attainable after adopting Web services, a major concern is the lack of security conventions after changing from a secure environment to an open and exposed

environment. Many security problems do arise in well-controlled environments from authorized parties. This means that security is achieved not only by ensuring that Web services are trusted but also by the clearance of other security threats such as conflict of interest and separation of duties. Therefore, this paper discussed the concept of conflict of interest and separation of duties in the Web services matchmaking process. Further, based on the WS-Policy, this paper demonstrated how to specify security assertions for preventing conflict of interest and maximizing separation of duties in the Web services matchmaking process. Furthermore, we are currently investigating the cost factor and coordination efforts when executing the WSMP algorithm in the context of dynamic composite Web services.

7. References

[1] David F. C. Brewer, and Michael J. Nash, "Chinese Wall Security Policy," Proceedings of the Symposium on Security and Privacy, 1989, pp. 206-214.

[2] Contentguard, "eXtensible rights Markup Language (XrML)," Version 2.0, 2001.

[3] J. Fontana, "Top Web Services Worry: Security," NetworkWorldFusion, January 2002.

[4] B. Hofreiter, C. Huemer, and W. Klas, "ebXML: Status, Research Issues, and Obstacles," Proceedings of Twelfth International Workshop on Research Issues in Data Engineering: Engineering E-Commerce/E-Business Systems, 2002, pp. 7-16.

[5] P. Holland, "Building Web Services From Existing Application," eAI Journal, September 2002, pp. 45-47.

[6] Maryann Hondo, Nataraj Nagaratnam, and Anthony Nadalin, "Securing Web Services," IBM Systems Journal, vol. 41, no. 2, 2002, pp. 228-241.

[7] Patrick C. K. Hung, "Specifying Conflict of Interest in Web Services Endpoint Language (WSEL)," ACM SIGecom Exchanges, vol. 3.3, 2002, pp. 1-8.

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

[9] IBM Corporation, "Web Services Transaction (WS-Transaction)," 2002.

[10] IBM Corporation, "Web Services Coordination (WS-Coordination)," 2002.

[11] IBM Corporation, "Security in a Web Services World: A Proposed Architecture and Roadmap," White Paper, Version 1.0, 2002.

[12] F. Leymann, "Web Services Flow Language (WSFL 1.0)," IBM Corporation, 2001.

[13] Netegrity, "JSAML Toolkit: Netegrity's Java Implementation of the Security Assertions Markup Language (SAML) Specification," White Paper, 2001.

[14] OASIS, "SAML 1.0 Specification Set: Committee Specifications," 2002.

[15] OASIS, "OASIS eXtensible Access Control Markup Language (XACML)," OASIS Standard 1.0, 2002.

[16] P. Ratnasingam, "The Importance of Technology Trust in Web Services Security," Information Management & Computer Security, vol. 10, no. 5, 2002, pp. 255-260.

[17] Sun Microsystems, "Web Service Choreography Interface (WSCI)," Version 1.0, 2002.

[18] S. Thatte, "XLANG - Web Services for Business Process Design," Microsoft Corporation, 2001.

[19] UDDI Organization, "UDDI Specification," Version 3.0, Published Specification, 2002.

[20] World Wide Web Consortium (W3C). Online: www.w3c.org

[21] Matunda Nyanchama and Sylvia Osborn, "The Role Graph Model and Conflict of Interest," ACM Transactions on Information and System Security, vol. 2, no. 1, 1999, pp. 3-33.

[22] Gail-Joon Ahn and Ravi Sandhu, "The RSL99 Language for Role-based Separation of Duty Constraints," Proceedings of the ACM Workshop on Role-Based Access Control, 1999, pp. 43-54.

[23] Richard T. Simon and Mary Ellen Zurko, "Separation of Duty in Role-based Environments," Proceedings of the 10th Computer Security Foundations Workshop, 1997, pp. 183-194.

[24] Michael J. Nash and Keith R. Poland, "Some Conundrums Concerning Separation of Duty," Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy, 1990, pp. 201-207.

[25] D. Richard Kuhn, "Mutual Exclusion of Roles as a Means of Implementing Separation of Duty in Role-based Access Control Systems," Proceedings of the Second ACM Workshop on Role-based Access Control, 1997, pp. 23-30.

[26] Virgil D. Gligor, Serban I. Gavrila and David Ferraiolo, "On the Formal Definition of Separation-of-duty Policies and Their Composition," Proceedings of 1998 IEEE Symposium on Security and Privacy, 1998, pp. 172 -183.

[27] Patrick C. K. Hung, Kamalakar Karlapalem and James Gray III, "Least Privilege Security in CapBasED-AMS," The International Journal of Cooperative Information Systems, vol. 8, no. 2 & 3, 1999, pp. 139-168.

[28] David D. Clark and David R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," Proceedings of 1987 IEEE Symposium on Security and Privacy, 1987, pp. 184-194.

[29] Liang-Jie Zhang and Tian Chao, "Business Explorer for Web Services (BE4WS)," IBM AlphaWorks, 2001. Online: www.alphaworks.ibm.com/tech/be4ws