Embedding Access Control Policy in Web Service Path Composition Algorithm

SHIH-CHEI CHOU AND JIN-YUAN JHU
Department of Computer Science and Information Engineering
National Dong Hwa University
Hualien, 974 Taiwan
E-mail: scchou@mail.ndhu.edu.tw

Web services accomplish requirements, which are complicated functions. To apply web services for a requirement, it should be decomposed into sub-functions for web services. After the decomposition, web services are selected to compose paths. 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. 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 compares the credit level numbers of web services with the security level numbers of arguments to evaluate the possibility of leaking the arguments. The possibility facilitates evaluating the successfulness of executing a path. After access control, the composition algorithm composes multiple paths.

Keywords: web service, access control, access control policy, web service composition, information flow control

1. INTRODUCTION

Web services serve requesters to accomplish their requirements. A requirement can usually be solved by multiple solutions. To accomplish a requirement, the requester uses web services to accomplish solutions. Since solutions are generally complicated, a solution should be decomposed into sub-functions that can be accomplished by web services. After that, selection [2, 13] identifies web services and composition [7, 9, 10, 14] composes a path. During selection and composition, the quality of service (QoS) criteria should be fulfilled [15, 28]. If problems occur, path replanning can heal the path [3, 20]. Finishing the path implies accomplishing the solution. There may be dishonest providers that cheat money or even steal sensitive information. Since requesters cannot distinguish honest providers from dishonest ones, requesters may invoke web service provided by dishonest providers, applying web services to accomplish solutions takes risks and security issues such as access control should be taken into consideration.

We researched on secure web service access. Existing access control policies decide whether a requester can invoke a web service. That is, the policies protect web services. In our opinion, requesters should also be protected. We thus designed a trust management mechanism [8]. 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. This protects the arguments sent to the web service. Our research also identifies that existing composition algorithms 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. We also noticed that existing composition algorithms generally compose only one “optimal” path. If web service(s) are selected from dishonest providers, the most optimal path may be not optimal. Our experience showed that embedding access control policy in composition can reduce execution failure. We also identified that if a path composition algorithm compose more than one path, the effort of path replanning can be reduced because of spare paths replacement [11, 26].

According to the above description, we designed a path composition algorithm that embeds a web service access control policy. The policy is composed of two levels. The upper level protects web services using attributes and credentials. The lower level is an adaptation of our trust management mechanism [8]. The mechanism compares the credit level numbers of web services with the security level numbers of arguments. 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 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, the credit level number of a web service during path composition may be different from that during path execution. Accordingly, our lower level access control policy adapts the original trust management mechanism using a possibility to replace the definite answer. After access control, our composition algorithm composes multiple paths that fulfill the QoS criteria. This paper proposes the access control policy embedded path composition algorithm.

2. RELATED WORK

The standards WS-Security [18] and XACML (extensible access control markup language) [17] are related to web service security. WS-Security is for cryptography. XACML [17] offers mechanisms to describe access control policies for web services. The model in [23] is an implementation of XACML, in which 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 for “role-base access control” [21]). The model in [27] defines its access control policies using two-leveled mechanism. The first level checks the roles assigned to both requesters and web services. A requester passing the first level should also pass the second level of the control, which is the attribute level. The model in [5] controls web service access using X-GTRBAC (XML-based generalized temporal RBAC) [6], which can be used in heterogeneous and distributed sites. It also applies TRBAC (temporal RBAC) [4] to control the factor of time. The model in [22] offers a language to enforce access control specification (access control policy). The model in [25] uses the language “pure-past linear temporal logic” to determine whether a composition of web service can be invoked. Our previous work [8] proposes a trust management mechanism for web service access control described in section 1. The model
Trust-Serv [24] dynamically chooses web services. The choosing is based on trust negotiation [29], which selects web services using credentials. The model in [12] defines strategies for negotiation policies using credentials. The model in [16] 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.

Since we embed access control in a web service composition algorithm, we also survey path decomposition. The model in [14] proposes the Quality and Relation Driven Services Composition (QRDSC) approach to compose web service paths. During composition, the relationships among services are considered in addition to the QoS. The model in [10] proposes an approach that takes the tightness of QoS criteria into consideration during web service composition. The model in [9] proposes an approach to dynamically compose web services. The approach is useful in an error-prone environment to improve the reliability of the composed path. The model in [7] 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.

3. THE APPROACH

3.1 Two-Level Access Control Policy

The upper level policy uses attributes and credentials to decide whether a requester can invoke a web service. If a requester possesses the credentials required by a web service and he knows the passwords of the credentials, the credential checking passes. As to attribute checking, we use enumeration and range to define the attributes acceptable by a web service. When a requester passes the upper level access control policy, he should also pass the lower policy. The policy is adapted from the trust management mechanism [8]. The original trust management mechanism uses the security level numbers of arguments and the credit level numbers of web services, as described below. The numbers are between 0 and a maximum value determined by the user.

(a) Security level numbers. Every argument passed to a web service is associated with a security level number. A larger number implies a more sensitive argument.
(b) Credit level numbers. Every web service is associated with a credit level number. A larger number implies a more trustable web service.

When a requester invokes 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 $\text{ARG}$ defined as $\{\text{ARG}_i \mid \text{ARG}_i$ is an argument$\}$; (b) the web service to be invoked is $\text{WEBSER}$; (c) $\text{ARG}_{clu}$ is the security level number of $\text{ARG}_i$; and (d) $\text{WEBSER}_{clu}$ is the credit level number of $\text{WEBSER}$. With the assumptions, if $\text{WEBSER}_{clu} \geq \text{Max}(\text{ARG}_{clu})$, the mechanism allows the invocation.

In the trust management mechanism, if a web service does not leak sensitive information for a period of time, its credit level number will be increased. If the web service leaks sensitive information, its credit level number is decreased. Since the lower level access control policy is adapted from the trust management mechanism, a requester is
allowed to invoke a web service only when \( \text{WEBSER} \geq \text{Max}(\text{ARG}_{i|\text{arg}}) \). However, since credit level numbers change from time to time, it is possible that the condition is true during composition but false during execution, and vice versa. The lower level policy uses a \textit{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 \( \text{ARG} \) to the web service \( \text{WEBSER} \). We also make the assumptions: (a) \( \text{ARG} = \{ \text{ARG}_i \mid \text{ARG}_i \text{ is an argument} \} \); (b) \( \text{ARG}_{i|\text{arg}} \) is the security level number of \( \text{ARG}_i \); (c) \( \text{WEBSER}_{\text{cln}} \) is the \textit{current} credit level number of \( \text{WEBSER} \); (d) \( n \) is a threshold to determine the possibility, which is defined by the user; (e) \( k = \text{WEBSER}_{\text{cln}} - \text{Max}(\text{ARG}_{i|\text{arg}}) \), which is used to simplify the following discussion; and (f) \( \text{POSI}_{\text{succ}} \) is the possibility that the requester can successfully invoke \( \text{WEBSER} \).

According to the assumptions above, if \( k \) is smaller than \(-n\), the credit level number of \( \text{WEBSER} \) is too small and \( \text{POSI}_{\text{succ}} \) is 0. If \( k \) is larger than \( n \), the credit level number of \( \text{WEBSER} \) is large enough and \( \text{POSI}_{\text{succ}} \) is 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 \(-n\) to be 0. Then, \( k \) 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 Eq. (1) below.

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

In the adaptation, if a requester identifies that a web service leaks information, the requester notifies a trustable organization to decrease the credit level number of the web service. The organization checks whether the requester is trustable. If the answer is positive, the number is decreased. Otherwise, the number is unchanged. If a credit level is not decreased for a period of time, the number is increased by the organization.

### 3.2 Information Leakage Detection

Detecting information leakage is crucial because it will change credit level numbers. Our information follow control model EXACML [8] is used for the detection.

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

(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). Letting \( \text{MD} \) be the set of object methods, the ACL \( \text{ACL}_{\text{var}} \) attached to a variable \( \text{var} \) is defined as \( \text{ACL}_{\text{var}} = \{ \text{RACL}_{\text{var}}, \text{WACL}_{\text{var}} \} \), in which (1) \( \text{RACL}_{\text{var}} \in 2^{\text{MD}} \). Object methods in \( \text{RACL}_{\text{var}} \) are allowed to read \( \text{var} \) and (2) \( \text{WACL}_{\text{var}} \in 2^{\text{MD}} \). Object methods in \( \text{WACL}_{\text{var}} \) are allowed to write \( \text{var} \).
(f) \( \text{DSOURCES} \) is a set of data sources (DSOURCEs).
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: \((RACL_{d_{var}} \subseteq \bigcup_{i=1}^{n} RACL_{vari}) \land (md1 \in \bigcap_{i=1}^{n} RACL_{vari})\)

Second EXACML secure rule: \(WACL_{d_{var}} \supseteq \bigcup_{i=1}^{n} DSOURCE_{vari} \cup \{md1\}\)

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 \(\{vari \mid vari \in VAR\}\) 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 \(\{RACL_{vari}, WACL_{vari}\}\); and (e) the DSOURCE of \(vari\) is \(DSOURCE_{vari}\). The first rule requires that \(d_{var}\) must be at least the same sensitive as the variables deriving \(d_{var}\). 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 \(d_{var}\). 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 \(\oplus\) as follows,

\[\bigoplus_{i=1}^{n} ACL_{vari} = (\bigcap_{i=1}^{n} RACL_{vari} \cap \bigcup_{i=1}^{n} WACL_{vari}).\]

Join trusts less or the same set of readers and will not lower down security level. Moreover it 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 \(d_{var}\) will be adjusted as follows,

\[DSOURCE_{d_{var}} = \bigcup_{i=1}^{n} DSOURCE_{vari} \cup \{md1\} \].

3.3 Access Control Policy Embedded Composition Algorithm

As mentioned, solutions accomplish requirements. Since solutions are complicated, a solution should be decomposed into sub-functions for web services. Fig. 1 depicts the

Fig. 1. The decom position of the solution “Rent a house”.

Start
1. Collect information of houses for rent
2. Negotiate price with broker
3. Negotiate price with house owner
4. Pay by check
5. Pay by credit card
6. Get house renting license
7. Get receipt
8. Move in
Stop
decomposition of the solution “Rent a house”. Below we give a definition $D_{sol}$ for the decomposition of a solution.

**Definition 1** $D_{sol} = (SF, AR, REL)$, in which
(a) $SF = \{sf_i | \text{sf}_i$ is a sub-function that can be accomplished by a web service$\}$.
(b) $AR = \{<\text{sf}_i, \text{sf}_j> | <\text{sf}_i, \text{sf}_j>$ is an arrow from $\text{sf}_i$ to $\text{sf}_j$, and $\{\text{sf}_i, \text{sf}_j\} \subseteq SF\}$. The existing of $<\text{sf}_i, \text{sf}_j>$ means that the sub-function $\text{sf}_i$ can be executed after $\text{sf}_j$.
(c) $REL = \{SEQ, \text{AND-split, AND-join, OR}\}$, in which: (a) $SEQ$ defines the execution sequence of sub-functions, (b) AND-split and AND-join respectively define the splits and joins for parallel sub-functions, and (c) OR define alternative sub-functions. We do not include loop because our composition algorithm transfers a graph into a tree. Loop is uneasy to solve. The research in [19] stated that finding an optimal path from a graph with loops is NP-hard. The paper [1] suggests to identify a sub-optimal solution. The paper [30] uses logs to offer a sub-optimal solution.

After decomposing a solution, web services should be selected for sub-functions, during which the QoS criteria should be fulfilled. Since some criteria should be small and some be large. 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. To check QoS, $WSQoSOp$ (web service QoS managing operation) should first be applied. Then, the selected web services should pass the rule $WSQoSRule$ (web service QoS checking rule).

$WSQoSOp$: \forall WSQoSc_i \text{ if } WSQoSc_i \text{ should be small, then } WSQoSc_i = WSQoSmax_i - WSQoSc_i$, in which (a) $WSQoSc_i$ is the $i$th web service QoS criterion, (b) $WSQoSmax_i$ is the value of $WSQoSc_i$, and (c) $WSQoSmax_i$ is the maximum value of $WSQoSc_i$.

$WSQoSRule$: \forall WSQoSc_i, WSQoSc_i > WSQoScl_i$, in which $WSQoScl_i$ is the minimum value for the $i$th web service QoS criterion.

Selection results in a web service invocation graph ($WSIG$). Fig. 2 depicts a $WSIG$ for Fig. 1. Circles are web services. Below we give a definition for a $WSIG$.

**Definition 2** $WSIG = (\{WS\}, AR, REL, \{LWS\})$, in which
(a) $WS = \{wsi | \text{wsi}$ is a web service for $\text{sf}_k$, in which $\text{sf}_k \in SF$ in Definition 1$\}$.
(b) $\{WS\} = \{WS_i | \text{WS}_i$ is a $WS$. The union of $\text{WS}_i$ constitutes $WSIG$ web services$\}$.
(c) $AR$ is the set of arrows. If an arrow exists between sub-functions $i$ and $j$ in a $D_{sol}$, an arrow exists between every component of $\text{WS}_i$ and that of $\text{WS}_j$.
(d) $REL$ is the same as that in Definition 1. If a relationship covers an arrow of a sub-function in a $D_{sol}$, the relationship covers the corresponding arrow of the web services that accomplish the sub-function.
(e) $LWS$ and $\{LWS\}$ are similar to $WS$ and $\{WS\}$. Component in $LWS$ is the last web service in a path (double-circled ones in Fig. 2).
After constructing WSIG, the upper level access control policy is applied. 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 $WSATTRNG$. With the assumptions and Definitions 3 through 5 below, rules used in the upper access control policy are listed in URules 1 through 3.

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

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

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

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

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

**URule 3** If a web service requires that $VAL(\text{attrng}_i)$ should be between $\text{attrng}_{\text{min}_i}$ and $\text{attrng}_{\text{max}_i}$, 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 (suppose Fig. 3 is the $WSIG$ after applying the $URules$). After applying the upper level access control, we identify paths from the $WSIG$. We first transfer a $WSIG$ into a web service invocation tree ($WSIT$). To transfer a $WSIG$ into a $WSIT$, we need to define a sub-web service invocation graph ($SubWSIG$) rooted at a node $w_{sk}$ as follows.

**Definition 6** $SubWSIG = (SubWS, SubAR, REL, SubLWS)$, in which
(a) $SubWS = \{w_{sk}\} \cup \{w_{si} | \text{\(w_{si}\) is a web service reachable from \(w_{sk}\) in \(WSIG\)}\}$. 
(b) \( \text{SubAR} = \{ \langle w_{s_i}, w_{s_j} \rangle \mid \langle w_{s_i}, w_{s_j} \rangle \in \text{SubWS} \} \).

(c) \( \text{REL} \) is the same set in Definition 1.

(d) \( \text{SubLWS} \subseteq \text{SubWS} \) and a component in \( \text{SubLWS} \) is the last web service in a path.

To transfer a \( \text{WSIG} \) into a \( \text{WSIT} \), the operation \( \text{TrTree} \) listed below can be used. \( \text{TrTree} \) will transfer Fig. 3 into Fig. 4.

\[
\text{TrTree. Repeat}
\]

Identify a node \( N_D \) in \( \text{WSIG} \) with \( N \) incoming arrows in which \( N > 1 \);

Duplicate \( N \) sub\( \text{WSIG} \) rooted at \( N_D \);

Let every incoming arrow point to the root node of one sub\( \text{WSIG} \);

Until every node is pointed by only one incoming arrow;

---

Fig. 3. The \( \text{WSIG} \) for Fig. 3 after applying the upper level access control policy.

Fig. 4. The web service invocation tree obtained from Fig. 3.

After the transferring, paths can be identified from the \( \text{WSIT} \). To facilitate path identification, we need the definition of a path \( \text{PTH} \) as follows,

**Definition 7** \( \text{PTH} = (WS, AR, REL, LWS) \), in which
(a) \( WS = \{w_{s_i} \mid w_{s_i} \) is a web service that accomplishes a sub-function of \( Dsol \) and differ-
ent web services accomplish different sub-functions).

(b) \( AR = \{ <wsi, wsj> | <wsi, wsj> \text{ is an arrow and } \{ wsi, wsj \} \subseteq WS \} \).

c) \( REL = \{ \text{AND-split, AND-join} \} \). 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.

d) LWS is similar to WS. Components in LWS are the last web services in a path.

Since a leaf can belong to only one path, we identify paths using a 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 \( IdPth \) below identifies paths from a WSIT.

\( IdPth \). Repeat // \( Cnode \) is the current node, \( ND, ND, ND, \) are nodes, STK is a stack
Mark an unmarked leaf \( LF_i \) in WSIT;
\( Cnode = LF_i \);
Repeat
Loop // Backtrack the tree
If \( <ND, Cnode> \) exists and no AND-split relationship covers \( <ND, Cnode> \), then
Record \( ND, Cnode \), and \( <ND, Cnode> \);
If \( ND \) is the first node in the WSIT, then
A path consisting of the recorded information has been identified;
Else \( Cnode = ND \);
End loop;
If an AND-split relationship covers the arrows in the set \( \{ <ND, Cnode> \} \), then
For each \( <ND, Cnode> \), do // Forward tracing
Record \( ND, Cnode \), \( <ND, Cnode> \), and the AND-split relationship;
Push(\( Cnode, STK \); // Push \( Cnode \) into the stack
\( Cnode = ND \).
Loop
If \( <Cnode, ND> \) is covered by the AND-split relationship, then
Record \( ND, Cnode \), and \( <Cnode, ND> \);
Repeat // Forward tracing ends at a leaf
If \( ND \) is a leaf, then \( Cnode = \text{Pop(STK)} \);
Else
Push(\( Cnode, STK \);
\( Cnode = ND \);
Until \( ND \) 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

After all paths are identified, the AND-join is manipulated using \( ANDjOP \) operation below.

\( ANDjOP \). Repeat
If web service \( WS \) appears in \( PTH \) times in which \( N > 1 \) then
Merge the \( N \) \( WS \) into one;
Redirect the arrows pointing to any of the original \( WS \) to the merged \( WS \);
Place an AND-join to cover the arrows pointing to the merged \( WS \);
Let arrows pointing out from the original \( WS \) be pointed out from the merged \( WS \);
Place an AND-split to cover the arrows pointing out from the merged \( WS \);
End if;
Until no duplicated web service in \( PTH \);
After the adjustment of AND-join, Eq. (1) is applied to compute the possibility of a requester to invoke a web service. Then, the successfulness possibility for every path should be computed. The possibility $P_{TH_{SUCC}}$ for the path $PTH$ can be computed using Eq. (2) below, in which web services in $PTH$ constitute the set \{\textit{ws}_i\} and $poss_i$ is the possibility for $ws_i$ obtained from Eq. (1).

$$P_{TH_{SUCC}} = \prod poss_i$$

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 $P_{THQoS_{SCV}_i}$ of the $i$th QoS criterion for the path $PTH$ is computed using Eq. (3), in which $QoS_{ws_i}$ is the $i$th QoS criterion’s value for $ws_i$ in $PTH$.

$$P_{THQoS_{SCV}_i} = \sum_{j} QoS_{ws_{ij}}, \text{if the operator is addition},$$

$$P_{THQoS_{SCV}_i} = \prod_{j} QoS_{ws_{ij}}, \text{if the operator is multiplication}.$$ (3)

To ensure the quality of a path, the $P_{THQoS_{Rule}}$ (path QoS rule) below should be fulfilled for a path.

$$P_{THQoS_{Rule}}: \forall P_{THQoS_{SCV}_i}, P_{THQoS_{SCV}_i} \geq P_{THQoS_{CL}_i}.$$ (3)

If a path $PTH$ passes the $P_{THQoS_{Rule}}$, its QoS value $P_{THQoS_{Val}}$ (path QoS value) is computed using Eq. (4), in which $P_{THQoS_{SCV}_i}$ is the $i$th QoS criterion’s value for the path and $W_{pi}$ is the weight of $P_{THQoS_{SCV}_i}$.

$$P_{THQoS_{Val}} = \sum_{i} P_{THQoS_{SCV}_i} * W_{pi}$$ (4)

After that, an overall value $P_{TH_{VAL}}$ for the path $PTH$ is computed using Eq. (5) below. Exhibition 1 summarizes the above description into an algorithm.

$$P_{TH_{VAL}} = P_{TH_{SUCC}} * w_1 + P_{THQoS_{Val}} * w_2$$ (5)

**Exhibition 1** The path composition algorithm

**Input:** $DSOL = \{ Dsoli | Dsoli is a $Dsol$ in Definition 1 \}$

RQCD = \{rqcdpwd | 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) | rqatt is an attribute of the requester, rqattval is the value of the attribute rqatt\}

\// Other input data are omitted

**Output:** $PTHS = \{pthi | pthi is a web service path to accomplish a $Dsol$ in $DSOL$\}$

**Algorithm:**

1. For each $Dsoli$ in $DSOL$, do
   1.1 Apply a selection algorithm to construct a $WSIG$ (see Definition 2) for $Dsoli$,
   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 PTHsuc
   1.9.2 For i = 1 to n, do // Suppose there are n QoS criteria
      1.9.2.1 Apply Formula 3 to compute the QoS value PTHQoScvi of the ith QoS criterion for the path
   1.9.3 If PTHQoSRule is not fulfill, delete the path and go back to step 1.9
   1.9.4 Apply Formula 4 to compute the QoS value PTHQoSVal of the path
   1.9.5 Apply Formula 5 to compute the overall value PTHVAL for the path
1.10 Select web service paths (by the requester) and place them into PTHS

4. EVALUATION

The solution “Rent a house” (see Fig. 1) is used as an example in the experiments. We selected ten students to attend the experiments. 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. To clarify the necessity of access control in path composition, we required the students to: (a) compose 20 paths by skipping the access control policy and (b) compose 20 paths without the skipping. After composition, the students executed the paths. The experiment result is shown in Fig. 5, which depicts the importance of considering access control during path composition.

![Fig. 5. The experiment data of the first experiment.](image)

To clarify how the access control policy affects the successfulness of a path, the students were required to composed only one path by applying our access control policy and algorithm. 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. The experiment result is shown in Fig. 6, which shows the following information: (a) even
every web service is honest, execution failure might still occur on the path that could be a consequence of small $k$ value in Eq. (1) and (b) more dishonest web services resulted in less successful possibilities.

5. PROBLEM DISCUSSION

Two important problems are listed below: (a) if requesters don’t know the meaning and credit level numbers, or don’t know that web services may leak information, requesters will not change the credit level numbers, and (b) if information leakage is not detected, credit level numbers will not be changed.

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 to detect information leakage and handle credit level numbers. 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. Before a precise access control standard is defined, the less precise access control standard XACML can be applied to implement our access control policy. The XACML-based environment is described below.

XACML offers mechanisms to describe access control policies for web services. Policies that can be described are limited to the static aspect. Therefore, XACML can implement our upper level access control policy. As to our lower level policy, it belongs to the dynamic aspect. XACML cannot implement the policy in the dynamic aspect. To overcome this shortcoming, XACML should be extended by adding a proxy to record web service credit level numbers and handles their increment and decrement. In the implementation, XACML should be managed by a trustable organization. We use Fig. 7 to depict an implementation of our access control policy using XACML. The upper level policy is implemented in XACML and the lower level in a proxy. Moreover, the standards XACML and Broker are managed by a trustable organization. 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.

6. CONCLUSIONS

A requester invokes web services to accomplish requirements, which can be solved by solutions. To accomplish a solution, it should be decomposed into sub-functions to apply web services. When composing web service paths to accomplish a solution, the security of web service access should be considered. 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. 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. 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. It 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 successfullness 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.

REFERENCES


Shih-Chien Chou (周世杰) is a Professor in the Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan. His research interests include software engineering, software reuse, and information flow control.

Jin-Yuan Jhu (朱暘元) received a Master degree from the Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan. He is currently served in Chinese Army.