Service Call Graph (SCG)
Information Flow Analysis in Web Service Composition

Ziyi Su[1] and Frédérique Biennier[2]
[1]*Department of Computer Science, Northeast Normal University, Changchun, China*
[2]*Lab. LIRIS CNRS, INSA-lyon, Avenue Albert Einstein, Villeurbanne, France*

Keywords: Web Service Composition, Workflow, Service Call Graph, Context Slicing, Dependency.

Abstract: This paper presents a method for analyzing Web Service-based dynamic business process, using a *business process slicing* method to capture the asset (service or information) derivation pattern, allowing to maintain providers' policies during the full lifecycle of assets in a collaborative context. Firstly, we propose a Service Call Graph (SCG) model, extending the System Dependency Graph, to describe dependencies among partners in a business process. Analysis can be done based on SCG to group partners into sub-contexts. Secondly, for analyzing SCG, we propose two slicing strategies, namely 'asset-based' and 'request-based' slicing, to deal with the scenarios of both pre-processing business process scripts and on-the-fly analyzing service compositions. Security analysis can be achieved focusing on each sub-context, by examining downstream consumers' security profiles with upstream asset providers' policies.

1 INTRODUCTION

Service-Oriented Architecture provides an environment for partners to share digital assets, including computing capability (e.g. Web Service) and information (e.g. data), in order to produce final artifacts (e.g. composed service or new information generated from data aggregation). As assets are shared beyond ownership boundary, the risk of intellectual property infringement (e.g. circumventing of trade secret, or even leakage to a competitor) associated to 'loss of governance' is a major barrier for moving toward collaborative business model (Linda et al., 2010) (Kagal and Abelson, 2010) (Daniele and Giles, 2009). Therefore security requirement is brought to an end-to-end scale, to ensure the protection of corporate patrimony value during its full lifecycle in the collaborative business process (covering the creation, dissemination, aggregation and destruction stages), paying particular attention to the way asset is used and protected by partners. Overcoming this barrier relies on a comprehensive method that can make a 'pre-decision' for selecting partners, as well as continuously regulating the partners behaviors.

In a collaborative context, such requirements involve that downstream recipients must gain assets provider's approval for re-disseminating such assets (or a new digital asset including it) (Park and Sandhu, 2002) (Bussard et al., 2010), .

In the Web-based collaborative scenario, the decision of choosing partners (and therefore, the exact paths of asset dissemination) is related to the business logic. Then from the perspective of security engineering for the collaborative context as a whole, ensuring that all the providers' security criteria are maintained during the full lifecycle of their assets is the essence. This involves analyzing the collaborative business process to track the dissemination paths of each asset, so that the co-effect of providers' policies can be calculated when assets merge and the consumers' security profiles are checked when assets are consumed.

This paper develops a method for analyzing complex collaborative context and applying fine-grained security policy to manage assets sharing activities. The basic thoughts involve that security foundation for a successful collaborative process is built when each provider's policy upon its asset is fulfilled during the whole business process.

Our solution is based on a Service Call Graph (SCG) extending the System Dependency Graph (SDG), for mining partners asset sharing relations in collaborative business processes. A data structure 'service call tuple' corresponding to the SCG is also proposed to capture dependencies among partners.

Then, we present both asset-based and request-based context slicing methods, for mining the 'assets aggregation' and 'requests aggregation' from the service call tuple list that represents a business process. Such aggregations decide the partitions of sub-contexts, where fine-grained security policies can be applied.

We analyze the sub-context developments, using 'pre-processing' and 'on-the-fly processing' strategies, and describe how down-stream provider assets security is achieved by managing sub-context developments.

Section (2) introduces the security policy model we use as foundation for security management. The Service Dependency Graph (SDG) is also introduced, based on which we develop a business process slicing method. Combined with the security policy model, fine-grained security configuration can be achieved in a Web Service composition scenario.

Section (3) proposes the SCG-based approach to describe assets aggregation patterns usually involved in a business process. Dependency between system partners, through asset sharing, is represented by the SCG and corresponding 'service call tuple'.

Section (4) introduces the business process analysis method, illustrated by the motivating use case. We propose two slicing strategies, namely 'asset-based' and 'request-based' slicing, to deal with the scenarios of both pre-processing business process scripts and on-the-fly analyzing service compositions. Security analysis can be achieved focusing on each sub-context, by examining downstream consumers' security profiles with upstream asset providers' policies.

2 BACKGROUND AND RELATED WORK

Web Service enables the openess of corporate Information System, the inter-operable interaction, agile work-flow and efficient values exchange. Such federated business paradigm brings new concerns about how to configure security among decentralized partners and how to protect resource in life-long scale. Fitting the open and collaborative Internet-based system paradigm, more adaptive attribute-based security policies (OASIS, 2005) (Su and Biennier, 2010) have been brought forward to express enriched security factors as well as consumption 'actions' upon resource. When applied to service composition scenarios, full lifecycle security for exchanged assets can be achieved with analysis of business process and adaption of security policies.

2.1 Attribute-based Security Policy Model

An attribute-based security policy has the ability to express fine-grained security factors related to system entities, through elements as Rights, Conditions and Obligations (see formula 1).

$$Assertion = (O, S, R, C, Rn, Ob, L) \qquad (1)$$

The semantics of the factors are as follows: 'O' (Object) is the resource bearing corporate asset value (service or information). 'S' (Subject) is the party that requests accessing the Right to the resource. 'R' (Right) is the Operation upon the resource that the Subject can be allowed to exercise. 'Rn' (Restriction) is the constraints upon the Right. For example a restriction 'three times' may be used to refine the right 'rendering a piece of multi-media file'. 'C' (Condition) is the requirements that must be satisfied for the Subject to access Rights upon the Object, including subject attributes (SAT), object attributes (OAT) or context related attributes ($CNAT$) – attributes of transaction context, environment, infrastructure, etc. 'Ob' (Obligations) is the action that 'must' be exercised. For example the obligation 'to delete acquired data in 10 days' can be associated to rights like 'read stock amount' and 'read client data'. 'L' (Logic Operator)represents the logic operators as 'imply'($\leftarrow$), 'and' ($\wedge$) and 'or' ($\vee$).

Such a policy model has the ability to accommodate 'point-to-point' security factors such as the the subjects and environment attributes. The 'due use' factors can also be expressed to regulate consumption actions. Nonetheless, such a security model is oriented to the one-to-one cooperation scenario. In a Web Service composition scenario, security requires that an upstream provider's policy should be met by downstream consumers that directly or indirectly receive information assets from the provider, in order to guarantee end-to-end security to assets. In such contexts the asset sharing pattern in the service composition should be analyzed.

2.2 System Dependency Graph

In a Web Service-based business federation, information assets are transferred across organization boundaries, possibly merging with other assets. In order to give a full lifecycle protection to an asset, it's necessary to capture the asset derivations. This is analogous to program slicing (GrammaTech,) (Zhao and Rinard, 2003) based on System Dependency Graph (SDG) (GrammaTech,) (Gu et al., 2008). Program slicing asks about which statements influence (backward slicing), or are influenced by (forward slicing), the current statement under exam, whereas collaborative process analysis asks about which processes (functionalities provided by a partner can be seen as a process, e.g. implemented with a Web Service) influence which processes, therefore tracing asset exchange and derivations.

We use a similar approach to 'slice' a collaborative context into sub-contexts. Each sub-context confines a scope of partners interrelated by assets exchanges (in other words, partners in different sub-context don't exchange asset, although they are in the same collaborative process). We firstly give an overview of the sub-context modes we may encounter when analyzing a collaborative context, before introducing the analysis method.

In the following sections, we discuss the analysis of a collaborative context of Web Service composition, guided by a use case, in order to introduce the basic thoughts of our analysis method.

3 SERVICE CALL GRAPH-BASED SERVICE COMPOSITION REPRESENTATION

We use a simple use case (see figure 1) of Web Service composition to facilitate our discussions.

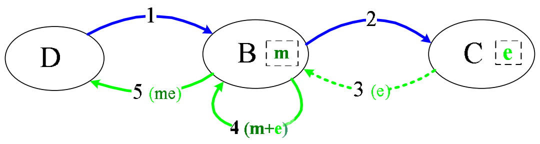

Figure 1: Service composition represented with SCG.

Use Case 1. An assurance enterprise 'Deirect assure'(D) consults 'medical information' (m) from 'Bonetat clinique'(B). Part of the information, 'cardiac exam' (e), is taken from a medical examination laboratory 'Cardis health'(C). The business process includes the following steps:

(1) D contacts B, requiring m;
(2) B contacts C, requiring e, in order to reply D;
(3) C sends e to B;
(4) B merges e with m; if success,
(5) B answers to D.

As B and C are asset (information) providers in this use case, a full lifecycle security management means that their policies should be respected during the whole lifecycle of their assets. This involves answering two questions:

(1) By which partners will an asset be accessed? A question of this type for use case 1 is "The 'Cardiac exam info' provided by C will be accessed by B or D, or both of them?"

(2) Which assets will a party be given access to? In use case 1 a question of this type is "Will D access the assets provided by B and C, either directly or indirectly?".

While both of these questions can be answered intuitionally for use case 1, the pondering procedure reflects the goal and method of context slicing. Question (1) is related to QoP aggregation among partners. Question (2) is linked to RoP aggregation. The goal is to enable down-stream asset security (Bussard et al., 2010), so that consumers should comply with the policies of the O-Assets (assets provided by partners) involved in the C-Asset (the artifacts of collaborative work, which aggregates several O-Assets) they want to access. Our method is based on a Service Call Graph resulting from the modification and extension of System Dependency Graph (SDG) (GrammaTech,) (Gu et al., 2008).

3.1 Service Call Graph

A participant in a collaborative contexts is analogous to a procedure in SDG: it receives calling information and yields results. We use $P_i \xleftarrow{c} P_j$ to denote that a party P_i depends on another party P_j with 'control dependency': whether P_i will be activated or not depends on P_j. We use $P_i \xleftarrow{d} P_j$ to denote that a party P_i depends on another party P_j with 'data dependency': data provided by P_j are involved in data produced by P_i. We propose a data structure 'Service Call Graph' (SCG) based on extensions of SDG to represent partners interactions in the collaboration context. These extensions can be illustrated with use case 1 (figure 1 is a SCG of use case 1):

- The first extension in our SCG model is that data dependency can belong to two types:
  - an ***aggregation dependency*** means that P_i involves data of P_j (the same as SDG);
  - a ***non-aggregation dependency*** denotes that data produced by P_i do not involve data from P_j (an extension of SDG).

  For example, in the SCG presented in figure (1), the blue edges (step 1 and 2) represent control dependency. The green edges (steps 3, 4 and 5) represent data dependency. Besides, the solid green lines (edge 4 and 5) mean that the output data (responses) ***include*** information from the input data (aggregation dependency). The dashed green line (edge 3) means that the output data ***do not*** include information from the input data (non-aggregation dependency).

- The second extension is that the ***assets*** carried by the message exchanges are attached directly to the edges in SCG (see edges 3, 4 and 5 in figure 1).

Furthermore, to capture assets derivation pattern, the ***indirect dependency*** relation should be retrieved, based on partner service calls in a business collaboration: $\forall P_i, P_j, P_k, \forall \alpha \in \{c, d\}$ where P_i, P_j and P_k are partners in a collaboration, c and d are *control dependency* and *data dependency* relations respectively, then P_i is ***indirectly dependent*** on P_k if $P_i \xleftarrow{\alpha} P_j \wedge P_j \xleftarrow{\alpha} P_k$. There are two types of indirect dependency.

- *Indirect* ***data*** *dependency* is the situation where each relation in a dependency chain is *data dependency*. We sum it up as an axiom:

  Axiom 1 (Indirect Data Dependency)**.**
  $\forall P_i, P_j, P_k$: $P_i \xleftarrow{d} P_j \wedge P_j \xleftarrow{d} P_k \Rightarrow P_i \xleftarrow{d} P_k$
  For example, in use case 1, whether D gets the results or not depends on the response of B. B's response in turn depends on response from C.

- *Indirect* ***control*** *dependency* is the situation where (one or more) *control dependency* relations exist in the dependency chain:

  Axiom 2 (Indirect Control Dependency)**.**
  $\forall P_i, P_j, P_k, \forall \alpha \in \{c, d\}$: $(P_i \xleftarrow{c} P_j \wedge P_j \xleftarrow{\alpha} P_k) \vee (P_i \xleftarrow{\alpha} P_j \wedge P_j \xleftarrow{c} P_k) \Rightarrow P_i \xleftarrow{c} P_k$
  As an example for indirect control dependency, in use case 1, whether C will be called or not depends on B. In turn, whether B will be called or not depends on D. So C is indirectly 'control dependent' on D.

We can see the slight difference between axiom 1 and axiom 2: Data dependency is transitive only when the edges in the dependency chain are all associated to 'data dependency', whereas when control dependency exists in a dependency chain, it propagates 'control dependency' to the chain.

When analyzing complex business process, e.g. those defined using WS-BPEL, one must consider the impact of 'variables', which are used to carry information inside the process. As information carried by variables are eventually exchanged between partners, the information exchanges between variables (e.g. through 'value assignment') also lead to assets derivation.

These variables can be complex data type (e.g. defined by XML schema). In this case, if a part of one variable is valued-assigned to a part of another variable (see the 'sample process' in WS-BPEL specification (OASIS, 2007)), the later variable is 'data dependent' on the former one. Thus we have the following axiom:

Axiom 3 (Direct Data Dependency between Variables)**.**
$\forall V_i.c_m, V_j.c_n$, $c_m \xleftarrow{d} c_n \Rightarrow V_i \xleftarrow{d} V_j$
where P_i and P_j stand for variables. c_m is a component of (a part of) P_i. c_n is a component of P_j.

This axiom describes the situation that, as in WS-BPEL a "variable" can have plural components, each of them a container that can be assigned value, the value exchange between components of two "variables" incurs data dependency between the two "variables". There are only data dependency relations between variables, as the only form of interactions between variable is data exchange. Therefore the conditions leading to indirect data dependency between variables can be described by axiom 1. In the following discussion about dependency relation, we don't need to differentiate variables from partners (i.e. 'partnerLink' in WS-BPEL), as we can see that dependency relations for partners and for variables can be described by the same set of axioms.

3.2 Service Call Tuple

We use a tuple $< P_i \overset{t}{\longleftrightarrow} P_j, \Delta >$ to denote the service call from P_i to P_j, Δ being the exchanged asset. We can have the following basic types of service call tuple:

- $< P_i \xrightarrow{c} P_j >$ denotes that P_i calls P_j with a message carrying no asset.
- $< P_i \xleftarrow{c} P_j >$ denotes that P_i receives a message from P_j that carries no asset.
  An example scenario including these two types of service call is when a mail agent queries a mail service for whether a mail is sent or not, and receives confirmation from the server. In such case the calling message and the response message are deemed as not carrying any asset (i.e. information needing protection). We can see that whether a message carries asset or not depends on the straining criteria of security in a specific application context.
- $< P_i \xrightarrow{d} P_j, \Delta_i >$ denotes that P_i calls P_j, by sending asset Δ_i.
- $< P_i \xleftarrow{d} P_j, \Delta_o >$ denotes that P_i receives a response from P_j that carries asset Δ_o.
- $< P_i \overset{\alpha}{\longleftrightarrow} P_j, \Delta_i, \Delta_o >$ denotes that P_i calls P_j, sending asset Δ_i, receiving response carrying asset Δ_o, where Δ_o includes information from Δ_i.
- $< P_i \overset{\alpha}{\longleftrightarrow} P_j, \Delta_i, \Delta_o, \not\subset >$ denotes that P_i calls P_j, sending asset Δ_i and receives a response carrying asset Δ_o, where Δ_o does not include information from Δ_i.

- $< P_i \xleftrightarrow{f} P_j, \not\subset >$ denotes that the interaction between P_i and P_j is failed, due to negative result of policy negotiation.

These tuples represent the edges of SCG. We can see that asset exchanges (and aggregations) occur with service calls.

3.3 Assets Aggregation

Usually, assets derivations (basically, either 'merging' or 'splitting') occur with partners' interactions. Therefore, recognizing assets derivation relations involves firstly formalizing partner interactions with service call tuples. Then the service call tuples list can be analyzed to track the asset merging or splitting activities. There are three situations that may incur such activities:

- If X sends information containing asset value to Y, who aggregates it with its own information (expressed as Y calling itself) and further sends it to Z. In this situation, we can identify the following service call tuple sequence:

$$\begin{aligned} &< X \xrightarrow{d} Y, \Delta_X > \\ &< Y \xleftrightarrow{d} Y, \Delta_X, \Delta_Y > \\ &< Y \xrightarrow{d} Z, \Delta_Y > \end{aligned} \tag{2}$$

- If X sends information within its request to Y and gets response(s) from Y that includes X's information. This situation is represented by the following service call tuple:

$$< X \xleftrightarrow{d} Y, \Delta_X, \Delta_Y > \tag{3}$$

  Extra attentions should be paid in this case, as we can not be sure that the response message includes information from the request message. Whether the output (responses) from a partner integrates the input (request) or not depends on the business logic of this partner's system. An example of such a case is when X sends some personal information to Y to calculate the insurance premium. If the response from Y consists in the insurance premium and the person's information, there is an assets derivation, otherwise if Y answers with only the insurance premium, there is no assets derivation. The decision of which information includes 'asset value' and should be protected is closely related to the application domains. In any case, we need to know relations between inputs and outputs to conclude whether assets derivation exists during a direct interaction or not. This can be done by analyzing partner's service functional description, e.g. WSDL in a Web Service context. It can also be done at the business process level, by adding extra indicators to a WS-BPEL script. In the modeling level, we use the following notations to define whether the partner response includes information from a request or not:

  – Most of the time, request information (or part of it) is included in the response, therefore we use the default tuple to represent it:

$$< Y \xleftrightarrow{d} Y, \Delta_i, \Delta_o > \tag{4}$$

  – whereas $\not\subset$ is used to indicate that no information of the request is included in the response:

$$< Y \xleftrightarrow{d} Y, \Delta_i, \Delta_o, \not\subset > \tag{5}$$

- If X fetches (expressed by '$\xrightarrow{c}$', as there is no asset value in the request) information from Y and aggregates its own information with it. We get the following tuples:

$$\begin{aligned} &< X \xrightarrow{c} Y > \\ &< X \xleftarrow{d} Y, \Delta_Y > \\ &< X \xleftrightarrow{d} X, \Delta_Y, \Delta_X > \end{aligned} \tag{6}$$

As an example, we build the list of service call tuples for use case 1 (See formula 7, where the tuples in the list are indexed by the steps of business process):

$$\begin{aligned} &< \tau 1, D \xrightarrow{c} B > \\ &< \tau 2, B \xrightarrow{c} C > \\ &< \tau 3, B \xleftarrow{d} C, (e) > \\ &< \tau 4, B \xleftrightarrow{d} B, (e), (me) > \\ &< \tau 5, D \xleftarrow{d} B, (me) > \end{aligned} \tag{7}$$

The assets derivation relations between partners are equivalent to data dependency relations between them. Therefore assets derivation trail, which decides the sub-context pattern, can be mined from the list of service call tuples.

4 SUB-CONTEXT SLICING

Like the information *reachability* questions in SDG, the assets derivation trail can be tracked by scanning the service call tuples list, paying particular attention to asset aggregation. Based on this, providers' policies upon assets can be maintained during assets derivations. This involves firstly allocating corelated assets in the same sub-contexts.

We use a data structure 'context development tuple' $< N_C, V_C, P_C, L_A, L_P, \tau >$ to record the information of sub-context development, where:

- N_C is the name of the sub-context,
- V_C its version,
- P_C its parent sub-context,
- L_A a list of all the asset involved in the sub-context,
- L_P the collection of policies in the sub-context,
- τ the step of business process.

This tuple is built by the sub-context slicing process which scans the SCG (e.g. service call tuple list) according to two strategies: asset-based slicing and request-based slicing.

4.1 Asset-based Slicing

The asset-based slicing method focuses on capturing the aggregation relation among assets. Using this method, a sub-context is created when the first O-Asset is launched into the collaborative context by the owner. When a new partner joins the context with a new O-Asset, the sub-context consisting of the existing asset is updated, if the new partner's O-Asset is merged with the existing C-Asset. Otherwise (i.e. the new partner's O-Asset is not merged with existing C-Asset), a new sub-context is created. In use case 1, the list of sub-context tuples is as following:

$$\begin{aligned} &< R_{CB}, 1, (\phi), (e), (RoP_C), \tau 3 > \\ &< R_{CB}, 2, (R_{CB}.1), (e,m), (RoP_C, RoP_B), \tau 4 > \\ &< R_{CB}, 3, (R_{CB}.2), (e,m), (RoP_C, RoP_B), \tau 5 > \end{aligned} \tag{8}$$

We can see that in step 3 (represented by $\tau 3$), the first sub-context is created, including the asset e and its related RoP_C. We name the context after the interaction leading to the creation of it, e.g. R_{CB} ('resource' sent from C to B). Its version is '1'. It has no parent context (ϕ). Then in step 4, as a new asset m merges with e, the sub-context $R_{CB}.1$ is updated to $R_{CB}.2$. In step 5, it remains unchanged.

This list describes the evolution of the sub-contexts. There is only one sub-context for use case 1, which can be represented with an assets derivation diagram (see figure 2).

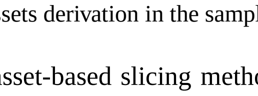

Figure 2: Assets derivation in the sample use case.

Using the asset-based slicing method, policy negotiation and aggregation (including conflicts detection) can not be done until the first asset is launched into the context (step 4 of use case 1). If there is a conflict, steps $\tau 2$ and $\tau 3$ are actually wastes of partners' resources and don't need to be proceeded. Therefore the asset-based slicing method should be used for pre-processing a business process script (e.g. WS-BPEL documents) before it is executed. To analyze a collaborative context on-the-fly, we need a request-based slicing method.

4.2 Request-based Slicing

The request-based slicing method creates a sub-context when the first request is made. Then, when a new partner joins the business process, either its QoP can be aggregated into an existing sub-context, or it will lead to the creation of a new sub-context. The decision is also straight forward: the QoPs of two partners should be aggregated, if they will access the same asset in future steps of the collaboration context. By this method, we get the following list of sub-context tuples for use case 1:

$$\begin{aligned} &< Q_{DB}, 1, (\phi), (QoP_D), \tau 1 > \\ &< Q_{DB}, 2, (Q_{DB}.1), (QoP_D, QoP_B), \tau 2 > \end{aligned} \tag{9}$$

This tuple list captures QoP aggregations. When the first request is made by D in step 1, a sub-context is created, including the QoP of D. We name the context after the interaction leading to the creation of it, e.g. Q_{DB} ('query' sent from D to B). In step 2, as B is requesting assets from C 'on behalf of' D, QoP_B and QoP_C are aggregated. Therefore sub-context $Q_{DB}.1$ is updated to $Q_{DB}.2$

However, deciding who will access the same asset is more tricky than it may firstly look like, especially when partners work asynchronously, (e.g. if after a partner X receiving a request from partner Y, another partner Z also sends request to X, before X responds to Y). We provide basic protocols for dealing with such cases:

- **Protocol 1.** *After X receiving a request from Y, all requests X sends to other partners are deemed as being* ***on behalf of*** *Y, until X responds to Y, or X receives a request from another partner Z.* This involves that a request from Y to X establishes an 'on behalf of' relation. Consequently, the QoP_Y should be aggregated into QoP_X for all the requests X sends after receiving the request from Y, until that X gets the result and responds to Y. The 'on behalf of' relation between X and Y ends when X responds to Y. It also can, however, be interrupted before that X responds to Y. The following two protocols regulate such cases.
- **Protocol 2.** *An 'X on behalf of Y' relation is interrupted by another 'X on behalf of Z' relation if*

Z makes a request after that X receives a request from Y and before that X responds to Y.

- **Protocol 3.** *An 'X on behalf of Y' relation interrupted by another request from Z can be resumed after X responding to Z, if X receives a response from a partner P, who was called by X 'on behalf of Y'.* This means that the 'on behalf of' relation can be **nested**. For example, with the following request-response sequence (i.e. service call tuple list in formula 10), we can say that the 'on behalf of' relation between X and Y is restored after X responding to Z (step 5), by the interaction where 'P responds to X', as P is a partner that X has requested on behalf of Y.

$$\begin{aligned}
&< \tau 1, Y \xrightarrow{c} X, \Delta_{i1} > \\
&< \tau 2, X \xrightarrow{c} P, \Delta_{i2} > \\
&< \tau 3, Z \xrightarrow{c} X, \Delta_{i3} > \\
&< \tau 4, X \overset{d}{\longleftrightarrow} Q, \Delta_{i4}, \Delta_{o4} > \qquad (10) \\
&< \tau 5, Z \xleftarrow{d} X, \Delta_{o3} > \\
&< \tau 6, X \xleftarrow{d} P, \Delta_{o2} > \\
&< \tau 7, Y \xleftarrow{d} X, \Delta_{o1} >
\end{aligned}$$

These are basic protocols because they handle the primary cases in service composition. When dealing with real-world complex business federations, more information concerning the business process and partner functionalities should be taken into consideration. Nevertheless this basic reasoning process remains in accordance with those described in these protocols.

In the following we discuss the employment of asset-based and request-based methods for context slicing. For this, we firstly give an overview of sub-context developments that can occur in a collaborative business process.

4.3 Context Development

During each step (partner interaction) of the business process, different types of sub-context development are caused by the partners service calls:

- **Create.** The creation of a new sub-context is always based on an independent QoP or RoP from a partner. In other words, if the partner provides an asset which is not aggregated with other assets in the *current step*, a new sub-context consisting of this asset and the corresponding RoP is created. Analogously, if the partner is calling others ***on its own behalf*** (i.e. not because it is doing so for responding to another partner) a new sub-context consisting of its QoP should be created.
- **Update.** On the contrary, updating an existing sub-context happens if the partner's asset has data dependency (according to discussions in section 3.3) with the assets belonging to an existing sub-context, or if this partners' assets are merged with existing assets. It also happens when the partner is requesting assets on behalf of another 'former' requestor, that is, it's QoP and the QoP of the former requestor should be 'transmitted' to the requested party. Therefore the QoPs are in the same sub-context.
- **Merge.** Merging sub-contexts is a special kind of update operation. It happens when two existing assets in two sub-contexts merge, or when the request sent by a partner is on behalf of several former requestors from different sub-contexts. In the later case, the different sub-contexts are corelated by the asset value in the responding message.
- **Split.** While 'splitting' a sub-context, several new sub-contexts are created. They all 'inherit' the assets and policies of the previous context. Context splitting can be caused by three types of interactions:
  - a party sends copies of the same asset to several partners and the copies are developed differently;
  - a party sends copies of the same request to several partners at the same time;
  - the business process has a control structure defining parallel activities.
- **End.** Ending a sub-context occurs when it is *merged*, *split* or when the whole *business process ends*.

These sub-context developments occur as asset sharing relations change, hence the context analysis is proceeded according to business process logic. In the Web Service contexts, one need to consider both scenarios of service orchestration guided by WS-BPEL and on-the-fly service compositions. We propose both a pre-processing method and an on-the-fly processing method for these scenarios.

4.4 Pre-processing and On-the-Fly Processing

In context slicing, *pre-processing* refers to the circumstances where a pre-defined business process (e.g. WS-BPEL script) is analyzed before the execution, to see whether it can be carried out or not, w.r.t. partners' security profile-request satisfiability. This can be done with the asset-based slicing method, using the policies and attributes of partners.

On the contrary, for *on-the-fly* processing, partners' RoPs and QoPs must be aggregated as soon as they join the collaboration context, in order to find out security policy conflicts more timingly. This requires using both asset-based and request-based slicing methods.

In our use case 1, on-the-fly slicing strategy first builds the QoP tuples (see formula 9) from the beginning of the business process, using request-based slicing. Then from step '$\tau 3$', RoP tuples are built (see formula 8), using asset-based slicing.

The RoP aggregation relations and QoP aggregation relations are used to generate the security polices and profiles of each sub-context. When a new partner joins the collaboration context, it is allocated to a sub-context according to whether it's an asset provider or consumer (or both). Its policy and profile are aggregated to the security policy and profile of that sub-context.

5 CONCLUSIONS AND FUTURE WORK

This paper develops a method for analyzing information assets sharing patterns in Web Service composition scenarios, therefore security configuration can be done in a fine-grained manner and ensure the overall security level in inter-enterprise level business federation. We introduce a 'Service Call Graph (SCG)' and a corresponding data structure 'service call tuple' extending the System Dependency Graph (SDG), to capture asset aggregation (and derivation) in a collaborative business process. A 'context slicing' operation can be made based on the SCG, to categorize partners that have direct and indirect assets exchange relations to the same 'sub-contexts'. Security policy negotiation and aggregation in the scope of each sub-context can ensure the full lifecycle security for assets. A detailed discussion has been given on the rational of our method, facilitated by a sample use case. Basically, 'data dependency' between partners incurs assets (and RoP policies) aggregation, whereas 'control dependency' between partners leads to the 'on behalf of' relation and QoP aggregation. According to data dependency, 'asset-based' slicing is sufficient for pre-processing a business process script (e.g. WS-BPEL script). Nevertheless our on-the-fly processing strategy applied to a business federation (e.g. dynamic service composition) requires both 'request-based' (due to control dependency) and 'asset-based' slicing. Future work involves the construction of security management paradigm with assets tagging systems and the context slicing engine for both asset-based and request-based analysis.

ACKNOWLEDGEMENTS

This work is supported by "the Fundamental Research Funds for the Central Universities" under grant Number 12QNJJ025.

REFERENCES

Bussard, L., Neven, G., and Preiss, F.-S. (2010). Downstream usage control. In *Proceedings of the 11th IEEE International Symposium on Policies for Distributed Systems and Networks*, POLICY '10, pages 22–29, Washington, DC, USA. IEEE Computer Society.

Daniele, C. and Giles, H. (2009). Cloud Computing: Benefits, risks and recommendations for information security. *Technical report, European Network and Information Security Agency (ENISA)*.

GrammaTech. Dependence graphs and program slicing-codesurfer technology overview. *Technical report, GrammaTech, Inc*.

Gu, L., Ding, X., Deng, R. H., Xie, B., and Mei, H. (2008). Remote attestation on program execution. In *STC*, pages 11–20.

Kagal, L. and Abelson, H. (2010). Access control is an inadequate framework for privacy protection. In *W3C Privacy Workshop*. W3C.

Linda, B. B., Richard, C., Kristin, L., Ric, T., and Mark, E. (2010). The evolving role of IT managers and CIOs–findings from the 2010 IBM global IT risk study. Technical report, IBM.

OASIS (2005). eXtensible Access Control Markup Language (XACML) version 2.0. http://docs.oasis-open.org/xacml/2.0/.

OASIS (2007). Web services Business Process Execution Language (WS-BPEL). http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html.

Park, J. and Sandhu, R. (2002). Originator control in usage control. In *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02)*, POLICY '02, pages 60–, Washington, DC, USA. IEEE Computer Society.

Su, Z. and Biennier, F. (2010). End-to-end security policy description and management for collaborative system. In *Sixth International Conference on Information Assurance and Security*, IAS 2010, pages 137 – 142.

Zhao, J. and Rinard, M. (2003). System dependence graph construction for aspect-oriented programs. Technical Report MIT-LCS-TR-891, Laboratory for Computer Science.MIT.