Embedding Access Control Policy in Web Service Path Composition Algorithm

Shih-Chien Chou* and Jin-Yuan Jhu
Department of Computer Science and Information Engineering
National Dong Hwa University, Taiwan
E-Mail: scchou@mail.ndhu.edu.tw

Abstract
Web services can accomplish requirements, which are generally complicated functions. To apply web services for a requirement, the requirement should be decomposed into sub-functions in which each sub-function can be accomplished by a web service. After the decomposition, web services are selected to compose paths for the requirement. During composition, secure access of web services should be considered. This paper proposes a two-leveled web service access control policy and a web service composition algorithm. We embed the policy in the algorithm. During selection, the upper level access control policy uses attributes and credentials to filter out the web services that cannot be invoked by a requester. The lower level policy then compares the credit level numbers of web services with the security level numbers of arguments. It then evaluates the possibility of leaking the arguments by a web service. The possibility facilitates evaluating the successfulness of executing a path. After the two-leveled access control, the composition algorithm composes multiple paths. The requester can select more than one path. One of the selected paths can be executed and the others are spare ones to reduce path replanning effort through replacement.

Keywords: Web service, access control, access control policy, web service composition

*Corresponding author
1. INTRODUCTION

The application of web services becomes wider. To facilitate using web services, service providers use a standard such as WSDL (web service description language) [34] to describe web services and then register the web services to a broker such as UDDI (universal description, discovery, and integration protocol) [24]. When a requester requires service, the broker facilitates discovering appropriate web services.

Web services offer functions to serve requesters. Requesters generally request web services to accomplish their requirements. A requirement can usually be solved by multiple solutions. For example, the requirement “Get a house to live in” can be solved by the solutions: “Buy a house” and “Rent a house”. To accomplish a requirement, the requester selects one or more feasible solutions. He then uses web services to accomplish the solutions. Since a requirement’s solutions are generally complicated functions, invoking a single web service to accomplish a solution is difficult. In this case, a solution should be decomposed into sub-functions, in which each sub-function can be accomplished by a web service. After that, selection [3, 12, 18] is applied to identify web services for each sub-function and composition [9, 13-14, 19] is applied to compose a web service path. During selection and composition, the quality of service (QoS) criteria should be fulfilled [1, 20, 35]. After composition, the composed path is executed. If problems occur during path execution, path replanning techniques can heal the path [4, 8, 26]. The path will eventually finish, which implies that the solution have been accomplished.

The above description reveals that the use of web services induces problems such as composition, QoS, and replanning. As the application of web services become wider and more popular, web service providers around the world are expected to be more. We believe that most providers are honest. However, there may be dishonest ones. Dishonest providers may provide web services to cheat money or even steal sensitive information, such as credit card information. Since requesters cannot distinguish honest providers from dishonest ones, requesters may invoke web service provided by dishonest providers. Some may think that requesters can invoke web services from providers they trust. Nevertheless, this type of invocation limits the use of web services because there may be web services not provided by the trustable providers but by others. To widen the application of web services, requesters should better not limit providers. In this regard, applying web services to accomplish solutions takes risks and security issues such as access control should be taken into consideration.

In the past years, our research focused on secure web service access. Existing web service access control policies (see section 2) generally use mechanisms such as attributes and credentials to decide whether a requester can invoke a web service. That
is, the policies protect web services. In our opinion, requesters should also be protected. For example, if a requester sends his credit card information to a dishonest web service for a payment, the credit card information may be stolen. Accordingly, our previous work protects requesters’ sensitive information using a trust management mechanism [10]. The mechanism used the security level numbers of arguments and the credit level numbers of web services for the protection. When a requester intends to invoke a web service, the credit level number of the web service should be at least the same as the maximum security level numbers of the arguments (i.e., the web service should be trusted by the requester). This protects the arguments sent to the web service, which implies that the requester is protected.

In addition to web service access control, we also eyed on web service path composition. We identified that existing composition algorithms normally take quality of service (QoS) criteria into consideration but fail to control secure access of web service. We think that composing a path by ignoring secure web service access may result in execution failure or even information leakage (e.g., leaking sensitive arguments). Perhaps replanning can heal an ill path. However, path replanning is not cheap and should be avoided if possible. We also noticed that existing composition algorithms generally compose only one “optimal” path. However, if web service(s) are selected from dishonest providers, the most optimal path may be not optimal. Moreover, execution failure may occur in the optimal path, which causes the expensive replanning process to take action. Path replanning may be unavoidable. However, we must reduce its frequency and low down its effort. Our experience showed that embedding access control policy in a path composition algorithm can reduce the frequency of execution failure. We also identified that if a path composition algorithm compose more than one path, the effort of path replanning can be reduced. The rationale is that the web services and sub-paths of the paths not under execution (we call them spare paths) can replace the failed web services or sub-paths of the executing path. Although we identified researches use web service replacement to heal an ill path [11, 15, 32, 37], they failed to consider the possibility of replacing a sub-path or even the entire path. Replacing the entire path may be necessary when execution failure occurs on the first web service of a path.

According to the above description, we designed a path composition algorithm that embeds a web service access control policy. In addition, the algorithm composes multiple paths to reduce path replanning effort. The access control policy is composed of two levels. The upper level policy protects web services using attributes and credentials to filter out web services that cannot be invoked by a requester. The lower level policy is an adaptation of the trust management mechanism of our previous work [10]. As described above, the mechanism compares the credit level numbers of
web services with the security level numbers of arguments to decide whether the web services are trustable. That is, the comparison gets a definite answer. In the mechanism, the credit level number of a web service will be increased every period of time if it does not leak information. On the other hand, the number will be decreased whenever it leaks information. Since a web service may be invoked by various requesters around the world, its credit level number may be changed from time to time. Since the time point of composing a path and that of executing the path should be different because of composition time, the credit level number of a web service during path composition may be different from the number during path execution. Accordingly, the lower level access control policy should not offer a definite answer.

Our lower level access control policy thus adapts the original trust management mechanism using a possibility to replace the definite answer. It is reasonable for this adaptation because a web service fails to pass the trust mechanism during path composition may pass the mechanism during path execution, and vice versa.

After the two-leveled access control, our composition algorithm composes multiple paths that fulfill the QoS criteria. This paper proposes the access control policy embedded path composition algorithm. Note that we do not offer a web service selection algorithm but suppose it is available whenever needed. This will not affect the contribution of this paper because selection is not the objective of this paper.

2. RELATED WORK

This paper proposes an access control policy embedded path composition algorithm. Below we survey the researches of web service access control and path composition.

2.1 Related Work on Web Service Access Control

Web service access control determines whether a requester can invoke a web service. It is a topic of web service security. We identify that the standards WS-Security [23] and XACML (extensible access control markup language) [22] are related to web service security. WS-Security controls the security of information transferring on the network. Therefore, it can be considered a standard related to cryptography. Since our research focuses on web service access control, WS-Security is out of the scope of our research. XACML [22] is a standard proposed by OASIS for web service access control. It offers mechanisms to describe access control policies for web services. In addition to the standards, we identified quite a few web service access control policies. We discuss some of them below.

The model in [29] is an implementation of XACML. In the implementation, requester uses PMI (privilege management infrastructure) for the checking, retrieval, and revocation of authentication. The model also determines whether a requester can
invoke a web service using an RBAC-based access control policy (RBAC is the abbreviation for “role-base access control” [27]). The model in [33] defines its access control policies using the RBAC concept. It is a two-leveled mechanism to control web service access. The first level is the service level. It checks the roles assigned to both requesters and web services. A requester passing the first level of access control should also pass the second level of the control, which is the attribute level. The level uses parameters as attributes of a service and assigns permissions to the attributes. A requester passing the first level of control can invoke a web service only when it possesses the permissions to access the attributes.

The model in [6] controls web service access using X-GTRBAC (XML-based generalized temporal role-based access control) [7], which can be used in heterogeneous and distributed sites. It also applies TRBAC (temporal role-based access control) [5] to control the factor of time. The model in [28] offers a language to facilitate enforcing the access control specification (or access control policy). The access control specification describes whether a composition of web services can be invoked. The model in [31] uses the language “pure-past linear temporal logic” (PPLTL) to determine whether a composition of web service can be invoked. The model considers history information in addition to the current status. Our previous work [10] proposes a trust management mechanism for web service access control. It determines whether a requester can invoke a web service using the credit level numbers of web services and the security level numbers of arguments. Under the control of the mechanism, a requester can invoke a web service only when the credit level number of the web service is at least the same as the maximum security level numbers of the arguments.

In addition access control policies, quite a few researches discuss negotiation among requesters and web services. Negotiation is also a type of access control. Below we survey some of them.

The model Trust-Serv [30] dynamically chooses web services at run time. The choosing is based on trust negotiation [36], which selects web services that can be invoked using credentials. State machines are used in Trust-Serv to describe the trust negotiation policies. The model in [16] defines strategies for negotiation policies using credentials. The model in [21] manages k-leveled of web service invocations using credentials. A k-leveled invocation is needed because a web service may invoke others, which results in multilevel invocations.

According to our survey, most existing access control policies protect web services only. This paper proposes a policy to protect both requesters and web services. In addition, we embed the policy in a web service composition algorithm for composing multiple paths.
2.2 Related Work on Web Service Path Composition

The model in [19] proposes the Quality and Relation Driven Services Composition (QRDSC) approach to compose web service paths. During composition, the relationships among services such as the pre- and post-conditions of web services are considered in addition to the QoS constraints.

The model in [14] proposes an approach that takes the tightness of QoS criteria into consideration during web service composition. The tightness of a QoS criterion can be adjusted by a requester according to his needs. The adjustment can improve the flexibility of the composition.

The model in [13] propose an approach to dynamically compose web services (i.e., web service composition is operating during the execution of other web services, which is different from the traditional static composition). The approach is useful in an error-prone environment to improve the reliability of the composed path. The approach proposes a mechanism named aggregated reliability (AR) to improve the reliability of web service selection and composition.

The model in [9] proposes an approach for dynamically composition. It selects multiple versions of a web service and uses a mechanism to monitor the execution of a web service path. When the mechanism identifies that a web service on the path is unavailable, it will dynamically selects another version of the web service to replace the unavailable one. This approach can improve web service path reliability.

There are many other web service composition approaches not mentioned here. What we intend to emphasize are: (a) the composition approaches generally select one optimal path, which may result in more frequent path replamping and (b) the composition approaches generally ignore web service access control. As described in section 1, access control is crucial when using web services. Therefore, a composition approach should not skip web service access control. According to the considerations, we propose a web service composition algorithm that embeds an access control policy. Moreover, the algorithm composes multiple paths.

3. THE APPROACH

This section proposes our access control policy and the path composition algorithm that embeds the policy.

3.1 Access Control Policy

We propose a two-leveled web service access control policy. The upper level policy uses attributes and credentials to decide whether a requester can invoke a web service. The checking of credentials is easy. If a requester possesses the credentials required
by a web service and he knows the passwords of the credentials, the checking passes. The checking of attributes is also easy. We use \textit{enumeration} and \textit{range} to define the attributes acceptable by a web service. For example, the attribute requirement “Title $\in \{\text{manager, vice department char, department chair, vice president, president}\}$, Age $\geq 20$, Sex $\in \{\text{man}\}$” states that a requester can invoke a web service if the following attribute conditions are all fulfilled.

a. The requester’s title should be within the following set: \{\text{manager, vice department char, department chair, vice president, president}\}. This attribute requirement is an enumeration.
b. The requester’s age should be at least 20. This requirement is a range.
c. The requester should be a man. This requirement is again an enumeration.

If both credentials and attributes are required by a web service, the requester should fulfill both of them. When a requester passes the upper level access control policy, he should also pass the lower one before invoking the web service. The lower level policy is adapted from the trust management mechanism of our previous work [10]. Below we describe the trust management mechanism and then the lower level access control policy proposed in this paper.

The original trust management mechanism uses two major components. They are respectively the security level numbers of arguments and the credit level numbers of web services, as described below.

a. \textit{Security level numbers}. When a requester intended to invokes a web service, every argument passed to the web service is associated with a security level number. A larger number implies a more sensitive argument. The number is between 0 and a maximum value determined by the user.
b. \textit{Credit level numbers}. Every web service is associated with a credit level number. A larger number implies a more trustable web service. The number is between 0 and a maximum value determined by the user.

When a requester intends to invoke a web service, he sends a set of arguments to the service, in which every argument is associated with a security level number. Suppose: (a) the arguments constitute the set $ARG$ defined as \{\text{ARG}_i \mid \text{ARG}_i$ is an argument\}, (b) the web service to be invoked is $WEBSER$, (c) $ARG_{in}$ is the security level number of $ARG_i$, and (d) $WEBSER_{in}$ is the credit level number of $WEBSER$. 
With the assumptions, if $WEBSER_{c_{ln}} \geq \text{Max}(ARG_{s_{ln}})$, the mechanism considers the web service trustable and allows the invocation.

In the trust management mechanism, the security level number of an argument is set by the requester to reflect the argument’s sensitivity. The number will not be changed during web service execution. On the other hand, the credit level number of a web service determines whether the web service is trustable. If a web service does not leak sensitive information for a period of time, its credit level number will be increased. On the other hand, if the web service leaks sensitive information, its credit level number will be decreased. The following example explains information leakage.

Suppose a web service displays credit card information on a screen and the information is captured by a malicious person, the web service leaks information. Detecting information leakage during web service execution can be achieved by an information flow control model. We will describe our leakage detection model in section 3.2.

As mentioned before, the lower level access control policy is adapted from the trust management mechanism. In the original trust management mechanism, a requester is allowed to invoke a web service only when $WEBSER_{c_{ln}} \geq \text{Max}(ARG_{s_{ln}})$.

However, since the credit level numbers of web services may change from time to time, it is possible that the condition is true during web service composition but false during path execution, and vice versa (note that requesters around the world may invoke web services, which causes decrement or increment of the web services’ credit level numbers). Accordingly, our lower level access control policy uses a possibility instead of a definite answer to depict whether a requester can invoke a web service. To obtain the possibility we suppose that a requester intends to pass the argument set $ARG$ to the web service $WEBSER$. We also make the following assumptions:

a. $ARG = \{ARG_i | ARG_i$ is an argument\}.

b. $ARG_{s_{ln}}$ is the security level number of $ARG_i$.

c. $WEBSER_{c_{ln}}$ is the current credit level number of $WEBSER$.

d. $n$ is a threshold to determine the possibility. It is defined by the user according to his experiences.

e. $k = WEBSER_{c_{ln}} - \text{Max}(ARG_{s_{ln}})$. There is no special semantics for $k$. We use it to simplify the following discussion.
f. $\text{POSI}_{\text{succ}}$ is the possibility that the requester can successfully invoke $\text{WEBSER}$ during path execution.

According to the assumptions above, we use Figure 1 to explain the obtaining of the possibility $\text{POSI}_{\text{succ}}$. If $k$ is smaller than $-n$, we think that the credit level number of $\text{WEBSER}$ is too small and $\text{POSI}_{\text{succ}}$ is considered 0 (remember that the threshold value $n$ is defined by the user according to his experiences). On the other hand, if $k$ is larger than $n$, we think that the credit level number of $\text{WEBSER}$ is large enough and $\text{POSI}_{\text{succ}}$ is considered 1. In other cases, $k$ is between $-n$ and $n$ and $\text{POSI}_{\text{succ}}$ will get a value between 0 and 1. To evaluate $\text{POSI}_{\text{succ}}$, we first adjust the evaluation base $-n$ to be 0. According to the adjustment, the $k$ value will be adjusted to be $k+n$. Since the range between $-n$ and $n$ is $2n$, $\text{POSI}_{\text{succ}}$ is the value $(k+n)/2n$. $\text{POSI}_{\text{succ}}$ can thus be evaluated using Formula 1 below.

$$
\text{POSI}_{\text{succ}} = \begin{cases} 
1, & \text{if } k > n \\
(k+n)/2n, & \text{if } k \text{ is between } -n \text{ and } n \\
0, & \text{if } k < -n
\end{cases} \quad (1)
$$

Figure 1. Evaluating the access possibility

In the adaptation, managing the credit levels of web services is a problem. There are two approaches for the management. The first one is storing a web service together with its credit level number. In this approach, every requester can change the number. A serious drawback associated with this approach is that dishonest requesters may incorrectly change the number according to malicious purposes. The second approach is saving the number in an organization trusted by every requester and service provider. For example, if the trust management mechanism concept is included in a standard, a trustable organization such as IBM or Microsoft can manage
the credit level numbers. In this approach, if a requester identifies that a web service leaks information, the requester notifies the organization to decrease the credit level number of the web service. The organization then checks whether the requester is trustable. If the answer is positive, the number is decreased as requested. Otherwise, the number is unchanged. If the credit level of a web service is not decreased for a period of time, the number is increased by the organization.

In the network, requesters and providers are around the world and generally independent. They need to be regulated by standards and the standards should be managed by trustable organizations. For example, both IBM and Microsoft provided a UDDI as the broker of web services (note that the two UDDIs have been closed for some reasons years ago [17]). We thus think that the above mentioned second approach is more acceptable.

According to the description above, the upper level of our web service access control policy is a hard one. It offers either a positive or negative answer because the attributes of requesters and the credentials owned by requesters changed rarely. On the other hand, the lower level policy is a soft one, which offers a possibility value. As a summary, when a requester intends to invoke a web service, he should first pass the upper level policy. After that, he will obtain a number indicating the possibility of successfully invoking the web service during path execution.

3.2 Information Leakage Detection
Detecting information leakage is crucial because the leakage will change credit level numbers of web services. To detect the leakage, an information follow control model can be used. We use our previous model [10] for the detection. The model is an extension of XACML [22] and is named EXACML (extended XACML). Below we describe the kernel of EXACML, in which the concept of RBAC is embedded.

\[ \text{EXACML} = (\text{USR, RLE, URA, VAR, ACLS, DSOURCEs}) \], in which

a. \text{USR} is a set of users. Users play roles.
b. \text{RLE} is a set of roles.
c. \text{URA} is a set of user-role assignments, which is defined as \( \text{USR} \rightarrow 2^{\text{RLE}} \).
d. \text{VAR} is the set of variables.
e. \text{ACLS} is a set of access control lists (ACLs). EXACML attaches an ACL to every variable. ACLs are permissions in EXACML. EXACML uses object methods to define ACLs. Letting \text{MD} be the set of object methods, the ACL \( ACL_{\text{var}} \) attached to a variable \( \text{var} \) is defined as \( ACL_{\text{var}} = \{RACL_{\text{var}}, WACL_{\text{var}}\} \), in which
1. $RACL_{var} \in 2^{MD}$. Object methods in $RACL_{var}$ are allowed to read $var$. RACL is for read access control.

2. $WACL_{var} \in 2^{MD}$. Object methods in $WACL_{var}$ are allowed to write $var$. WACL is for write access control.

f. $DSOURCES$ is a set of data sources (DSOURCEs). Each variable is associated with a DSOURCE to facilitate write access control. DSOURCE records the object methods by which a variable’s data are written.

According to the above definition, EXACML uses the following rules to prevent information leakage during web service execution, in which the first rule ensures the security for read access and the second for write access.

**First EXACML secure rule:**

$$RACL_{d\_var} \subseteq \bigcap_{i=1}^{n} RACL_{var_i} \land (md1 \in \bigcap_{i=1}^{n} RACL_{var_i})$$

**Second EXACML secure rule:**

$$WACL_{d\_var} \supseteq \bigcup_{i=1}^{n} DSOURCE_{var_i} \cup \{md1\}$$

To define the rules, we made the following assumption: (a) the variable $d\_var$ is assigned a value derived from the variables in the set $\{var_i \mid var_i \in VAR\}$ in the definition of EXACML and $i$ is between 1 and $n$, (b) the assignment appears in the object method $md1$, (c) the original ACL of $d\_var$ is $\{RACL_{d\_var}; WACL_{d\_var}\}$, (d) the ACL of the $i^{th}$ variable that derives $d\_var$ is $\{RACL_{var_i}; WACL_{var_i}\}$, and (e) the DSOURCE of $var_i$ is $DSOURCE_{var_i}$. In the first rule, the condition $RACL_{d\_var} \subseteq \bigcap_{i=1}^{n} RACL_{var_i}$ requires that $d\_var$ must be at least the same sensitive as the variables deriving $d\_var$. The condition $md1 \in \bigcap_{i=1}^{n} RACL_{var_i}$ is necessary because the method $md1$ reads the variables deriving $d\_var$ and assigns the derived value to $d\_var$. The second rule requires that the data sources of the variables deriving $d\_var$ should be within $WACL_{d\_var}$, because the data derived from the variables are written into $d\_var$. The rule also requires that the method $md1$ must be within $WACL_{d\_var}$ because the write operation is performed by the method.

If both the secure rules are satisfied, the derived data is assigned to the variable $d\_var$. After that, the ACL of $d\_var$ should be changed by the join operation to prevent indirect information leakage. We use the symbol $\oplus$ to represent the join operator. With join, $ACL_{d\_var}$ will be changed to $\bigoplus_{i=1}^{n} ACL_{var_i}$ after the derived data is assigned to $d\_var$. The join operation for the variables in the set $\{var_i \mid var_i$ is a
variable} is defined below:

$$\bigoplus_{i=1}^n ACL_{var} = (\bigcap_{i=1}^n RACL_{var} \cap \bigcup_{i=1}^n WACL_{var})$$

The join operation trusts less or the same set of readers. Therefore, join will not lower down security level. On the other hand, the operation trusts more writers. This is reasonable because a writer that can write a variable should be regarded as a trusted data source for the data derived from the variable. In addition to joining ACLs, the DSOURCE of \(d_{\text{var}}\) will be adjusted as follows:

$$DSOURCE_{d_{\text{var}}} = \bigcup_{i=1}^n DSOURCE_{var} \cup \{md1\}$$

\(DSOURCE_{d_{\text{var}}}\) is set the union of the DSOURCEs of the variables deriving \(d_{\text{var}}\) and the method set \(\{md1\}\). The union of the DSOURCEs is obvious because all data sources deriving the computation result should be considered data sources of the result. The method \(md1\) is also a data source because the computation result is written by \(md1\) to \(d_{\text{var}}\).

EXACML should be embedded in web services to monitor their execution. During web service execution, any violation of the EXACML secure rules results in information leakage. EXACML will detect the violation and reports it to the requester. The requester then notifies the trustable organization to decrease the credit level number of the web service. The organization then checks whether the requester is trustable. If the answer is positive, the number is decreased as requested. Otherwise, the number is unchanged.

### 3.3 Access Control Policy Embedded Composition Algorithm

As mentioned before, a requirement may have multiple solutions. A requester may select one or more feasible solutions for the requirement. Since every solution is a complicated function, a solution should be decomposed into sub-functions in which every sub-function can be accomplished by a web service. For example, Figure 2 depicts the decomposition of the solution “Rent a house” for the requirement “Get a house to live in”. To simplify the following discussion, we number every sub-function in the figure. In general, more than one path can be applied to accomplish a solution. For example, the path consisting of sub-functions 1, 2, 5, 6, 7, and 8 can accomplish the solution “Rent a house”. Below we give a definition \(Dsol\) for the decomposition of a solution (note that Figure 2 is a \(Dsol\)).
Definition 1. $D_{sol} = (SF, AR, REL)$, in which
   a. $SF = \{sf_i | sf_i$ is a sub-function that can be accomplished by a web service$\}$.
   b. $AR = \{<sf_i, sf_j> | <sf_i, sf_j>$ is an arrow from $sf_i$ to $sf_j$, and $\{sf_i, sf_j\} \subseteq SF\}$.
      The existing of $<sf_i, sf_j>$ means that the sub-function $sf_j$ can be executed after $sf_i$ finishes.
   c. $REL = \{SEQ, AND-split, AND-join, OR\}$. That is, $REL$ is a set containing relationships to construct a workflow, in which: (a) $SEQ$ defines the execution sequence of sub-functions (e.g., sub-function 2 can be executed after sub-function 1 finishes in Figure 2), (b) AND-split and AND-join respectively define the splitting and joining points for parallel sub-functions (e.g., sub-functions 4 and 8 in Figure 2 are respectively an AND-split and an AND-join), and (c) $OR$ define alternatives of sub-functions (e.g., any of sub-functions 2 and 3 in Figure 2 can be selected to execute).

   In $REL$, we do not include the important relationship repetition (i.e., loop). The rationale is that our composition algorithm transfers a graph into a tree. If a graph contains a loop, the transferring cannot be achieved. The research in [25] stated that finding an optimal web service path from a graph with loops is a NP-hard problem. The paper [2] suggests to identify a sub-optimal solution to avoid the NP-hard problem. And, the paper [38] proposes a solution using logs (i.e., historical experiences) to offer a sub-optimal solution. In this paper, we temporarily ignore loops. One member in our laboratory is finding a sub-optimal solution for a graph containing loops. Therefore, solving the problem is a future work.
After decomposing a solution, web services should be selected for the sub-functions in the solution, during which the QoS criteria should be fulfilled. Since the primary objective of this paper is not designing a QoS model, we only use the following four QoS criteria: cost, execution time, reliability, and availability. In general, some criteria values should be small, such as cost. On the other hand, some criteria values should be large, such as reliability. To define a QoS model, we have to consistently consider what QoS value is “good”. In this paper, we consider a large QoS value as good. For a QoS criterion whose value should be small, the requester should set a maximum value for the criterion to minus the criterion’s value. The difference should be large for the QoS criterion. For example, suppose the maximum cost is USD 10. Also suppose that web services 1 and 2 respectively cost USD 6 and 7. When using USD 10 to minus the costs, the differences are USD 4 and 3, respectively. Web service 1 is thus considered better than web service 2. With the management, we can consistently consider a large QoS value as “good”.

To check QoS, the operation $WSQoSOp$ (web service QoS managing operation) below should first be applied. Then, the selected web services should pass the following QoS checking rule $WSQoSRule$ (web service QoS checking rule).

**WSQoSOp:** \( \forall WSQoS_{Ci}, \text{if } WSQoS_{Ci} \text{ should be small, then } WSQoS_{cv_i} = WSQoS_{max_i} - WSQoS_{cv_i} \), in which

- \( WSQoS_{Ci} \) is the i\(^{th} \) web service QoS criterion.
- \( WSQoS_{cv_i} \) is the value of the i\(^{th} \) web service QoS criterion.
- \( WSQoS_{max_i} \) is the maximum value of the i\(^{th} \) web service QoS criterion.

**WSQoSRule:** \( \forall WSQoS_{cv_i}, WSQoS_{cv_i} \geq WSQoS_{cl_i} \), in which

- \( WSQoS_{cl_i} \) is the minimum value for the i\(^{th} \) web service QoS criterion.

Selecting web services for sub-functions in a solution results in a web service invocation graph (WSIG). Figure 3 depicts a WSIG for the solution in Figure 2. Circles in Figure 3 are web services. It is natural that every web service in the graph passes $WSQoSRule$. To simplify the following discussion, we number every web service in the figure. Moreover, we do not duplicate a web service’s name if its name is the same as that of another. Below we give a definition for a WSIG.
**Definition 2.** $WSIG = (\{WS\}, AR, REL, \{LWS\})$, in which

a. $WS = \{ws_i \mid ws_i$ is a web service that can accomplish $sf_k$, in which $sf_k \in SF$ in Definition 1}. In other words, $WS$ is a web service set whose components can only accomplish a single sub-function in a $Dsol$. For example, the web service set $\{1.1, 1.2, \ldots, 1.i\}$ in Figure 3 is a $WS$.

b. $\{WS\} = \{WS_i \mid WS_i$ is a $WS$ and the union of $WS_i$ constitutes the web services in $WSIG\}$.

c. $AR$ is the set of arrows, whose definition is similar to that in Definition 1. If an arrow exists between sub-functions $i$ and $j$ in a $Dsol$, an arrow exists between every component of $WS_i$ and that of $WS_j$. Please compare Figures 2 and 3 to confirm this.

d. $REL$ is the same as that in Definition 1. If a relationship covers an arrow of a sub-function in a $Dsol$, the relationship covers the corresponding arrow of the web services that accomplish the sub-function. Please compare Figures 2 and 3 to confirm this.

e. $LWS$ and $\{LWS\}$ are similar to $WS$ and $\{WS\}$. The only difference is that a component in $LWS$ is the last web service in a path. In Figure 3, every component of a $LWS$ is double-circled. Note that every component in a $LWS$ is also a component of a $WS$.

![Figure 3. The WSIG for the solution "Rent a house"](image)

After constructing the web service invocation graph, the upper level access control policy is applied to filter out the web services that cannot be invoked by the requester. Both credentials and attributes are used in the policy. Suppose: (a) the credentials required by a web service constitute a set $WSCD$, (b) the attributes whose values required by a web service are enumerations constitute a set $WSATTENU$, and
(c) the attributes whose values required by a web service are ranges constitute a set $WSATTRNG$. Then, we need the following definitions to define the rules for the upper level access control policy.

**Definition 3.** $WSCD = \{(cd_i, pwd_i) \mid (cd_i, pwd_i) \text{ is a credential/password pair, in which } cd_i \text{ is a credential and } pwd_i \text{ is the password for } cd_i\}$.

**Definition 4.** $WSATTENU = \{\{\text{attenu}_i, \{\text{attval}_i\}\} \mid \text{attenu}_i \text{ is an attribute; } \{\text{attval}_i\} \text{ is a set of } \text{attenu}_i\text{'s values that are acceptable by the web service}\}$.

**Definition 5.** $WSATTRNG = \{\{\text{attrng}_i, \text{attrng}_{\text{min}_i}, \text{attrng}_{\text{max}_i}\} \mid \text{attrng}_i \text{ is an attribute; } \text{attrng}_{\text{min}_i} \text{ and } \text{attrng}_{\text{max}_i} \text{ are respectively the minimum and maximum values of } \text{attrng}_i\}$.

According to the definitions above, the rules used in the upper access control policy are described below:

**URule 1.** If the credentials owned by a requester is $RQCD$, then $RQCD \supseteq WSCD$.

**URule 2.** If a web service requires that $VAL(\text{attenu}_i) \in \{\text{attval}_i\}$, then the requester should possess an attribute $\text{attenu}_i$ and $VAL(\text{attenu}_i) \in \{\text{attval}_i\}$. Here $VAL(\text{attenu}_i)$ extracts the value of $\text{attenu}_i$.

**URule 3.** If a web service requires that $VAL(\text{attrng}_i)$ should be between $\text{attrng}_{\text{min}_i}$ and $\text{attrng}_{\text{max}_i}$, then the requester should possess an attribute $\text{attrng}_i$ and $VAL(\text{attrng}_i)$ should be between $\text{attrng}_{\text{min}_i}$ and $\text{attrng}_{\text{max}_i}$.

**URules** 1 through 3 should be simultaneously fulfilled for a web service. Otherwise, the web service fails to pass the upper level access control policy and will be filtered out. To facilitate the following discussion, we suppose that Figure 4 is the $WSIG$ after applying the $URules$.
After applying the upper level access control policy, we identify paths from the WSIG. As mentioned before, our algorithm does not consider loops currently. A graph without loops can be transferred into a tree to identify paths. To transfer a graph into a tree, we need to define a sub-service invocation graph (SubWSIG) rooted at a node $w_{sk}$ as follows (suppose SubWSIG is a sub-graph of WSIG).

**Definition 6.** SubWSIG = (SubWS, SubAR, REL, SubLWS), in which

a. $SubWS = \{w_{sk}\} \cup \{w_{si} | w_{si} \text{ is a web service reachable from } w_{sk} \text{ in WSIG}\}$.

b. $SubAR = \{<w_{si}, w_{sj}> | <w_{si}, w_{sj}> \text{ is an arrow in WSIG and } \{w_{si}, w_{sj}\} \in SubWS\}$.

c. REL is the same set in Definition 1.

d. $SubLWS \subseteq SubWS$ and every component in $SubLWS$ is the last web service in a web service path.

As an example, Figure 5 is a SubWSIG of Figure 4 rooted at web service 5.1. To transfer a WSIG into a tree, the graph should be transferred into a graph where every node in the latter graph is pointed by exactly one arrow (the latter graph is a tree). With Definition 6 in mind, the operation TrTree listed below will transfer a WSIG into a web service invocation tree (WSIT).

**TrTree.** Repeat

- Identify a node $ND_i$ in WSIG with $N$ incoming arrows in which $N > 1$;
- Duplicate $N$ subWSIG rooted at $ND_i$;
- Let every incoming arrow point to the root node of one subWSIG;
- Until every node is pointed by only one incoming arrow;
After applying TrTree, the web service invocation graph in Figure 4 will be transferred into the WSIT in Figure 6.

Having transferred a WSIG into a WSIT, we can identify paths from the WSIT. To facilitate path identification, we need the definition of a path PTH as follows. Note that every PTH accomplishes the solution Dsol in Definition 1.

**Definition 7.** $PTH = (WS, AR, REL, LWS)$, in which

a. $WS = \{wsi \mid wsi$ is a web service that accomplishes a sub-function of Dsol and different web services accomplish different sub-functions\}$. It is important that only one web service is needed for a sub-function in a path.

b. $AR = \{<wsi, wsj> \mid <wsi, wsj>$ is an arrow from wsi to wsj, and $\{wsi, wsj\} \subseteq WS\}$.

c. $REL = \{AND-split, AND-join\}$. AND-split will be handled during path identification and AND-join will be handled after the identification. In a WSIT, if two or more arrows are covered by an AND-split relationship, the web services pointed by the arrows should all within WS in a path because they should all be finished to accomplish Dsol.

Since a leaf of a tree can belong to only one path, we identify paths using
backtracking process started from leaf nodes. The backtracking process will be rewound into a forward tracing process when a node with an AND-split relationship is encountered. In the case, a depth first process will be used to identify all web services covered by the AND-split relationship. The operation \textit{IdPth} below identifies paths from the web service invocation tree \textit{WSIT}, in which the descriptions following the symbol “//” are comments.

\textbf{IdPth. Repeat} \hspace{1em} //Cnode is the current node, \(ND_i, ND_j, ND_k\), are nodes, \textit{STK} is a stack

Mark an unmarked leaf \(LF_i\) in \textit{WSIT};

\(Cnode = LF_i;\)

Repeat

\hspace{1em} Loop \hspace{1em} //Backtrack the tree

If \(<ND_i, Cnode>\) exists and no AND-split relationship covers \(<ND_i, Cnode>\), then

\hspace{1em} Record \(ND_i, Cnode, \) and \(<ND_i, Cnode>\);

\hspace{1em} If \(ND_i\) is the first node in the \textit{WSIT}, then

\hspace{1em} A path consisting of the recorded information has been identified;

Else

\hspace{1em} \(Cnode = ND_i;\)

End loop;

If an AND-split relationship covers the arrows in the set \(<ND_j, Cnode>\), then

For each \(<ND_j, Cnode>\), do //Forward tracing

\hspace{1em} Record \(ND_j, Cnode, <ND_j, Cnode>\), and the AND-split relationship;

\hspace{1em} Push\((Cnode, \textit{STK});\) //Push Cnode into the stack

\hspace{1em} \(Cnode = ND_j;\)

Loop

\hspace{1em} If \(<Cnode, ND_k>\) is covered by the AND-split relationship, then

\hspace{1em} Record \(ND_k, Cnode, <Cnode, ND_k>\);

\hspace{1em} Repeat //Forward tracing ends at a leaf

\hspace{1em} If \(ND_k\) is a leaf, then

\hspace{1em} \(Cnode = \text{Pop} (\textit{STK});\)

Else

\hspace{1em} Push\((Cnode, \textit{STK});\)

\hspace{1em} \(Cnode = ND_k;\)

\hspace{1em} Until \(ND_k\) is a leaf;

End Loop;

End For;

Until a path is identified;

Until every leaf is marked; //Every leaf is marked means every path is identified
We use Figure 7 to explain the *IdPth* operation. Suppose the operation starts from leaf node 1, it will record nodes 1 and 9 and the arrow <1, 9>. It will then record nodes 9 and 14 and <9, 14>. Since <9, 14> is covered by an AND-split relationship, *IdPth* will forward trace the tree according to the AND-split relationship. It first push node 14 into STK and then trace from node 14 to 10 and then from node 10 to 3. It then backs to node 10 and traces to node 4. The trace follows the depth first procedure and will record every nodes, arrows, and AND-split relationships it traces. Eventually, node 14 is popped. Since no other arrows are covered by the AND-split relationship. The operation *IdPth* backtracks to node 17, which is invoked by the requester. Since <17, 14> is not covered by an AND-split relationship, a path has been identified. The path is composed of the nodes linked by the dotted arrows and the AND-split relationships in Figure 7.

![Figure 7. Example web service invocation tree](image)

![Figure 8. AND-join management](image)
After all paths are identified, the AND-join is manipulated. In the manipulation, duplicated web services in a path are AND-join ones. For example, there are two web service 8.3 appears in the path linked by dotted lines in Figure 6. In this case, the web service will be executed only once when web services 6.7 and 7.6 finish. As another example, the path identified in Figure 8(a) shows that web service 3 is an AND-join. In this case the composition algorithm will adjust the path in Figure 8(a) to become 8(b). The AND-join in a path $P_{TH}$ is manipulated using $ANDjOP$ operation below.

$ANDjOP$. Repeat

If web service $WS_i$ appears in $P_{TH}$ $N$ times in which $N > 1$ then

- Merge the $N$ $WS_i$ into one;
- Redirect the arrows pointing to any of the original $WS_i$ to the merged $WS_i$;
- Place an AND-join to cover the arrows pointing to the merged $WS_i$;
- Let the arrows pointing out from the original $WS_i$ be pointed out from the merged $WS_i$;
- Place an AND-split to cover the arrows pointing out from the merged $WS_i$;

End if;

Until no duplicated web service in $P_{TH}$;

After the adjustment of paths for AND-join, Formula 1 is applied to compute the possibility of a requester to invoke a web service (i.e., the lower level web service access control policy is applied). The possibilities are then marked on the arrows of the paths. For example, the number 0.69 on the arrow $<9, 1>$ in Figure 7 means that the possibility of the requester to successfully invoke web service 1 is 0.69.

After applying the lower level access control policy, the successfulness possibility for every path should be computed. The possibility $PTH_{succ}$ for the path $P_{TH}$ can be computed using Formula 2 below, in which web services in $P_{TH}$ constitute the set \{ws\} and $poss_i$ is the possibility for $ws_i$ obtained from Formula 1.

$$PTH_{succ} = \prod_i poss_i$$

(2)

If $PTH_{succ}$ for a path is too small, the path should be deleted. Therefore, a threshold $ThdPTH_{succ}$ for $PTH_{succ}$ should be set by the requester. This induces the successful possibility rule ($SUCCRule$) below.

$SUCCRule$. $PTH_{succ} \geq ThdPTH_{succ}$
After computing the successfulness possibility for every path, the QoS value for every QoS criterion of the path should be computed. In the computation, the operator (e.g., addition or multiplication) applied to the criterion should first be determined. Then the value $PTHQoScv_i$ of the $i^{th}$ QoS criterion for the path $PTH$ is computed using Formula 3, in which $QoS_{ws_{ij}}$ is the $i^{th}$ QoS criterion’s value for $ws_j$ in $PTH$.

$$PTHQoScv_i = \sum_j QoS_{ws_{ij}} \text{, if the operator is addition}$$

$$PTHQoScv_i = \prod_j QoS_{ws_{ij}} \text{, if the operator is multiplication} \quad (3)$$

To ensure the quality of a path, the value $PTHQoScv_i$ of the $i^{th}$ QoS criterion for the path should pass the limitation $PTHQoScli$. That is, the $PTHQoSRule$ (path QoS rule) below should be fulfilled for a path. Otherwise, the path will be deleted.

**PTHQoSRule:** $\forall PTHQoScv_i, PTHQoScv_i \geq PTHQoScli$

If a path $PTH$ passes the $PTHQoSRule$, its QoS value $PTHQoSval$ (path QoS value) is computed using Formula 4, in which $PTHQoScv_i$ is the the $i^{th}$ QoS criterion’s value for the path and $Wp_i$ is the weight of $PTHQoScv_i$. $Wp_i$ is decided by the requester and $\sum_i Wp_i = 1$.

$$PTHQoSval = \sum_i PTHQoScv_i * Wp_i \quad (4)$$

By now, we have computed two values for every path, which are: (a) the successfully finishing possibility $PTHsucc$ and (b) the QoS value $PTHQoSval$. Using the two values, we can compute an overall value $PTHVAL$ for the path $PTH$ using Formula 5 below, in which $w_1$ and $w_2$ are respectively the weights of $PTHsucc$ and $PTHQoSval$. Their summation is 1.

$$PTHVAL = PTHsucc * w_1 + PTHQoSval * w_2 \quad (5)$$

The overall value $PTHVAL$ facilitates selecting web service paths. Since different requesters may give different values to $w_1$ and $w_2$, they may select different path(s). Below we use an algorithm to summarize the above description.
Exhibition 1. The path composition algorithm

Input: \( DSOL = \{ Dsol_i \mid Dsol_i \) is a Dsol in Definition 1\}

\[ RQCD = \{ rqcdpwd, \mid rqcdpwd is a pair \( (rqcd, pwd) \), in which \( rqcd \) is a credential owned by a requester and \( pwd \) is the password of \( rqcd \)\}\]

\[ RQATT = \{ (rqatt, rqattval) \mid rqatt is an attribute of the requester, rqattval is the value of the attribute \( rqatt \)\}\]

// Other input data are omitted

Output: \( PTHS = \{ pth_i \mid pth_i \) is a web service path to accomplish a Dsol in DSOL\}

Algorithm:

1. For each \( Dsol_i \) in \( DSOL \), do
   1.1 Apply a selection algorithm to construct a WSIG (see Definition 2) for \( Dsol_i \)
   1.2 Apply WSQoSOp for every web service in WSIG
   1.3 Delete all web services from WSIG that fail to fulfill WSQoSRule
   1.4 Delete all web services from WSIG that fail to fulfill any of URules 1 through 3
   //Apply the upper level access control policy
   1.5 Apply the operation TrTree to transfer WSIG into a WSIT
   1.6 Apply the operation IdPth on WSIT to identify web service paths
   1.7 Apply the operation ANDjOP for every path to handle AND-join
   1.8 Apply Formula 1 on WSIT to compute the possibility of a requester to invoke a web service  // Apply the lower level access control policy
   1.9 For each path identified from Step 1.6, do
      1.9.1 Apply Formula 2 to compute the successfulness possibility for the path \( PTHsucc \)
      1.9.2 If SUCCRule is not fulfilled, delete the path and go back to Step 1.9
      1.9.3 For i = 1 to n, do // Suppose there are n QoS criteria
         1.9.3.1 Apply Formula 3 to compute the QoS value \( PTHQoS_{cvi} \) of the \( i^{th} \) QoS criterion for the path
      1.9.4 If PTHQoSRule is not fulfill, delete the path and go back to Step 1.9
      1.9.5 Apply Formula 4 to compute the QoS value \( PTHQoSval \) of the path
      1.9.6 Apply Formula 5 to compute the overall value \( PTHVAL \) for the path
   1.10 Select web service paths (by the requester) from those that are not deleted by Step 1.9 and place them into \( PTHS \) //The overall value \( PTHVAL \) can be used as a reference in the selection

4. EVALUATION

Our method selects multiple secure web service paths from a large set of possible paths. The usefulness of healing ill path using spare paths can be predicted (the researches in [11, 15, 32, 37] mentioned this type of healing). On the other hand,
embedding access control policy in a web service composition algorithm seems unusual but useful. We use experiments to evaluate the usefulness of our approach. The solution “Rent a house” of the requirement “Get a house to live in” (see Figure 2) is used as an example in the experiments. We selected ten students to attend the experiments. In the experiment, we first clarified the necessity of access control in path composition. We then evaluated how the access control policy affects the successfulness of a composed path.

Before the experiments, we required the students to implement thirty web services for each sub-function in Figure 2 and gave the web services different QoS criteria values, attributes, and credentials. In addition, each web service was given an initial credit level number. Moreover, we let the students (i.e., requester) possess the same attributes and credentials.

![Figure 9. The experiment data of the first experiment](image)

To clarify the necessity of access control in path composition, we required the students to: (a) compose 20 paths using our algorithm but skip the access control policy and (b) compose 20 paths without skipping the policy. After the composition, the students executed the paths and checked the successfulness of the execution (i.e., counted the number of paths not banned by the access control policy). During the execution, we set up access control parameters including attribute and credential requirements for web services, credit levels of web services, and security levels of arguments. The experiment result is shown in Figure 9, which depicts the importance of considering access control during path composition.

To clarify how the access control policy affects the successfulness of a path, the students were required to select at least five web services for each sub-function and then composed only one path by applying our access control policy and algorithm. To prevent the student from composing the same path, we required them to select different web services for the same sub-function. If more than one student did compose the same path, we required them to replace one or more web services in the
path. According to the above manipulation, the students composed ten different paths. We then required the students to execute the paths. During execution, we required the students to simulate the cases from no dishonest web service up to seven dishonest ones (to simulate a dishonest web service, we required the students to decrease its credit level number every delta time). The above simulation repeated ten times. We required the students to composed different path each time. We then collected the successful possibilities of the paths composed by the student. The experiment result is shown in Figure 10. Since we only repeated ten times of the simulation, Figure 10 may look somewhat strange. Nevertheless, we can still identify the following important information from the figure:

a. The figure shows that even every web service is honest, execution failure might still occur on the path. This could be a consequence of small $k$ value in Formula 1.
b. The credit level number of dishonest web services will be decreased every delta time. Therefore, more dishonest web services resulted in less successful possibilities.

![Figure 10. The experiment data of the second experiment](image)

### 5. PROBLEM DISCUSSION

The lower level of the proposed access control policy is closely related to requesters and web service providers around the world because the credit level numbers of web services will be affected by them, as described below:

a. It is possible that requesters don’t know the meaning and the use of web services’ credit level numbers. It is also possible that requesters don’t know that web services may leak information during execution. In the cases, requesters will not change the credit level numbers of web services. This will cause our lower level access control policy to become meaningless.
b. It is possible that a web service leaks information but the leakage is not detected. Undetected information leakages will not cause the decrement of web services’ credit level numbers. This again causes our lower level access control policy to become meaningless.

As mentioned before, requesters and providers are around the world and generally independent. They need to be regulated by standards and the standards should be managed by trustable organizations, such as IBM and Microsoft. The problems listed above can be solved through standards and trustable organizations. That is, an organization trusted by every requester and provider should apply a standard mechanism to detect information leakage and handle the increment and decrement of the credit level numbers. Under this circumstance, the problems mentioned above can be solved. Although convincing a trustable organization to use the access control policy proposed in this paper is not easy, we believe that a standard offering precise access control will be defined sooner or later. When the standard is defined, we hope that the concept of our two-leveled access control policy can be embedded. Before a precise access control standard is defined, the less precise access control standard XACML can be applied to implement our access control policy. Note that the experiments described in section 4 were obtained from a simulated environment instead of the XACML-based environment because the latter environment is still under implementation. The XACML-based environment is described below.

As described in section 2, XACML offers mechanisms to describe access control policies for web services. Policies that can be described are limited to the static aspect, such as attribute and credential checking. Therefore, XACML can implement our upper level access control policy. As to our lower level policy, it
belongs to the dynamic aspect because credit level numbers of web services may be changed from time to time. XACML cannot implement the policy in the dynamic aspect. To overcome this shortcoming, XACML should be extended by adding a proxy. The proxy records web service credit level numbers and handles their increment and decrement. In the implementation, XACML should be managed by a trustable organization. We use Figure 11 to depict an implementation of our access control policy using XACML. The figure shows that our upper level policy is implemented in XACML and the lower level policy in a proxy. Moreover, the standards XACML and Broker are managed by a trustable organization. As to the proxy, since it is not a standard, the trustable organization may be unwilling to manage it. Therefore, the function of the proxy is invoked by XACML. The implementation is operated as follows. When a requester identifies that a web service leaks information, the requester notifies the trustable organization to decrease the credit level number of the web service. The organization transfers the request to XACML. XACML then checks whether the requester is trustable. If the answer is positive, XACML requests the proxy to decrease the number as requested. Otherwise, the number is unchanged. If the credit level of a web service is not decreased for a period of time, the number is increased by the proxy. With the upper access control policy implemented in XACML and the lower one in the proxy, our path composition algorithm can function normally. Since XACML must confirm the existence of a web service through the assistance of a broker, Figure 11 shows the cooperation between XACL and the broker.

6. CONCLUDING REMARK
A requester invokes web services to accomplish his requirements, which can usually be solved by multiple solutions. To accomplish a solution, the solution should be decomposed into sub-functions, in which each sub-function can be accomplished by a web service. In this regard, the web services accomplishing a solution constitute a web service path.

When composing web service paths to accomplish a solution, the security of web service access should be considered to improve the successfulness of path execution. Current access control policies generally protect web services. Nevertheless, the sensitive information sent to a web service may be leaked by the web service. Accordingly, requesters should also be protected. Another problem regarding web service composition is that currently available algorithm generally composes one optimal path. If the execution of the path fails, the time consuming path replanning process should be used to heal the path. In our opinion, a web service composition algorithm can compose multiple paths. When failure occurs on the
executing path, the sub-paths or web services of the spare paths can be used to continue the execution. This may reduce the effort of the path replanning process.

This paper proposes a two-leveled web service access control policy and a path composition algorithm, in which the policy is embedded in the algorithm. The upper level access control policy protects web services using attributes and credentials to filter out the web service that cannot be invoked by a requester. The lower level access control policy compares the credit level numbers of web services with the security level numbers of arguments. It then evaluates the possibility of leaking the arguments by a web service. In other words, the lower level policy protects the requesters. After the two-leveled access control, the composition algorithm composes multiple paths that fulfill the QoS criteria.

We use an experiment to clarify the necessity of considering access control during path composition. The experiment result shows that the successful rate of a path is higher if the composition algorithm takes access control into consideration. We also use an experiment to clarify how the access control policy affects the successfulness of a path. From the experiment, we identify that the possibilities computed in the lower level access control policy is useful in protecting requesters’ sensitive information (Figure 10 proves this, in which more dishonest web services results in less successful possibility).

Currently, our path composition algorithm can only operate on web service invocation graphs without loops. According to others’ researches, handling loops in a composition algorithm is not easy. In the future, we will enhance our algorithm to operate on web service invocation graphs with loops. Perhaps grouping a sub-web service graph containing loops into a web service pack may work. After the grouping, the graph contains no loops and our algorithm can be applied. We then solve the loops within the web service packs. However, we are still researching on the problem.
REFERENCES


Intelligent Agent Technology, 598-604.


34. W3C. Web Services Description Language (WSDL) 1.1. *available on* [http://www.w3.org/TR/2001/NOTE-wsdl-20010315](http://www.w3.org/TR/2001/NOTE-wsdl-20010315)


