Specification and Verification of Authorization Policies for Web Services Composition

Mohsen Rouached and Claude Godart

LORIA-INRIA-UMR 7503
BP 239, F-54506 Vandœuvre-les-Nancy Cedex, France
{mohsen.rouached,claude.godart}@loria.fr

Abstract. The management and maintenance of a large number of Web services is not easy and, in particular, needs appropriate authorization policies to be defined so as to realize reliable and secure Web Services. The required authorization policies can be quite complex, resulting in unintended conflicts, which could result in information leaks or prevent access to information needed. This paper proposes a logic based approach using for specifying authorization policies in Web services composition.

1 Introduction

Service Oriented Architecture allows for considerably more complex interaction models than the classical client/server model, including symmetric peer-to-peer interactions where both parties want to check authorizations, or multi-party composed services where authorizations are an issue for each component service. Therefore, an appropriate authorization framework is needed to smooth the flow of a transaction between multiple services whilst respecting the privacy of the data used. This is a complex task since each individual service may have its own authorization requirements. The traditional authorization service is not appropriate in this kind of interactions where a coordinating service would need to exchange policy and credential information as well as managing the operation details. Managing these authorization exchanges can lead to processing bottlenecks within the service as well as privacy concerns given that the coordinating service retains visibility and control. That is why a balanced approach to security is needed, taking into account not only security considerations at the level of the infrastructure, but also requirements that follow from business policies and processes (composition of Web services). There are several unique security-related characteristics that need to be addressed to develop secure business processes with Web services, including authorization capabilities, authorization models, authorization may require various levels of scope, and authorizations may be dynamically (re-)allocated to subjects.

Our objective is to support compositions of Web services taking into account the authorization requirements of each Web service provider and composing only those that are compatible regarding these requirements.

The rest of the paper is structured as follows. In Section 2, we present how we specify the authorization policies using the event calculus logic (EC). Section 3 concludes and outlines future work.

2 Formalizing Authorization for Composite Web Services

In the context of Web services a service is seen as a resource that is provided within the system, to which access is controlled. A service can also request other services and is actively involved in computation. In our formal policy model, a Web service can therefore be seen as both object (s_{targ}) and subject (s_{src}). The type of request made to the Web service is modelled as an action.

2.1 Authorization Model

To allow the necessary level of control over the behaviour of the Web service composition authorization policies should be defined in a language flexible enough to allow the specification of conditions that can include multiple triggering events that may take place over time. The EC language seems to be the best basis to start from. We adapt a simple classical logic form of the EC [1], whose ontology consists of (i) a set of time-points,(ii) a set of time-varying properties called fluents, (iii) a set of event types (or actions). The logic is correspondingly sorted, and includes the predicates $Happens$, $Initiates$, $Terminates$ and $HoldsAt$, as well as some auxiliary predicates defined in terms of these. $Happens(a, t)$ indicates that event (or action) a actually occurs at time-point t. $Initiates(a, f, t)$ (resp. $Terminates(a, f, t)$) means that if event a were to occur at t it would cause fluent f to be $true$ (resp. $false$) immediately afterwards. $HoldsAt(f, t)$ indicates that fluent f is true at t.

To achieve a complete specification that supports formal reasoning in EC, the following elements must be represented in the model.

- Functions that can be used as parameters in the basic predicate symbols of EC. We define these functions as events that may occur during the composition execution. Below, the introduced events are explained. In these formulas, V_p represents the set of parameters values for the operations supported by services.
  - $Op(s, Action(V_p))$: used to denote the operations specified in a policy function or event (see below).
  - $requestAction(s_{src}, Op(s_{targ}, Action(V_p)))$: represents the event that occurs whenever a service source attempts to perform an operation on a target service.
  - $doAction(s_{src}, Op(s_{targ}, Action(V_p)))$: represents the event of the action specified in the operation term being performed by the service s_{src} for the service s_{targ}.
  - $rejectAction(s_{src}, Op(s_{targ}, Action(V_p)))$: the event that occurs after the enforcement decision to reject the request by a particular source service to perform an action is taken.
  - $permit(s_{src}, Op(s_{targ}, Action(V_p)))$: represents the permission granted to a source service to perform the action defined in the operation on the target service.
  - $deny(s_{src}, Op(s_{targ}, Action(V_p)))$: used to denote that the source service, s_{src}, is denied permission to perform that action on the target service s_{targ}.

– We need to add specific predicate symbols. Indeed, in our case many of the function definitions above contain the tuple $(s_{src}, Op(s_{targ}, Action(V_p))$. To check if the members of this tuple are consistent with the specification of the Web service composition, we define the $isValidComp$ predicate. As such it must be used in any rule where functions with the tuple $(s_{src}, Op(s_{targ}, Action(V_p))$ are involved.

Then, the complete authorization enforcement model is illustrated in Figure 1. As shown, once the service source makes a request to perform an action on the service target, the target service's access controller processes it. To do this, the access controller evaluates the request by referring to the policy repository and the access control model. If the action is permitted, the access control model will proceed to do the requested action. Otherwise, if the action should be denied, the access control system will reject the action. We precise that the scheme is symmetric, i.e each of the two services could be target, source, or target and source at the same time.

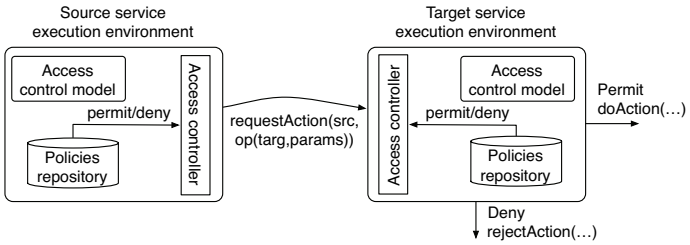

Fig. 1. Authorization Enforcement Model

As shown in Figure 1, we distinguish two scenarios to represent the enforcement model. The first scenario models the behaviour of the target service's access controller, generating a *doAction* event when an action is permitted. This event would trigger the relevant service behaviour rules thus causing the composition state to change according to the specification. The second one models a target service's access control monitor rejecting the action to prevent a denied operation from being performed.

2.2 Authorization specification

In order to correctly interact with the enforcement model described above, each policy specification rule should initiate the appropriate policy function symbol (permit,deny) for each of the events. So for example, a positive authorization policy rule should specify that the fluent $permit(s_{src}, Op(s_{targ}, Action(V_p)))$ holds when the event $requestAction(s_{src}, Op(s_{targ}, Action(V_p)))$ occurs and the constraints that control the applicability of the policy hold. Additionally, the fluent $permit(s_{src}, Op(s_{targ}, Action(V_p)))$ should cease to hold once the action has been performed thus making it possible to re-evaluate the policy rule on subsequent requests to perform the action. The EC representation of this functionality is

indicated in the $autho^{+}$ specification in Figure 2. It also shows how each of the other policy types would be represented by rules in the formal notation. For each rule, the *Constraint* predicate is introduced to specify the pre- and post-conditions for each operation. It can be represented by a combination of *HoldsAt* terms.

The $autho^{-}$ specification represents a negative authorization policy by stating that, if the *Constraint* holds and the event requesting the action happens, the action is denied. It is specified by the $autho^{-}$ part of Figure 2.

$(autho^{+})$ $isValidComp(s_{src}, Op(s_{trg}, Action(V_p))) \wedge Constraint \Longrightarrow$
Initiates$(requestAction(s_{src}, Op(s_{trg}, Action(V_p))), permit(s_{src}, Op(s_{trg}, Action(V_p))), t)$
$(autho^{+})$ $isValidComp(s_{src}, Op(s_{trg}, Action(V_p))) \Longrightarrow$
Terminates$(doAction(s_{src}, Op(s_{trg}, Action(V_p))), permit(s_{src}, Op(s_{trg}, Action(V_p))), t)$

$(autho^{-})$ $isValidComp(s_{src}, Op(s_{targ}, Action(V_p))) \wedge Constraint \Longrightarrow$
Initiates$(requestAction(s_{src}, Op(s_{trg}, Action(V_p))), deny(s_{src}, Op(s_{trg}, Action(V_p))), t)$
$(autho^{-})$ $isValidComp(s_{src}, Op(s_{trg}, Action(V_p))) \Longrightarrow$
Terminates$(rejectAction(s_{src}, Op(s_{trg}, Action(V_p))), deny(s_{src}, Op(s_{trg}, Action(V_p))), t)$

Fig. 2. Authorization Specification.

The second part of the rule shows how the *deny* fluent will be terminated once the decision to reject that action has been taken, thus allowing the specification to be re-evaluated on subsequent requests. Note that the termination parts for these policies do not have any constraints and can be generically specified for the whole service composition.

3 Conclusion

In this paper, we presented a framework for managing authorization policies for Web service compositions. More specifically, we have described the use of the EC logic for developing a language that supports specification and analysis of authorization policies for Web service composition.

There are several directions for future work to further improve the presented work. One thread in our future work will focus on the policies refinement and the generalisation of the reasoning technique to handle other security properties. Another alternative considers the integration of our model into software development process in practical settings [2].

References

1. R. Kowalski and M. J. Sergot. A logic-based calculus of events. *New generation Computing 4(1)*, pages 67–95, 1986.
2. M. Siponen, R. Baskerville, and T. Kuivalainen. Integrating security into agile development methods. *hicss*, 07:185a, 2005.