Model-Checking of Safety and Security Aspects in Web Service Flows

Shin Nakajima

National Institute of Informatics
and
PRESTO, Japan Science and Technology Agency
nkjm@nii.ac.jp

Abstract. Web service flow is essentially a description of a distributed collaboration system, in which more than one Web service providers participate. The flow should have safety properties such as deadlock freedom and application specific progress properties. At the same time, the flow should satisfy some security properties since it is executed in an open network environment. This paper introduces an idea of a lattice-based security label into BPEL, a Web flow description language being standardized, in order to detect potential insecure information leakage. It further discusses that both the safety and security aspects can be analyzed in a single framework using the model-checking verification techniques.

Keywords: Web Service Flow, Information Flow, Lattice-based Security Label, Model-Checking

1 Introduction

Web service is widely accepted as a new technology in business network environment in which each participant acts as a service provider [4]. And as a robust framework to compose lots of the service, Web service flow description languages are proposed [3][11][17] to express that Web service providers are combined to show collaborative behavior. Since the flow is executed in an open network environment such as the Internet, both safety and security are the two aspects of particular interest in the Web service framework.

Safety requires that the Web service flow, as a distributed collaboration description, is logically and functionally correct. The flow should be free from deadlock and satisfy some application specific progress properties [12]. And the model-checking verification techniques are shown successful for the automatic analysis of the safety aspect of Web service flows [8][13][14][15].

Security is one of the major concerns in the Web service framework. It needs some high-level security policy given as a non-functional property [1] and refers to various things depending on which layer in the Web service technology stack we are considering [4]. WS-Security deals with the secure end-to-end communication of the SOAP messaging, and WS-Authorization is proposed to be a standard

N. Koch, P. Fraternali, and M. Wirsing (Eds.): ICWE 2004, LNCS 3140, pp. 488–501, 2004.

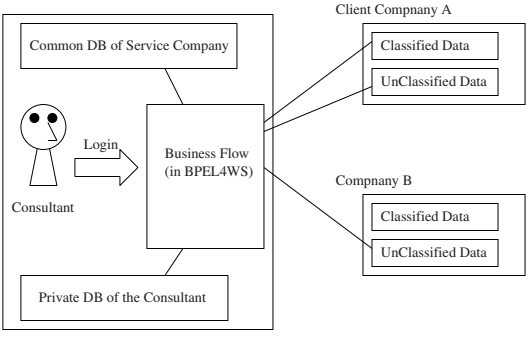

Fig. 1. Out-sourcing Web Service

service for the access control. Another important area regarding to security is the non-interference, which requires that information having different security levels does not interfere with each other. It is concerned with a problem of the information flow control to prevent the potential information leakage.

The problem of the information flow control has been a primary interest in the computer security. High-level access control policies and their enforcement have been proposed [1]. Especially, a lattice-based control model [7] provides a basis for various high-level access control policies including the information flow control [16].

Since the Web service framework is gaining a wide acceptance as a key infrastructure in the networked information systems, high-level security policy of the non-interference will become an important issue. For example, a Web-based service company, getting a contract of *out-sourcing* service from a client, implements its business logic in terms of Web service flows and accesses to classified documents in the client company. The business logic description is required to show that the classified information is never leaked at all. It is desirable to have a method to formally analyze whether the Web service flow is secure in view of the non-interference.

This paper proposes to use the model-checking verification technique [5] to analyze Web service flow descriptions in view of both safety and security. A main contribution of the paper is to show that BPEL [3] descriptions, with a small extension using the lattice-based security label [7], can be analyzed in terms of the non-interference.

2 Security Concerns in Web Service

2.1 Information Leakage in Web Service Flows

Figure 1 shows an example Web service scenario motivating our work. The service company gets a kind of *out-sourcing* contract from the client company A, and does some tasks previously done in the company A internally. In order that

the service company works for the client, it should make a free access to classified documents as well as unclassified ones. When the whole scenario is implemented in the Web service framework, the security becomes a major concern since the classified information goes out of the company A via the open network such as the Internet.

The service company may use Web service flow languages such as BPEL [3] to implement its service business logic. In order for the BPEL description to access the information, the client A should establish a Web service server to provide necessary information for the service company.

In view of the client company A, its classified information, manipulated by the BPEL business logic, should never be leaked at all to, for example, the company B. The client company A may require the service company to show a formal certification that the confidentiality is satisfied. The classified information never interferes with less confidential data while the BPEL description works on the information.

Since Web service has its basis on the Internet, security is one of the major concerns in the technology standardization [4]. WS-Security deals with the secure end-to-end communication of the SOAP messaging service. It employs cryptography to make messages encrypted and to provide means for the sender signature. A. Gordon et al [9] employs the spi calculus for the verification of the security aspect of combining multiple communication protocols, each being implemented on top of SOAP. WS-Authorization is proposed to be a standard service for the access control of the Web service. The proposal includes a method for describing access control policies. It, however, does not deal with the non-interference problem arising from multiple accesses. In summary, the problem of controlling information leakage in the Web service flow has not been discussed in details so far in the Web service framework.

2.2 BPEL

As a standard language for describing Web service flows that compose more than one Web services, BPEL4WS (Business Process Execution Language for Web Service), or BPEL for short, was proposed. BPEL v1.0 was made public in July 2002 as a new language to supersede both WSFL [11] and XLANG [17]. And BPEL v1.1 [3] is considered in this paper.

BPEL is a behavioral extension of WSDL (Web Service Description Language) [2]. WSDL is basically an interface description language for Web service providers, which contains information enough for the clients to access. The client invokes a Web service provider using WSDL. The invocation is *one-shot*, which means that WSDL does not describe global states of the provider.

On the other hand, BPEL is a language for expressing behavioral compositions of Web service providers. It can express *a causal relationship* between multiple invocations by means of control and data flow links. BPEL employs a distributed concurrent computation model with variables. This paper mostly concerns with the distributed concurrent language aspect of BPEL.

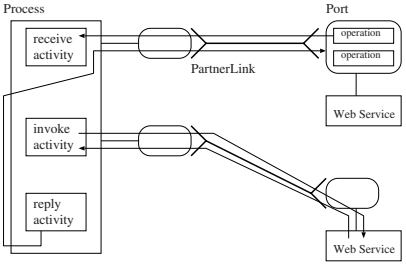

Fig. 2. Example Primary Entities in BPEL

Figure 2 illustrates some of primary language elements in BPEL. A main construct of the Web service flow is `Process`, which is a net-based concurrent description connecting more than one `Activity`'s with control links. Some of the primitive `Activity`'s, in turn, is a place that sends/receives messages to/from external Web service providers. Each Web service provider can be seen as a `Port` instance of a particular `Port Type`, which has appropriate WSDL description as its sub-elements. And `Partner Link` specifies which `Activity` is linked to a particular Web service provider of the `Port`.

BPEL provides a variety of primitive `Activity`'s. Some of them are shown in the example in Figure 2. The `receive` activity waits for invocation requests from the outside, the `invoke` activity initiates an execution request on the Web service provider and receives result values, and the `reply` returns some value to the original outside initiator as the result of the computation of the `Process`.

In addition to the above primitive `Activity`'s, BPEL provides an `assign` activity for accessing variables. It also has other activities concerning to implement control flows such as `sequence` (sequential executions), `switch` (branch on conditions), `while` (repetitions), and `flow` (concurrency). The `flow` activity corresponds to a flow graph that can represent concurrency. The flow graph consists of the `Activity`'s as nodes and `Link`'s as edges representing control links.

Last, BPEL introduces a lexical context with `scope` activity. The lexical context defines an effective scope of variables and various handlers such as exception. Further, `scope` activity can have a `serializable` attribute, which specifies multiple concurrent accesses to the variables inside the scope are serialized.

3 Lattice-Based Access Control

3.1 Basic Model

The problem of the information flow control is a primary interest in the computer security. The access control deals with the problem of deciding whether a particular user (Principal) can have an access to a particular resource (Target). Namely, each access can be checked in an individual manner. We can say, for example, that a principal P1 is permitted to have a read access to a target T1 while its write access to T2 is inhibited.

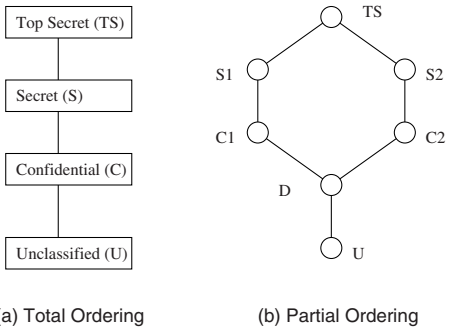

Fig. 3. Lattice of Security Labels

The above basic access control alone, however, is not able to exclude invalid data-flows arising from a series of read and write accesses. In some cases, each individual access is permitted, but a resultant net data-flow potentially violates some global requirement. This is a problem of the information flow control [1].

Here is a simple example scenario regarding to the information flow control. A principal P1 writes to a target T2 the data read from another target T1. Further, another principal P2 reads from T2 and writes to T3. We assume here that all the four individual accesses are permitted, and also that we have a high-level security policy saying that T1 and T3 should be of non-interference with each other. The policy means, in particular, that the data originally stored in T1 should not be flowed into T3. However, a series of the accesses results in a global data-flow from T1 to T3 since each access is allowed.

In order to remedy the situation, a lattice-based approach to the information flow control has been proposed [7]. The idea is that we assign security labels to the principal as well as the target, and define a partial ordering between the labels. The label of the principal is called Security Clearance, and Security Classification for the case of the target.

The ordering $\succ$ reflects the intension of how one is more important than the other in view of the secrecy. We can write as below to represent that a security label SL_i is more secure than SL_j.

$$SL_i \succ SL_j$$

We assume that the set of security labels is finite and the ordering $\succ$ forms a finite lattice [6]. Figure 3 illustrates two example lattice of security labels. Figure 3 (a) is a simple case where four labels are arranged in a total order. Top Secret is more secure than Secret (TS $\succ$ S). Figure 3 (b) illustrates another example in which security labels form a lattice with partial orderings. C1 and C2 are more secure than D (C1 $\succ$ D $\wedge$ C2 $\succ$ D), but they are incomparable with each other. TS is a maximal of C1 and C2.

Further, we introduce a constraint relationship *dominates* defined as

$$SL_i \textit{ dominates } SL_j \stackrel{\wedge}{=} SL_i \stackrel{*}{\succ} SL_j$$

For example, a document with a security classification of *secret* (S) dominates *unclassified* (U) documents. A senior member having a *secret* (S) security clearance can have a right access to both *secret* (S) and *unclassified* (U) documents. But a junior staff member with a *unclassified* (U) security clearance can only read *unclassified* (U) documents.

With the lattice-based approach, we can solve the example problematic situation mentioned above. Below L(X) refers to the security label attached to X, where X refers to either a principal or a target. First, we assume that the high-level security policy requires

not (L(T3) dominates L(T1)) .

If all the four accesses are allowed, we have the following four constraints satisfied.

L(P1) dominates L(T1) ... [A1]
L(T2) dominates L(P1) ... [A2]
L(P2) dominates L(T2) ... [A3]
L(T3) dominates L(P2) ... [A4]

By means of the transitivity of the dominates relationship, the four relations results in

L(T3) dominates L(T1),

which is in contradiction with the required security policy, and thus the access violation can be detected.

3.2 Declassification

The lattice-based method is a promising approach for the information flow control. However, the basic model in Section 3.1 is not practical. Its idea is basically to allow information flow only from low to high, which can exclude invalid flows resulting in a leakage of secure information.

This, however, causes a problematic situation. Once a principal with the highest security clearance reads some data, the data cannot be accessed or written to a target resource with a lower security classification. The information is swallowed into a *black-hole*. The lattice-based control method can be a basis for the confidentiality, but not adequate in view of the availability.

As a remedy for the availability problem, the idea of declassification is proposed [1]. In short, the basic lattice-based model employs the static security label only. On the other hand, the declassification model uses the security labels determined dynamically at runtime in checking the dominates relationships.

The following example considers a case where a principal P1 reads data from a target T1 and then writes it to another target T3. We further assume that the following three constraint relationships are satisfied.

L(P1) dominates L(T1) ... [B1]
L(P1) dominates L(T3) ... [B2]
L(T3) dominates L(T1) ... [B3]

In particular the third relationship ([B3]) indicates that the security classification of T3 is higher than that of T1. Although [B3] shows that the flow from T1 to T3 is possible, the flow is not allowed according to the basic model. The reason comes in order.

The first relationship ([B1]) allows P1 reading data from T1. When P1 writes the data to T3, the second one ([B2]) is checked, which specifies that P1 is not allowed to write data to T3. Therefore, the flow from T1 to T3 is inhibited.

The declassification model relaxes the strict application of the static security labels in checking constraints. The model introduces a notion of DCL (DeClassified Label). We assume here that a trusted principal writes data to a target with a security classification that the principal dominates. The basic model disallows the write because the principal is more secure than the target. In the new model, we choose an appropriate DCL value from the underlying lattice, and use it in place of the label of the principal so that the write access is allowed. The value of DCL is the one dominating the target that the principal has a read access, and at the same time the one that the written target dominates.

For example, in the above case, we introduce the following three relationships in order to decide the value of DCL.

L(P1) dominates DCL ... [C1]
DCL dominates L(T1) ... [C2]
L(T3) dominates DCL ... [C3]

The first relationship ([C1]) ensures that DCL is lower than P1, which is a basic global constraint on DCL. The second one ([C2]) shows that the flow from T1 to DCL is allowed, which is in accordance with the condition on P1 and T1 ([B1]). And the last one ([C3]) allows the flow from DCL to T3, which together with [C2], results in a global flow from T1 to T3 as requested. If the appropriate DCL does exist, then we can use the value in place of the security clearance of the principal for the checking process.

3.3 Global Analysis

According to the lattice-based access control model, each constraint rule is obtained at each execution point. In the first example in Section 3.1, the rule ([B1]) is obtained at the point where the principal P1 has a read access to the target T1. However, in order to check whether the information flow is possible or not, all the rules along (potential) execution paths should be collected. It is because the net information flow is possible only if all the rules along the execution path are satisfied. Therefore, a global flow analysis is needed for collecting rules.

The global flow analysis method is basically a data-flow analysis algorithm. For the case of the basic lattice-based model in Section 3.1, the label value of the

security clearance of a principal moves downward as *a data token* in execution paths. And when an execution path reaches a point where the principal has either a read or write access to a target, the label value is checked against the *dominates* relationship at the point. The analysis should be done not for a single execution path, but for all the potential execution paths.

The analysis method becomes complicated when we employ the declassification model. As seen in Section 3.2, we have to show that an appropriate DCL value exists in the underlying lattice structure, which requires to solve a set of *dominates* constraint relationships. It, however, the method can be simplified.

Our simplified method is a minor variant of the above analysis for the basic model. As the label value flowing downward, we use an *initial* guess of DCL instead of the value of the principal. And at each execution point accessing a target TA, the value is updated as a *maximal* of its security label and the old DCL value. The initial value of DCL is set to be L(T0) which is the security classification label of the target T0, while T0 is what the principal has a read access firstly in the execution path.

For an illustration of the method, we use here a simple example. The scenario consists of a series of accesses, a read to T1, a read to T2, and a write to T3. We have three relationships [D1] to [D3] for the accesses. And we assume [D4] between the two targets, T1 and T2, that accept read accesses from the principal.

DCL_1 dominates L(T1) ... [D1]
DCL_2 dominates L(T2) ... [D2]
L(T3) dominates DCL_2 ... [D3]
L(T1) dominates L(T2) ... [D4]

At the first read access ([D1]), DCL_1 is set to L(T1). At the second, our method sets DCL_2 to be a maximal of DCL_1 and L(T2), actually a maximal of L(T1) and L(T2). And in this example DCL_2 becomes L(T1) because of the rule [D4].

On the contrary, if we override DCL_2 at [D2] to be L(T2), we cannot detect a potential security violation in a case where the following relationships are also assumed.

L(T3) dominates L(T2) ... [D5]
L(T1) dominates L(T3) ... [D6]

Because of the rule [D6], the net flow is not secure. However, the rule [D3] allows such a flow because of the rule [D5] in the case where DCL_2 is L(T2).

In concluding this section, we have to point out that devising a precise analysis method needs the semantics of the language we are considering. In this paper, we have to consider BPEL semantics in details.

4 BPEL with Security Label

This section discusses how we introduce the notion of the security label into BPEL. Our information flow control method uses the lattice-based approach with the declassification.

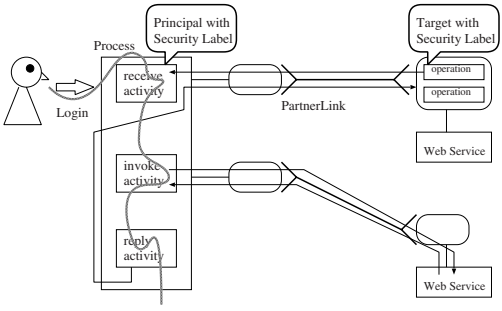

Fig. 4. BPEL with Security Label

We consider here an extension to BPEL. The extension includes (a) introducing the principal with security clearance, and (b) identifying `Port` with the target having security classification. Figure 4 schematically shows how we attach the security label information to the BPEL language elements.

First, we consider how to introduce the idea of principal to BPEL descriptions. Generally a principal represents a client user of the system, and its security clearance level is assigned in the authentication process of *login*. However, since more than one ways to implement the login process are possible, we cannot a priori decide how the security clearance is obtained. Therefore, we only introduce a new reserved variable `Principal`, and use the `assign` activity to set the value.

```
<assign>
   <copy>
     <from variable='...' part='...' />
     <to variable='Principal' property='Security Clearance' />
   </copy>
</assign>
```

`Principal` is a single-assignment variable, and read-only throughout the process execution afterward.

Second, we identify a Web service provider with a target having security classification. The Web service provider can be considered as a persistent data storage. BPEL allows to exchange information with the outside by means of messages through the `Link` to/from the `Port` (see Figure 4). And thus we identify `Port` with the target.

An alternative approach is to regard variables appeared in `assign` activity as the target because a variable is a place-holder for values. However, since the variables in `assign` activity are confined in the enclosing scope, the value is forgotten once the execution control is exited from the scope without any significant side-effects. We have not taken this approach.

Next, we need some XML representation for expressing the high-level security policy requirements using the dominates relationship. Each requirement would

be either positive or negative. The whole security policy is a collection of such constraint relationships.

```
<securityPolicy name='orderingServicePolicy'>
  <admit>
     <dominates high='T2' low='T1' />
  </admit>
  <inhibit>
     <dominates high='T3' low='T1' />
  </inhibit>
</securityPolicy>
```

Last, the security lattice is defined in terms of a collection of successor ($\succ$) relationships. Below is a partial definition of the example lattice in Figure 3 (b).

```
<lattice name='systemB'>
   <succ high='TS' low='S1' />
   <succ high='TS' low='S2' />
   <succ high='S1' low='C1' />
   ...
</lattice>
```

5 Model-Checking of Extended BPEL

5.1 Overview

We propose to use the model-checking verification technique [5] to analyze our extended BPEL descriptions from the viewpoint of the lattice-based information flow control. It extends the work on model-checking WSFL descriptions for the analysis of the safety aspect [13] to include the security labels.

As seen in Section 3.3, the analysis basically needs to explore all the potential execution paths of a given extended BPEL description. We have to devise an analysis method that is faithful to the operational semantics of BPEL, which has concurrency as its core language primitives. Since global flow analysis of concurrent systems becomes complicated, we will not implement an algorithm from scratch, but devise a way of using an off-the-shelf model-checker for the analysis. It is because the model-checking technique has been successful for the verification and analysis of concurrent systems.

We use SPIN model-checker [10] for the analysis engine. It is because SPIN is a quite engineered tool that can handle a large state space efficiently. SPIN also provides various language primitives that are useful for the verification and analysis of practical software. In Section 5.2 we will consider how we encode BPEL language primitives in Promela, the input specification language of SPIN. At the same time, we will discuss why SPIN is useful for our purpose, namely as a prototyping engine for the analysis of the information flow control.

Roughly a BPEL process description is translated into a Promela process. However, it is not enough just to have a Promela process for the BPEL to be

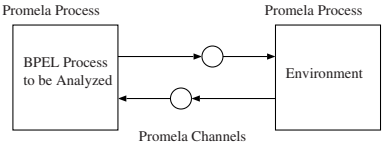

Fig. 5. Closed System

analyzed. The environment that communicate with the current BPEL, namely all the service providers that have interaction with the BPEL, should be explicitly represented. Figure 5 illustrates the situation where the environment is also modeled as a Promela process to have a *closed* system. And the communication is done by sending and receiving messages via Promela channels.

5.2 Encoding BPEL in Promela

As explained in Section 2.2, BPEL has a variety of language constructs to represent various activities. What to do is to give semantics to each activity in view of the security label analysis.

First we consider how to encode the control aspect of BPEL descriptions since the control flow becomes a basis for constructing the state space to be explored. Since each activity has different semantics in regard to establishing control flows, the control activity should be considered individually.

The `sequence` activity (sequential execution) is implemented by a Promela sequencing separator (`;`), and the `switch` (branch on conditions) turns out to be a Promela conditional statement (`if ... fi`). The `while` (repetition) is encoded with a Promela repetition statement (`do ... od`) with an appropriate loop condition checking as below.

```
do
  :: (<condition>) -> ...
  :: else -> break
od
```

Second, the `assign` activity is used for dealing with variable accesses. It establishes the flow `from` to `to` variables, and becomes a basis for constructing data-flows. The translation to Promela is mostly straight forward because we can use Promela variables.

Third, the `flow` activity, among others, provides concurrency in the activity execution in a single process description. The idea comes from the net-oriented concurrent language of WSFL [11]. Figure 6 illustrates an example `flow` activity. The example `flow` activity itself is enclosed in the top level `sequence`, which specifies a sequencing of three activities, `receive`, `flow`, and `reply`. The dotted line is meant to represent sequential execution in the diagram. The `flow` has three concurrently executing `sequence`'s, each having multiple primitive activities. The diagram also shows two solid curves, one from `invoke` to `invoke`

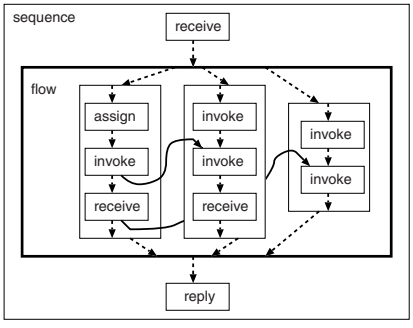

Fig. 6. Flow Activity Example

and another from `receive` to `invoke`. The curves add further control flows to synchronize execution of concurrently executing activities.

As discussed elsewhere on the model-checking of WSFL [13][14], we use Promela process to represent activities executing concurrently. In the example shown in Figure 6, we have three Promela processes as a translation of the `flow` activity with three enclosing `sequence`'s. Further, we use Promela variable to implement the synchronization in accordance with the semantics of the superimposed control flow shown as the solid curves in the diagram. We also deal with the DPE (dead-path elimination) feature relating to the concurrent `flow` activity. How we encode the DPE in Promela is reported in the previous work [14].

Last, the primitive activity (`invoke`, `receive`, or `reply`) is the point of the interaction with the target. In which direction the information is flowing, namely read or write, is our main concern.

For example, an `invoke` activity establishes an out-going information flow (write) to an external Web service whose `PortType` is `shippingService`.

```
<invoke partnerLink='customer'
        portType='shippingServicePT'
        operation='shippingNotice'
        inputVariable='shipNotice'>
```

And, in view of the security labels, a dominates relationship should satisfy in order to make the flow possible:

L(shippingServicePT/shippingNotice) dominates L(Principal).

Since the above `invoke` activity allows a flow from the current BPEL process, the channel send communication carries the security label of Principal.

```
cout!ShippingNotice(Principal);
```

Then the environment Promela process, playing a role of `shippingService`, receives the message and checks if the dominates relationships hold.

```
do
  :: cin?ShippingNotice(X) -> assert(dominates(self,X))
  ...
od
```

where the variable `X` carries the security label of Principal, and `self` refers to that of the `shippingService`.

It should be paid a special attention to how we implement the function `dominates` in Promela. The function uses the information equivalent to the given lattice. And it decides whether the dominates relationship holds between two input parameters, which will require a search in the lattice structure. Note that the function `dominates` is purely functional, having no-side effects.

If, in a simple-minded, we encode the function `dominates` in Promela directly, it will involve a lot of data-centric computation leading to a large state-space exploration, which is less efficient. Our approach is to implement the function `dominates` in C language by making use of the SPIN feature of the embedded C code [10]. From Promela descriptions, `dominates` is treated as a primitive function not increasing the size of the state space to be explored in the model-checking process.

6 Discussion and Conclusion

We have proposed to use the model-checking verification technique to analyze BPEL descriptions in view of both safety and security. In particular, our main interest in the security aspect is non-interference. Our proposal includes a small extension to the current BPEL language specification as well as the introduction of the lattice-based security label.

Although the security label used in the examples in this paper is simple, it is possible to encode other security policy for a variety of purposes. Actually R. Sandhu [16] discusses that various high-level security policies can be represented by means of the lattice-based control model.

Some literatures mention to use model-checkers in the analysis of the Web service flows. H.Foster et al [8] uses the LTSA model checker for the analysis of safety property of BPEL. S.Nakajima [13][14] uses the SPIN model-checker for the safety analysis of WSFL. S.Narayanan et al [15] uses a Petri Net formalism to provide decision procedures for the analysis of the Web service written in DAML. All these work concerns with the safety properties only. Using the model-checker in the analysis of the security property in the Web service flow is new.

Although the idea of the lattice-based approach to the information flow control is not new [1][7][16], the integration with the Web service technology, BPEL in particular, is new. Since it needs a fine-grained control and data flow analysis of BPEL descriptions, the analysis of the non-interference is a problem quite dependent on the BPEL semantics. It is tightly coupled with the Web service flow language specification. We cannot have a separate service component for such a purpose although the Web service framework currently seeks for independent components for various domains such as WS-Authorization or WS-Security [4].

Last, up to the time of writing this paper, we cover a core part of BPEL language only. BPEL is a large language that has many interesting features such as compensation, fault, and event handlers. To cover these features is left for future work.

References

1. M.A. Bishop. *Computer Security: Art and Science*. Addison-Wesley 2003.
2. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Service Description Language (WSDL). W3C Web Site, 2001.
3. F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business Process Execution Language for Web Services. Version 1.1, May 2003.
4. F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana. The Next Step in Web Services. *Comm. ACM*, Vol. 46, No. 10, pages 29–34, October 2003.
5. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, 1999.
6. B. Davey and H. Priestley. *Introduction to Lattices and Order (2ed.)*. Cambridge, 2002.
7. D.E. Denning. A Lattice Model of Secure Information Flow. *Comm. ACM*, Vol.19, No.5, pages 236–243, May 1976.
8. H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based Verification of Web Service Compositions. In *Proc. ASE 2003*, September 2003.
9. A. Gordon, K. Bhargavan, and C. Fournet. A Semantics for Web Services Authentication. In *Proc. POPL 2004*, pages 198–209, January 2004.
10. G.J. Holzmann. *The SPIN Model Checker*. Addison-Wesley 2004.
11. F. Leymann. Web Services Flow Language (WSFL 1.0). IBM Corporation, May 2001.
12. S. Nakajima. On Verifying Web Service Flows. In *Proc. SAINT 2002 Workshop*, pages 223–224, January 2002.
13. S. Nakajima. Verification of Web Service Flows with Model-Checking Techniques. In *Proc. Cyber World 2002*, pages 378–385, IEEE, November 2002.
14. S. Nakajima. Model-Checking of Web Service Flow (in Japanese). In *Trans. IPS Japan*, Vol.44, No.3, pages 942–952, March 2003. A concise version presented at OOPSLA 2002 Workshop on Object-Oriented Web Service, November 2002.
15. S. Narayanan and S.A. Mcllraith. Simulation, Verification and Automated Composition of Web Services. In *Proc. WWW-11*, 2002.
16. R. Sandhu. Lattice-Based Access Control Models. IEEE Computer, Vol.26, No.11, pages 9–19, November 1993.
17. S. Thatte. XLANG – Web Services for Business Process Design. Microsoft Corporation, May 2001.