Trust-Serv: A Lightweight Trust Negotiation Service

Halvard Skogsrud[1], Boualem Benatallah[1], Fabio Casati[2], Manh Q. Dinh[1]

[1] University of New South Wales
Sydney NSW 2052, Australia
{halvards,boualem,mdinh}@cse.unsw.edu.au

[2] Hewlett-Packard Laboratories
Palo Alto, CA, 94304 USA
fabio.casati@hp.com

1 Introduction

In Web service environments, scalable access control methods are required, as requester populations are often large and dynamic. For this reason, requester identities are often not known in advance, and traditional access control models that rely on identity to determine access do not fit. Other models require requesters to submit *credentials* (i.e., signed assertions describing attributes of the owner) along with service invocations. These models often do not consider credentials to be resources, an assumption that does not hold when credentials may contain sensitive information. *Trust negotiation* addresses these problems by granting access based on the level of trust established in a negotiation between the requester and the provider. During this trust negotiation, credentials are exchanged to gradually build trust.

However, several issues need to be addressed before trust negotiation can become a viable technology. The description of trust negotiation *policies* is mainly characterized by the use of ad hoc methods, such as editing XML-based policy documents. These policies specify which resources (i.e., credentials and service operations) that can be disclosed at a given state of the trust negotiation, and the conditions to disclose them. Since trust negotiation policies may be quite complex, largely ad hoc development can be time-consuming and error prone. Incorrectly specified policies may cause, e.g., confidential information to be revealed to unauthorized parties, and open avenues for loss of data through malicious or ignorant clients or partners.

Another interesting problem that has not been addressed before is the *lifecycle management* of trust negotiation policies [4]. Enterprise policies often change for a variety of reasons, including emerging competitors, new products, mergers and acquisitions, updated business processes, and changes to laws and regulations. For trust negotiation to become a viable security solution for enterprises, there is a need for high-level frameworks and tools to provide support for automating the development, enforcement, and evolution of trust negotiation policies.

Motivated by these concerns, we have developed the Trust-Serv platform for model-driven trust negotiation in Web service environments [4]. One innovative feature of Trust-Serv is the state machine-based model for the specification of trust negotiation policies, which is translated into structures used to automatically control trust negotiations at run-time. Another innovative features is the support for *dynamic policy evolution*, that is, changes to a policy while there are ongoing trust negotiations.

2 Trust-Serv Design Overview

In Trust-Serv, trust negotiation policies are expressed as state machines [4]. States represent the level of trust achieved by the requester so far in the negotiation. By entering a new state, a requester is given access to the resources mapped to that state. However, instead of associating resources directly to states, we use the abstraction of *roles* [1]. Roles are semantic abstractions that describe some function performed by people or processes (e.g., *developer* and *tester*). Trust-Serv maps resources to roles and roles to states. Roles are cumulative, so a requester may be a member of several roles.

Transitions are extended beyond traditional state machines to capture security abstractions necessary for trust negotiation by labeling them with *authorization abstractions*. Authorization abstractions define the conditions that must be met for transitions to be fired. There are two types of authorization abstractions; *explicit* and *implicit*. Explicit transitions are triggered by actions performed by the requester. One type of explicit transition is *credential disclosure*. This type of transition requires the other party to disclose one or more credentials, and it may place constraints

Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004

on acceptable values of attributes contained within the credentials. Other explicit transitions include *provisions* and *obligations*. Provisions require actions to be taken before proceeding, while obligations require actions to be taken in the future. We represent provisions as service operations that must be invoked before the trust negotiation can proceed. Obligations are represented as service operations that require the requester to digitally sign a message stating what service operation to invoke, the deadline, and any compensation required if the obligation is not met. Implicit transitions are triggered by the provider. *Timed transitions* are implicit, and they are fired when timeout events occur, usually because the requester did not perform any action for a period of time.

Once trust negotiation policies have been specified, Trust-Serv will take care of enforcing them on behalf of the users, relieving developers from the need of implementing such logic into the service provider's code, as described below.

Enterprise policies often change to adhere to changing business strategies. Lifecycle management of policies is therefore an important issue to be considered in trust negotiation systems. Since trust negotiations may be long-lived, it may be necessary to change a policy while there are ongoing trust negotiations based on this policy. It may not always be possible to let these ongoing negotiations finish according to the old policy, especially if it was found to be in breach of laws and regulations, or if it did not properly protect business assets. In these situations, it is necessary to appropriately *migrate* the trust negotiation instances to the new policy.

Trust-Serv supports several *migration strategies* to handle migration of trust negotiation instances from an old policy to a new policy, details of which may be found in [4]. The use of these strategies is controlled by a *strategy selection policy*, which is a meta-policy specified separately from the trust negotiation policies. This meta-policy is a collection of rules, such as "Requesters who have visited state B are migrated to the new policy". When a trust negotiation policy is updated, a strategy selection policy is defined. Ongoing trust negotiations are then evaluated against this meta-policy, and the outcome of this evaluation determines which strategy to apply to each trust negotiation instance and how to migrate it to follow an appropriate policy.

3 Implementation

The Trust-Serv platform provides environments where (i) service developers may create and manage trust negotiation policies for their Web services, and (ii) both providers and requesters may observe the negotiations, and participate by negotiating manually, if desired. Trust-Serv is implemented as an extension to the Self-Serv platform [3]. Self-Serv supports Web service development based on established standards such as SOAP, WSDL (Web Service Description Language), and UDDI (Universal Description, Discovery, and Integration) [2]. An overview of this architecture is shown in Figure 1. Trust-Serv is implemented in C# using Microsoft Visual Studio .NET 2003, and implementations of security standards are provided by the Web Service Enhancements (WSE) for Microsoft .NET.

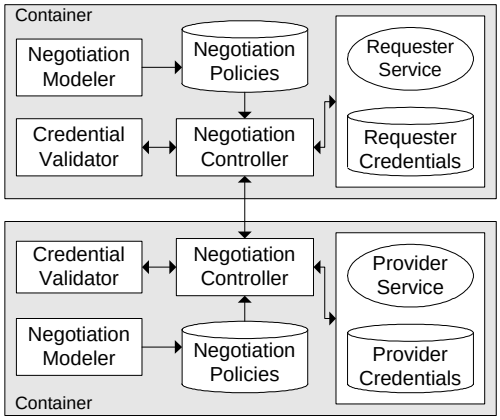

Figure 1: Architecture of Trust-Serv.

The Trust-Serv architecture introduces the notion of Web service *containers*. A container acts as a middleware layer that takes on the chore of enforcing trust negotiation policies, avoiding the need to code this logic into the Web service itself. This means that tasks such as controlling negotiations and verifying credentials are delegated to the container, thereby considerable simplifying service development. The three main components of the Trust-Serv container are the *controller*, the *modeler*, and the *validator*.

The Trust-Serv container features a *trust negotiation controller*, which is implemented as a Web service that provides the capabilities to participate in trust negotiations. The controller is responsible for receiving messages such as service operation invocations and credential disclosures, determining if new trust negotiation instances should be created, and triggering transitions if their conditions are met. Messages are sent between trust negotiation instances and service instances as SOAP request and response messages.

The *trust negotiation modeler* facilitates the specification and management of trust negotiation policies. Trust negotiation policies are edited through a visual interface, as shown in Figure 2. The modeler is also used to automatically generate *control rules*, which are used by the controller to determine actions to be taken in response to events. The control rules are generated from the state machine-based specification of the trust negotiation policy using algorithms presented in [4], and they are represented as XML documents.

The *credential validator* performs the task of validating credentials. Validation includes checking the

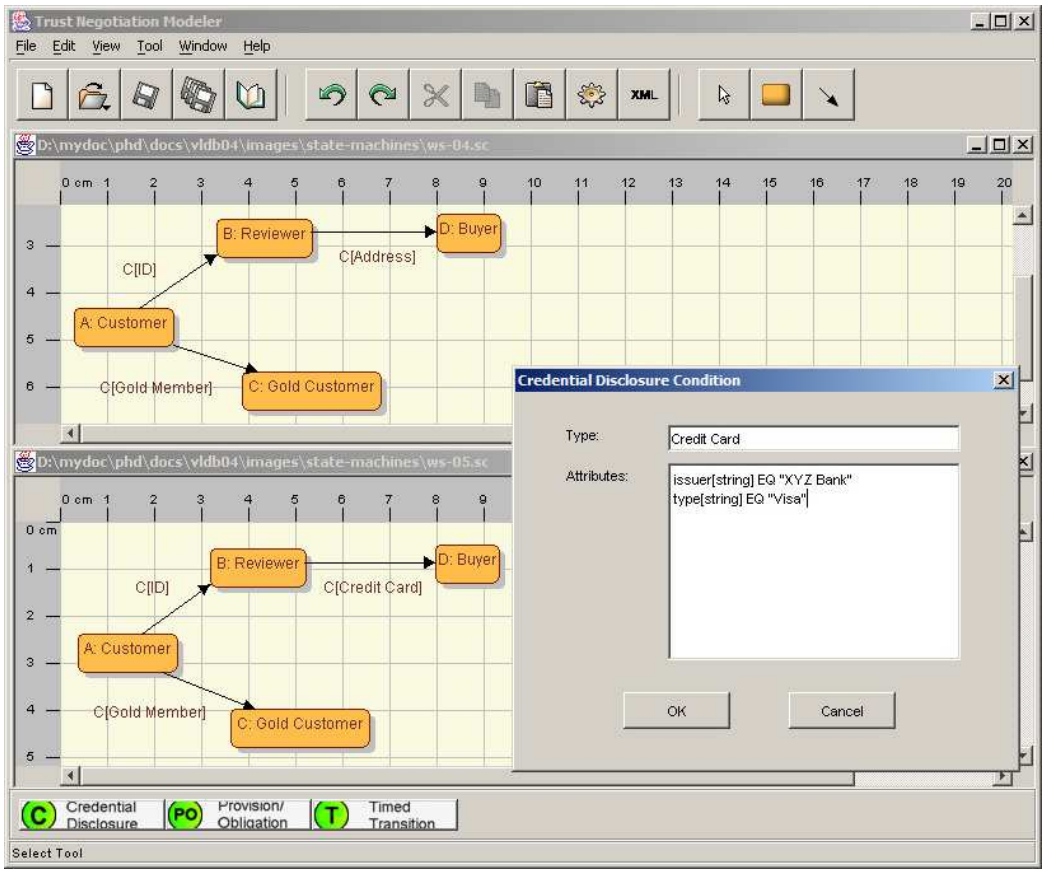

Figure 2: Defining and managing policies in Trust-Serv.

expiry date, verifying the issuer's signature, and ensuring that the credential has not been revoked. Rather than implementing this functionality in the container, we use Security Token Services (STS) as defined in the WS-Trust specification (www-106.ibm.com/developerworks/library/ws-trust). The STS is a trust service that can be used to outsource the complexity of public key infrastructure. The credential validator communicates credentials to the STS, which performs the validation and returns the result.

4 Demo Scenario

A bookshop scenario has been developed using the Trust-Serv platform. The scenario involves an online bookshop and a requester, Alice. The scenario works as follows: Alice invokes an operation of the bookshop service to buy a book. The trust negotiation controller of the bookshop service intercepts the request and initiates a trust negotiation with Alice's trust negotiation controller. Later, the provider deploys a new policy, and the trust negotiation with Alice is migrated to this new policy. We will demonstrate (i) how to define a trust negotiation policy for the bookshop service, (ii) how to perform trust negotiations both automatically and manually, and (iii) how to migrate ongoing trust negotiations to new policies using strategies.

Defining a trust negotiation policy. Firstly, we will demonstrate how to create a trust negotiation policy using the modeler. We will define a policy for the provider service, as shown in Figure 2. However, before the roles may be mapped to the states, the roles must first be created. We will show how this is achieved in the trust negotiation modeler. We will also show how credentials are mapped to roles. By default, the credentials are stored in plain XML files, so there is no need to deploy a DBMS purely for the purpose of supporting trust negotiation. However, support for a DBMS may be configured if desired.

Defining transition conditions based on our authorization abstractions are achieved in separate windows, as shown in the lower right-hand corner of Figure 2. Once the definition of the policy is complete, we will show how it is converted into control rules and how the control rules are deployed to the provider's container.

Performing trust negotiations. Once the provider's container is ready, we will deploy a requester service called Alice. Alice has her own Web service with its own trust negotiation policy and container, and her service will attempt to invoke operations of the provider service.

For the purpose of the demonstration, we have developed *trust negotiation monitors*. These monitors

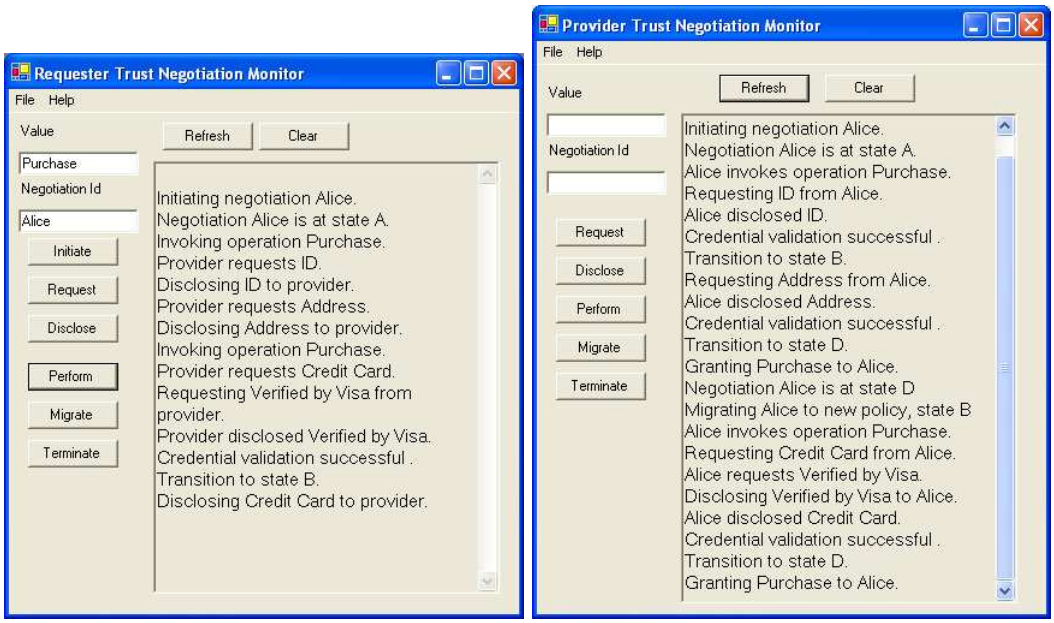

Figure 3: Alice invokes the `Purchase` operation twice, first according to policy *P.I*, then according to *P.F*. The window on the left shows Alice's trust negotiation monitor, while the window on the right is the provider's monitor. Since Alice's trust negotiation was migrated to state B in the new provider policy *P.F* after the first invocation of the `Purchase` operation, she has to disclose her `Credit Card` for the second invocation.

may be deployed by the requesters and the providers, and they show the progress of the negotiation, including all the messages sent between the controllers of the requester and the provider. In addition to monitoring the trust negotiations, the monitors also permit both the requester and the provider to negotiate manually, by invoking trust negotiation operations on each others' controllers. Figure 3 shows both the requester's monitor and the provider's monitor.

Alice starts by invoking the `Purchase` operation of the provider service. When she sends the invocation, her container deploys a trust negotiation controller instance to handle this trust negotiation. Alice's invocation is intercepted by the provider's container, which similarly spawns a controller instance to deal with Alice. The controller instances send requests and responses back and forth according to the policies of both the requester service and the provider service. In the provider's monitor, we can see when the requester service (Alice) enters a new state. Finally, Alice has negotiated sufficient trust, and her invocation of the `Purchase` operation is forwarded to the provider service, which sends its reply (Figure 3).

Migrating trust negotiations. At this point, the provider decides to change its policy from the one shown in the upper window of Figure 2 to the one in the lower window. For Alice, this requires a different credential to be disclosed to invoke the `Purchase` operation, a credential Alice has not yet disclosed. The provider also defines a strategy selection policy to appropriately migrate all existing trust negotiations. As a result of the policy update, Alice's trust negotiation instance is migrated from state D in the old policy to state B in the new policy, and her membership in the `Buyer` role is deactivated (Figure 2).

Alice then attempts to invoke the `Purchase` operation again. However, due to the migration of her trust negotiation instance at the provider, she no longer has access to this operation. Her trust negotiation then resumes, and the provider requests that she discloses her `Credit Card` to continue. Upon disclosing this credential, Alice is again able to invoke the `Purchase` operation (Figure 3).

In a more complex scenario we will deploy multiple requesters and show how a different migration strategy may be applied for each trust negotiation instance.

References

[1] D. Ferraiolo et al. Proposed NIST Standard for Role-Based Access Control. *ACM Trans. Information and System Security*, 4(3), Aug. 2001.

[2] M. P. Papazoglou and D. Georgakopoulos. Service-Oriented Computing. *Comm. ACM*, 46(10), 2003.

[3] Q. Z. Sheng et al. SELF-SERV: A Platform for Rapid Composition of Web Services in a Peer-to-Peer Environment. In *Proc. VLDB*, Aug. 2002.

[4] H. Skogsrud, B. Benatallah, and F. Casati. Trust-Serv: Model-Driven Lifecycle Management of Trust Negotiation Policies for Web Services. In *Proc. 13th World Wide Web Conf.*, May 2004.