PATTERNS FOR WEB SERVICES STANDARDS

By

Ola Ajaj

A Thesis Submitted to the Faculty of the

College of Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

Florida Atlantic University

Boca Raton, Florida

May 2010

PATTERNS FOR WEB SERVICES STANDARDS

By

Ola Ajaj

This thesis was prepared under the direction of the candidate's thesis advisor, Dr. Eduardo B. Fernandez, Department of Computer and Electrical Engineering and Computer Science, and has been approved by the members of his supervisory committee. It was submitted to the faculty of the College of Engineering and Computer Science and was accepted in partial fulfillment of the requirements for the degree of Master of Science.

SUPERVISORY COMMITTEE:

Eduardo B. Fernandez, Ph.D.
Thesis Advisor

Maria M. Larrondo Petrie, Ph.D.

Michael VanHilst, Ph.D.

Borko Furht, Ph.D.
Chair, Department of Computer & Electrical Engineering and Computer Science

Karl K. Stevens, Ph.D., P.E.
Dean, College of Engineering and Computer Science

Barry T. Rosson, Ph.D.
Dean, Graduate College

April 13, 2010
Date

ACKNOWLEDGEMENTS

I am grateful for the guidance given by Dr. Eduardo B. Fernandez and to the members of the Secure Systems Research Group for the feedback they provided in my presentation of the patterns included in this thesis. I dedicate this degree and I wish to express deep appreciation to my parents, brothers, sisters, relatives and close friends for their continuous support and encouragement throughout this masters program; without which I would be lost and so I am unconditionally appreciative. I also thank Motorola, Inc. where for supporting my studies.

ABSTRACT

Author: Ola Ajaj

Title: Patterns for Web Services Standards

Institution: Florida Atlantic University

Dissertation Advisor: Dr. Eduardo B. Fernandez

Degree: Master of Science

Year: 2010

Web services intend to provide an application integration technology that can be successfully used over the Internet in a secure, interoperable and trusted manner. Policies are high-level guidelines defining the way an institution conducts its activities. The WS-Policy standard describes how to apply policies of security definition, enforcement of access control, authentication and logging.

WS-Trust defines a security token service and a trust engine which are used by web services to authenticate other web services. Using the functions defined in WS-Trust, applications can engage in secure communication after establishing trust.

BPEL is a language for web service composition that intends to provide convenient and effective means for application integration over the Internet. We address

security considerations in BPEL and how to enforce them, as well as its interactions with other web services standards such as WS-Security and WS-Policy.

TABLE OF CONTENTS

1. INTRODUCTION

Web services intend to provide an application integration technology that can be used over the Internet in a secure, interoperable and trusted manner. Without a clear definition of how web services can manage secure communications and establish trust relationships with other partners, it would be hard to perform any kind of interaction, and malicious web services could use their business interactions to perform illegal actions.

Several committees such as W3C and OASIS are developing web services standards. Their standards are rather complex and verbose and it is not easy for designers and users to understand their key points. Descriptions of standards are given in natural language and are long documents. For example, WS-Policy is 85 pages of text and XML examples. By expressing web services security mechanisms and standards as patterns, we can verify if an existing product implementing a given security mechanism supports some specific standard [Fer06].

A pattern can abstract the main features of a standard and makes it much easier to understand. We use UML models for intuitive and precise descriptions of pattern solutions.

A product vendor can use the pattern standards to guide the development of a product. By expressing standards as patterns, we can compare them and understand them

better. Mobile devices cannot implement full web services standards. With patterns we can analyze a UML model to find which aspects can be reduced or eliminated.

The WS-Policy standard provides means of possible configurations of web services and enforces some level of pre-defined security and authentication, while The WS-Trust standard defines how to establish trust between interacting parties; we present here patterns for these standards.

The WS-Policy and WS-Trust patterns follow the POSA [Bus96] template and are similar to the style of the security patterns written in [Sch06].

As a web service composition language, BPEL can be a convenient and effective means for application integration over the Internet in typical B2B interaction scenarios. However, for BPEL to keep its promises it is necessary to provide more support for security. Vendors and companies are looking to have their requirements of authentication, integrity and confidentiality satisfied. In this thesis we will address security considerations in BPEL and how to enforce them as well as its interactions with other web services standards such as WS-Security and WS-Policy.

The goals of this research are important in the overall development and implementation of web services, and they are:

- Write patterns that describe the following standards and clarify their use:
  - WS-Policy
  - WS-Trust
- Identify relationships between dependability patterns.
- Identify security considerations for BPEL and how they are related to web services standards such as WS-Security and WS-Policy.

Chapter 2 provides necessary background information. Chapter 3 presents "AJIAD" the travel agency we use as an example in this manuscript. Chapter 4 illustrates a pattern diagram that relates the patterns of this work with previous work of the Secure System Research Group. Chapter 5 presents a pattern for the WS-Policy standard. Chapter 6 presents a pattern for the WS-Trust standard. Chapter 7 identifies security considerations for BPEL. In Chapter 8 we conclude, and highlight potential areas for future work.

2. BACKGROUND

2.1. Patterns

A pattern is an encapsulated solution to a recurrent problem in a given context and can be tailored to fit different situations. Patterns can be very useful in the development of software systems. Patterns allow for reusability, maintainability, and a systematic approach to defining and implementing mechanisms. They can serve as a guideline for application developers and help support best practices.

2.2. WS-Specifications

In order to extend web services capabilities. Microsoft, IBM and VeriSign and others have initiated web services standards, WS-*, to secure SOAP-message transmissions (handled by OASIS) [mic07].

WS-Specifications denoted by “WS-*” have been developed or are being developed. There are a variety of specifications associated with web services. These specifications are in varying degrees of maturity and are maintained or supported by various vendors and companies. These specifications may complement, overlap, and compete with each other. There is not a single managed set of specifications that is consistently used or recognized owning body across them all.

Web services standards can be lengthy documents with many details that would make it difficult for vendors to develop products and difficult for users to decide which product to use. Several organizations with different goals have also developed standards that may interfere and conflict with each others.

The solution we have to solve such confusion is by developing clear and consistent patterns that are easy to understand and use by developers to construct the required web services.

The Secure Systems Research Group (SSRG) at Florida Atlantic University (FAU) [ssr10] has made significant efforts in the direction of developing patterns for WS-Specifications and its related subgroups which has resulted in developing patterns for Digital Signature with Hashing and XML Signature [Has09a], XML Encryption [Has09b] and WS-Security [Has09c] respectively.

We are continuing developing more patterns for the rest of the specifications as needed for the purpose of having complete reference guide and catalog.

The Web Services Security specification (WS-Security) describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. This will also provides a set of mechanisms to help developers of web services secure SOAP message exchanges.

2.3. Motivation for WS-Policy

The web services policy framework defines a base set of constructs that can be used and extended by other web services specifications to describe a broad range of service requirements and capabilities.

One of the fundamental challenges that service-oriented architecture (SOA) ties to address is orchestrating a series of very dynamic interactions between disparate web services in accordance with enterprise-class standards for Quality-of-Service. WS-Policy helps to achieve this objective by facilitating agreement between producers and consumers of web services. More specifically, WS-Policy provides a means for describing and communicating the capabilities and requirements of specific web services in a coherent and reliable manner, ensuring that specific preconditions are fully met within each interaction.

As an underlying component of WS-Policy, the Web Services Policy 1.5 - Attachment specification can be used to bind specific policies to unique services via either WSDL (Web Services Definition Language) or UDDI (Universal Description, Discovery and Integration). In the case of a UDDI registry, it defines how policies can be stored and accessed within an associated repository to deliver good performance.

2.4. WS-Policy

Web services can be chaotic without a clear definition of how to use them. The concepts behind WS-Policy and related standards, such as WS-Security Policy, provide a means to specify possible configurations of a web service, and also to enforce defined security and authentication.

The web services policy framework, or WS-Policy, is a specification that allows a web service to have a set of rules (for example, required security tokens, supported encryption algorithms, and privacy rules, etc.) that must be met, or consumed. Authors of clients that consume web services are then to study the policy information to see whether or not they can comply with these policies. Thus, a client could not be written to simply access a web service that has a policy that requires all messages be encrypted or signed in a certain way. Nor could a client access a web service having a policy requiring a timestamp and send a message that does not have one. And that is the goal of WS-Policy: to specify policy information that web service consumers must comply with.

As defined in W3C, WS-Policy is a specification that declares the interaction between web services and its consumers by allowing web services to use XML to advertise their policies (on security, Quality of Service, etc.) and for web service consumers to specify their policy requirements. WS-Policy is a W3C recommendation since September 2007 [w3c07a].

Even Web Services Policy 1.5 - Framework does not cover discovery of policy, policy scopes and subjects, or their respective attachment mechanisms. We will consider Policy Attachment which is a mechanism for associating policy with one or more policy scopes. A policy scope is a collection of policy subjects to which a policy applies. A policy subject is an entity (e.g., human, computer, message, an endpoint, interaction, resource) with which a policy can be associated.

2.5. Why do we need Policies?

Policies are high-level guidelines defining the way an institution conducts its activities in its business, professional, economic, social, and legal environment [Fer09].

In the web-based system which considered a secure system, it is hard to have secure connections between involving entities without defining how and what should be exchanged. By enforcing policies through specific mechanisms, these entities will have a high level of controlling its resources and activities.

The implementers, designers, providers and requestors of a web service are in charge of defining their own way of interacting with its conditional policy. Using their own defined-policy or a combination of policies gives different alternatives that are applicable to fit the nature of web-based systems having diverse objectives and environments.

A large number of the most common policies in practice have been proposed and implemented [Fer09]. Some of these policies are more specific e.g. confidentiality, Integrity and availability apply only to security aspects while other policies are applicable to more general aspects.

While designing web services policies, most of the concern will be toward classifying these policies into high-level policies that could be used as a reference to design low-level policies. Moffett and Sloman [Mof88] classify system security policies in three levels (general policies, specific policies and access rules) that are applicable to any web-based secured system.

Without applying policies to constrain the use of the web services, these services will have no means of reaching reliability and integration, will lack security and privacy, and will lose its reason of creation (to provide service, improve quality, increase function, reduce cost and enhance business). For this reason, many common policies are generally referred to authorization aspects that need to be defined to fit the need of the web service. For example, a customer of a website for a travel agency can check his account if she provides the required proof of security certifications in her web browser.

While designing any secured web service, having different objectives and functionalities for different web services in some cases may result in conflicted and overlapped policies. This could be resolved by putting the expected interacting policies into hierarchies with suitable priorities. For example, Authorization polices take over obligation policies.

WS-Policy represents a set of specifications that describe the capabilities and constraints of the security (and other business) policies on intermediaries and end points (for example, required security tokens, supported encryption algorithms, and privacy rules) and how to associate policies with services and end points.

2.6. Relation between WS-policy and WS-Security

WS-policy and WS-security [oas06] are parallel. In fact there are WS-Security Policy specifications that have many assertions that can be added to WS-Policy documents [ibm09a]. Creating a policy that requires means of signing, encryption and time stamping may support the security of a web service. In other words, for a secured web service, there is no response for the incoming messages that didn’t fulfill the policy requirements.

2.7. Policy Attachment to WSDL and UDDI

A Policy Attachment is a mechanism for associating a policy with one or more entities. It details how policies are attached to bindings and is essentially the glue that enforces a web service to adhere to a policy. A Policy attachment can be either WSDL based or UDDI-based. Since UDDI registry houses information about web services and their providers, it is essential that the information contained in a web service's WSDL document is accurately mapped to the UDDI data model. This means that subsequent search operations to discover a registered web service is possible based on the information that is mapped from the WSDL.

Policies can determine acceptable or required behavior for the components of a UDDI registry. Policies can be activated or deactivated as required, and can vary from one registry to another. An XML-based expression grammar for policies is described in the Web Services Policy Framework (WS-Policy) specification, published by the W3C [w3c07a]. The specification also describes how a policy can be associated with a registry object.

2.8. WS-Trust

WS-Trust is a WS-* specification standard that provides extensions to WS-Security and WS-policy for the issue of dealing with the issuing, renewing, and validating of security tokens, as well as with ways to establish and assess the broker trust

relationships between participants in a secure message exchange [ibm09a], [oas06], [w3c07a].

The WS-Trust specification was created as part of the Global XML Web Services Architecture (GXA) framework, which is a protocol framework designed to provide a consistent model for building infrastructure-level protocols for web services and applications [Box02]. It was authored by Microsoft, IBM, Verisign, and RSA Security and was approved by OASIS as a standard in March 2007.

WS-Trust defines concepts such as a security token service and a trust engine which are used by web services to authenticate other web services. In other words, using the extensions defined in WS-Trust, applications can engage in secure communication designed to work within the web services framework.

2.9. Relations between WS-Trust, WS-Policy and WS-Security

WS-Security begins with the assumption that, if one of the parties uses a particular type of security token within the WS-Security header, then the other party will be able to interpret and process this token. A fundamental issue that WS-Security neither addressed nor attempted to address is how two entities (a SOAP client and SOAP Service) can agree on the nature and characteristics of the security tokens that are the fundamentals of WS-Security.

To interact with WS-Policy, the following example emphasizes the need to have a WS-Trust standard. If every company has defined its own policies for what a secure message will be, without taking in consideration the capabilities of the partners/customers with which they have business, the chances of having an intersection between requirements and capabilities of parties would seem small. It sounds like an interoperability nightmare.

The motivation toward WS-Trust is supported by the facts that there are different formats for security tokens (e.g. X.509 certificates, Kerberos tickets, SAML assertions, XACML policies, etc.), and it's unlikely to expect that the endpoint will understand each of these options and the fact that there is no guarantee that there will be an intersection between the sets of supported security token formats of different SOAP actors who are willing to use WS-Security to secure their messages [Mad03]. This is of course not a limitation of WS-Security; it simply reflects the heterogeneity of the security environments between which WS-Security must operate.

2.10. BPEL

BPEL as standardized by OASIS [oas07], [wik09] is the most popular language for web service composition. This is supported by the fact that many software companies have integrated BPEL orchestration engines into their products.

The combination of web services from different providers is done in order to create a more advanced and collaborating web service. Two paradigms are applied to represent the workflow, one takes care of which web services participate in the

interaction (control flow) and the other deals with the data been transferred between these interactions (data-flow).

According to various control-flow patterns, BPEL has the ability to provide interactions among web services that participate in the composition. There are three main activities involved in the interaction: <invoke> for invoking an operation from one of the partner web services, <reply> to send a response for the requestor, and <receive> to receive a request from the requestor.

On the other side, there are non-functional aspects which need to be considered for the process. Security, reliability and Quality of Service are the most important issues in this respect for BPEL. Since the development of many web services standards such as WS-Security [oas06], WS-Policy [w3c07a], WS-Trust [oas09a] and WS-Federation [oas09b], more concerns have been addressed to cover the importance of providing a high level of non-functional aspects for all these specifications in order to encourage the companies to adopt web services in their business activities. These companies now are prioritizing key issues such authorization, authentication, confidentiality, integrity and privacy to evaluate their services with its competitors [mic06].

3. AJIAD TRAVEL AGENCY

Ajiad is a travel agency that has expanded its office services to cover the online trade customers. *Ajiad* offered many of its everyday operations to a web services-based system, some of which have a certain level of privacy and security for the customers who have been granted privileges. *Ajiad* now declared new rules for defining the way its web services should be accessed by means of policies in terms of whom, when, and in what way they can be used.

To clarify the idea, the assumptions applied to this example are the following:

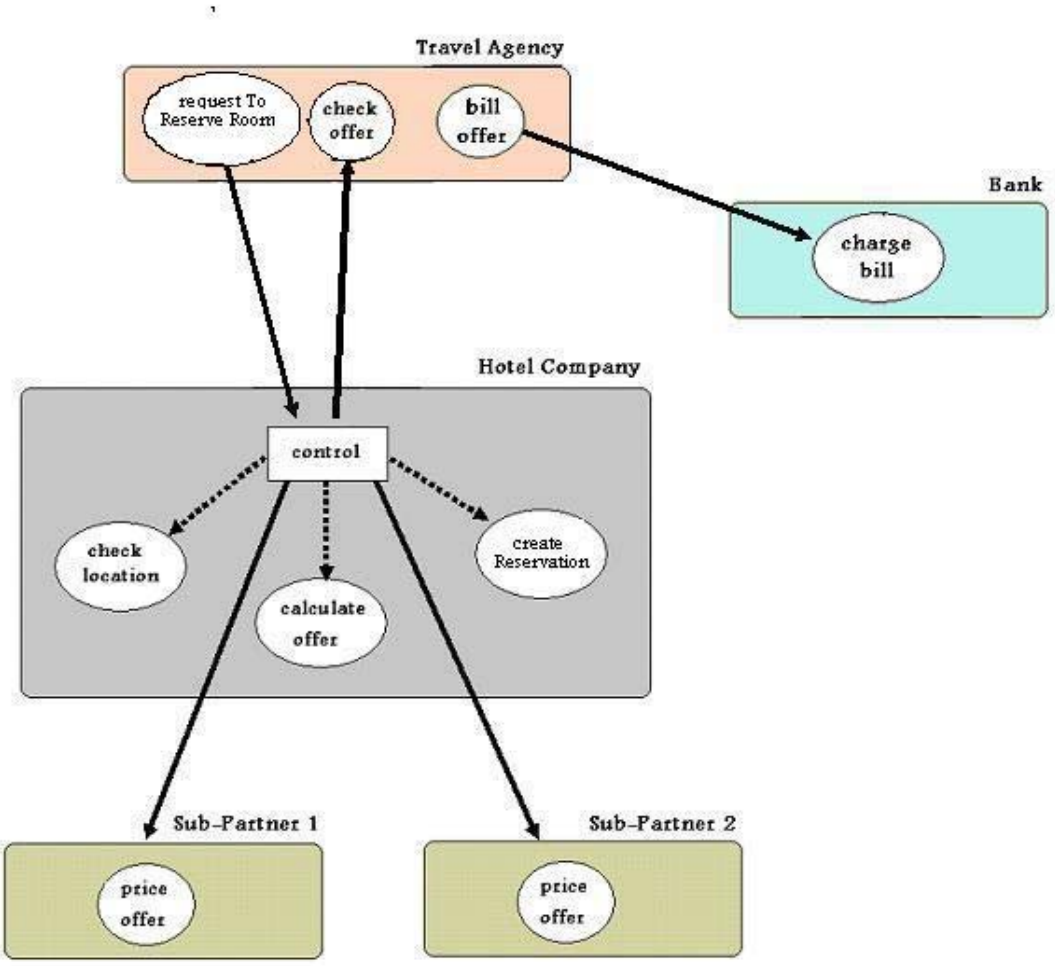
The development team at *Ajiad* had already developed the required web services. These web services were created on the same procedure that will use SOAP to interact with the content Management system, define the web service using Web Services Description Language (WSDL) and interact with a Universal Description, Discovery and Integration (UDDI) registry. While doing that, the staff also used (UDDI) to find data within one, which ultimately led them to create one for the company to enable other organizations to interact with the *Ajiad*.

The team implemented its own way to prevent unauthorized access to their system by means of WS-Security for their web services.

Now, the next task for the development team is how to make sure that the clients accessing their web services are not necessarily using the new web services that they require. This query led to develop and define policies for the web services as to how and in what way they can be used. In other words, it's necessary now to enforce the clients accessing the web services of ***Ajiad*** to do so in prescribed manner ensuring the security constraints implemented before.

The need to have appropriate configuration and security policies is highly demandable for web services to be truly useful in the real life environment. Actually this problem the web services faces now. WS-Policy solves this problem by allowing policy definitions to be bound to web services. You can secure the data while transporting and limit access to authorized individuals or organizations, and then verify that information has not been altered in transit by enforcing a certain level of encryption and digital signatures. Requiring and signing timestamp for policy will prevent messages from being captured and replayed.

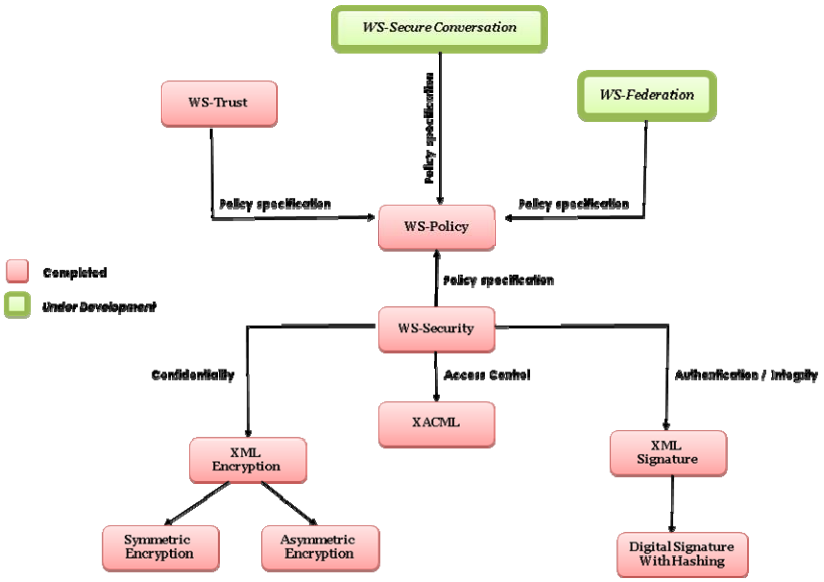
Figure 1: An Example of Ajiad Travel Agency



4. PATTERN DIAGRAM FOR WEB SERVICES STANDARDS

Figure 2 shows how the patterns that we are presenting (the WS-Policy pattern, the WS-Trust pattern) fit with some existing patterns(The WS-Security pattern , Patterns for the eXtensible Access Control Markup Language, Digital Signature with Hashing and XML Signature patterns, Symmetric Encryption and XML Encryption Patterns). WS-SecureConversation and WS-Federation (italic and black borders) are part of our future work in web service standards. All the lower-level patterns shown here have been written by the Secure Systems Research Group (SSRG) at FAU.

Figure 2: Pattern Diagram for web services



5. WS-POLICY PATTERN

5.1. Intent

Without a clear definition of how to use web services, their use could be chaotic. WS-Policy defines a base set of assertions that can be used and extended by other web services specifications to describe a broad range of service requirements and capabilities.

WS-Policy provides a way to check the requests made by requestors in order to verify they satisfy their assertions and their conditions before interacting with the web service.

5.2. Example

Ajiad is a travel agency that intends to provide online services to its customers. *Ajiad* now offers many of its everyday operations as a web services-based system, some of which have a given level of security for customers.

In the current situation, some of *Ajiad's* Travel Agency customers have been accessing web services they are not allowed to access. The reasons for that are outdated

and weak services not having systematic guidelines to control their use. As a result *Ajiad* is losing money. In other words, *Ajiad* has reliability and security problems.

5.3. Context

Distributed applications need to communicate in a collaborative way to perform work in a web services environment. For this objective, they use the Internet which is an unreliable and insecure environment.

5.4. Problem

Without applying relevant policies, web services will have no means to assure reliability, availability and security in their integration and may lose their ability to provide service, improve quality, increase function, reduce cost and enhance business. It is not clear for the users what is illegal or which conditions apply to the use of web services.

The possible solution is constrained by the following forces:

- **Data confidentiality of web services:** Malicious users may try to read or modify sensitive information. We need to define appropriate policies to protect the information.

- **Message exchange:** The provider may perform a malicious activity; we need to assure the delivery of messages between partners and gives a requester the ability to verify its provider.

- **Policy Integrity:** Malicious users may try to replace policy assertions to their own benefit. We need to assure that policy assertions have not been modified.

- **Degree of security and provision:** A policy may offer several alternatives that vary from weak to strong requirements. An adversary may remove alternatives, except the weakest one. We need to stop this action which would result in misuses.

5.5. Solution

Web services can be protected against unauthorized access by having policies that provide conditions in order to use them. Requesters willing to use protected web services are required to comply with its policy.

Each policy is defined in terms of nested constructs which convey the restrictions implied by the policy. When the policy is attached to a web service, clients wanting to transact with that web service must comply with its assertions (e.g. signing, encryption, timestamp, and username) as specified in the policy.

In general, any entity in a web services based system may expose a policy to convey conditions under which it provides service. Satisfying assertions in the policy usually results in behavior that reflects these conditions. For example, if two entities -

requester and provider - expose their policies, a requester might use the policy of the provider to decide whether or not to use the service. A requester may choose any alternative since each is a valid configuration for interaction with the service, but a requester must choose only a single alternative for that.

A **Policy** is a collection of policy alternatives that has its own name, reference (to be accessed from other subjects) and ID. A policy with zero alternatives contains no choices; a policy with one or more alternatives indicates choice in requirements or capabilities within the policy. A **PolicyAlternative** is a collection of policy assertions. Alternatives and assertions are not necessary ordered.

A **PolicyAssertion** represents a capability, a constraint, or a requirement of the behavior of a web service (e.g. guarantee of message delivery). Or it could be defined as a declaration of certain facts, such as “Jad was granted update privileges to database X at time Y”. In alternative view we can define a policy assertion to be a set of requirements. For example, a policy assertion might specify the security token types that are used to digitally sign or encrypt SOAP messages between the client and Web service. A **PolicyAssertion** identifies a behavior that is a requirement or capability of an entity (e.g., human, computer, message, an endpoint, interaction, resource). Satisfying assertions in the policy usually results in behavior that reflects these conditions. A **PolicyAssertion** has two parameters used to define the behavior indicated by the assertion: attributes and children. A **PolicyAssertionType** represents a class of policy assertions to indicate domain-specific semantics (e.g., security, transactions). A **PolicyAssertion** may reference another policy.

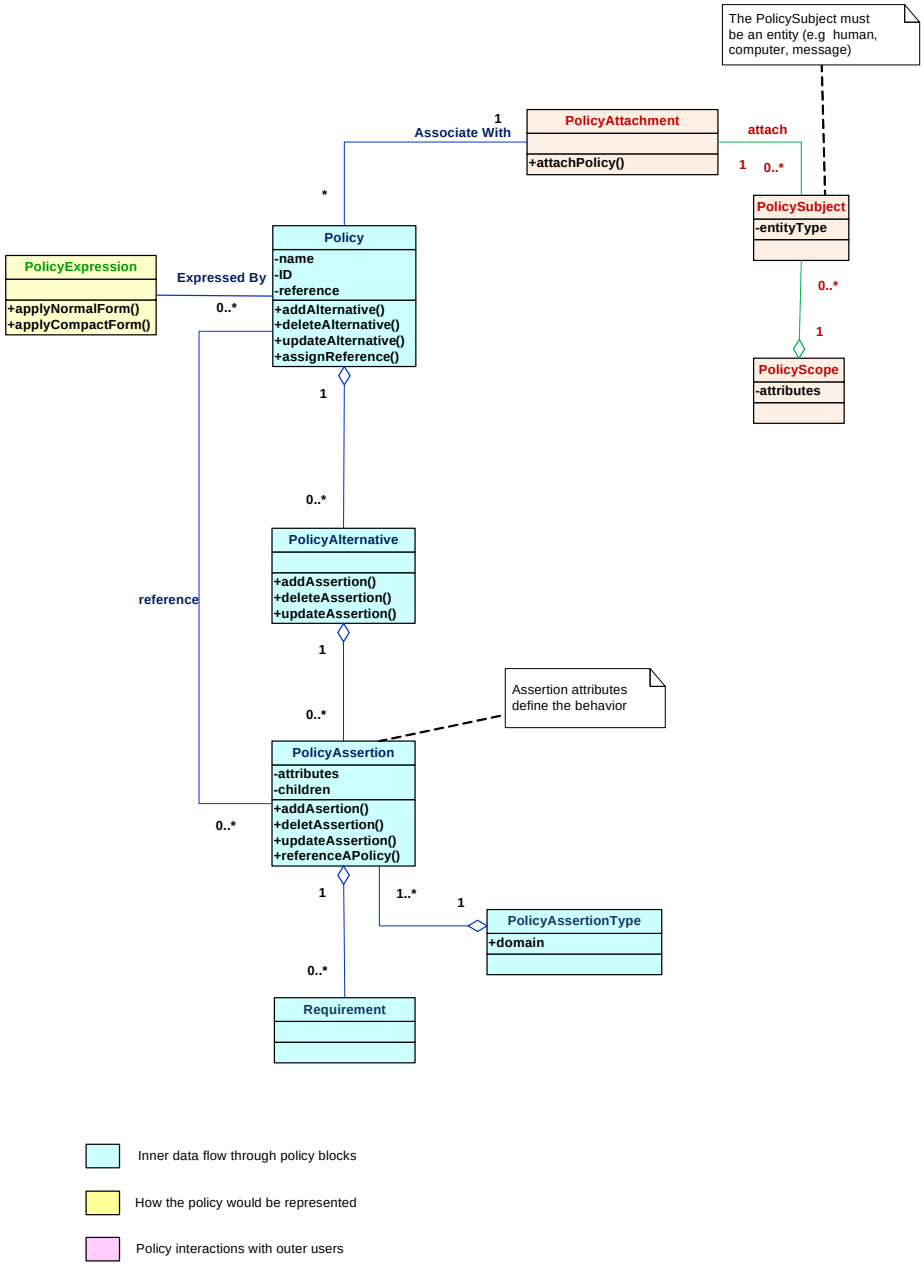
The formal term for a policy is **PolicyExpression**, and we use it to convey a policy in an interoperable form. In other words a **PolicyExpression** is a set of one or more policy assertions that combined together will perform a specific task. It could be interrupted also as a Form that is either structured in a normal or a compact form to express a policy.

A **PolicyAttachment** is a mechanism for associating a policy with one or more entities. It details how policies are attached to bindings and is essentially the glue that enforces a web service to adhere to a policy. A **PolicySubject** is an entity (e.g., human, computer, message, endpoint, interaction, resource) with which a policy can be associated, while a **PolicyScope** is composed of a collection of policy subjects to which a policy applies.

Structure

Figure 3 describes the structure of this pattern.

Figure 3: Class Diagram for the WS-Policy Pattern



Dynamics

We describe the use cases "*create a policy for a web service*" and "*request a service*".

Create a policy (figure 4):

Summary: A provider will create a new policy for an existing web service.

Actors: policy provider.

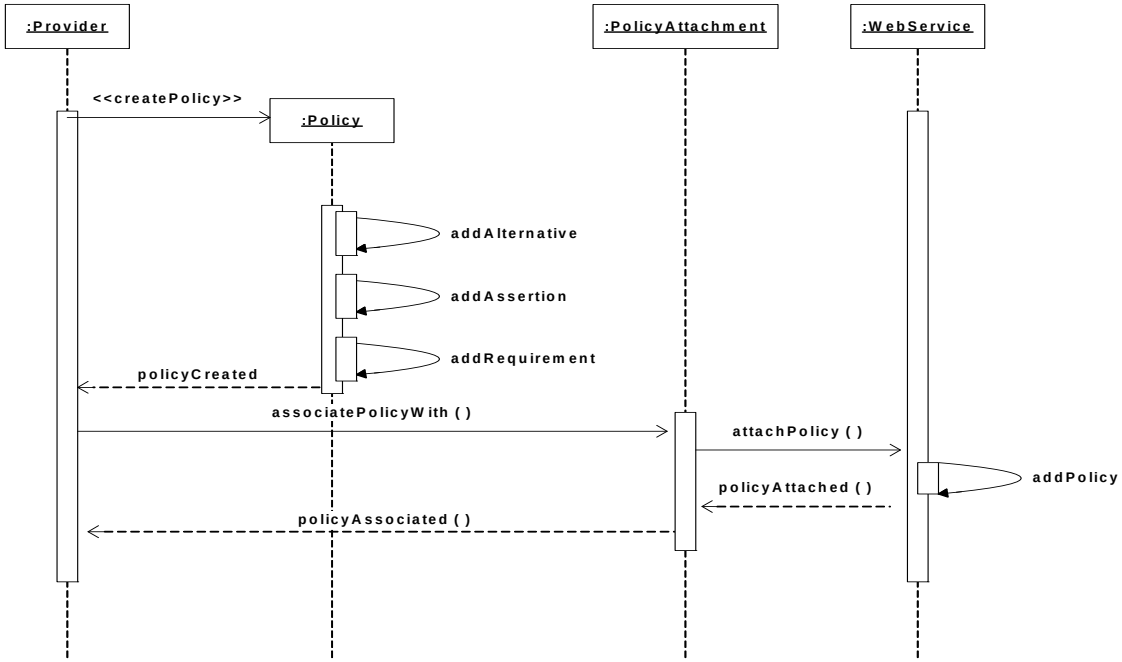
Precondition: The provider has already created a web service.

Description:

1. The policy provider will create the policy by specifying its required alternatives, assertions and requirements. The provider creates as many assertions as necessary to meet the conditions for the web service.
2. All the alternatives, assertions and requirements are added to the created policy.
3. The provider will send a request to the Policy Attachment to associate the policy with the end entity (web service).
4. The Policy Attachment will attach the policy to the web service, which in turn updates its content, adds the policy and acknowledge the Policy Attachment.
5. A reply from the Policy Attachment informs the provider that the attachment process is completed.

Postcondition: The provider has attached the policy to its designated web service.

Figure 4: Sequence diagram for creating a new policy



Request a service (Figure 3):

Summary: A requester will request the use of a web service that has a policy.

Actors: Requester.

Precondition: The provider had already created a web service with a policy that controls its services.

Description:

a. The requester sends a request to use the web service.

b. The web service forwards the request to the PolicyAttachment.

c. The provider will ask the web service to check its policy for verifications.

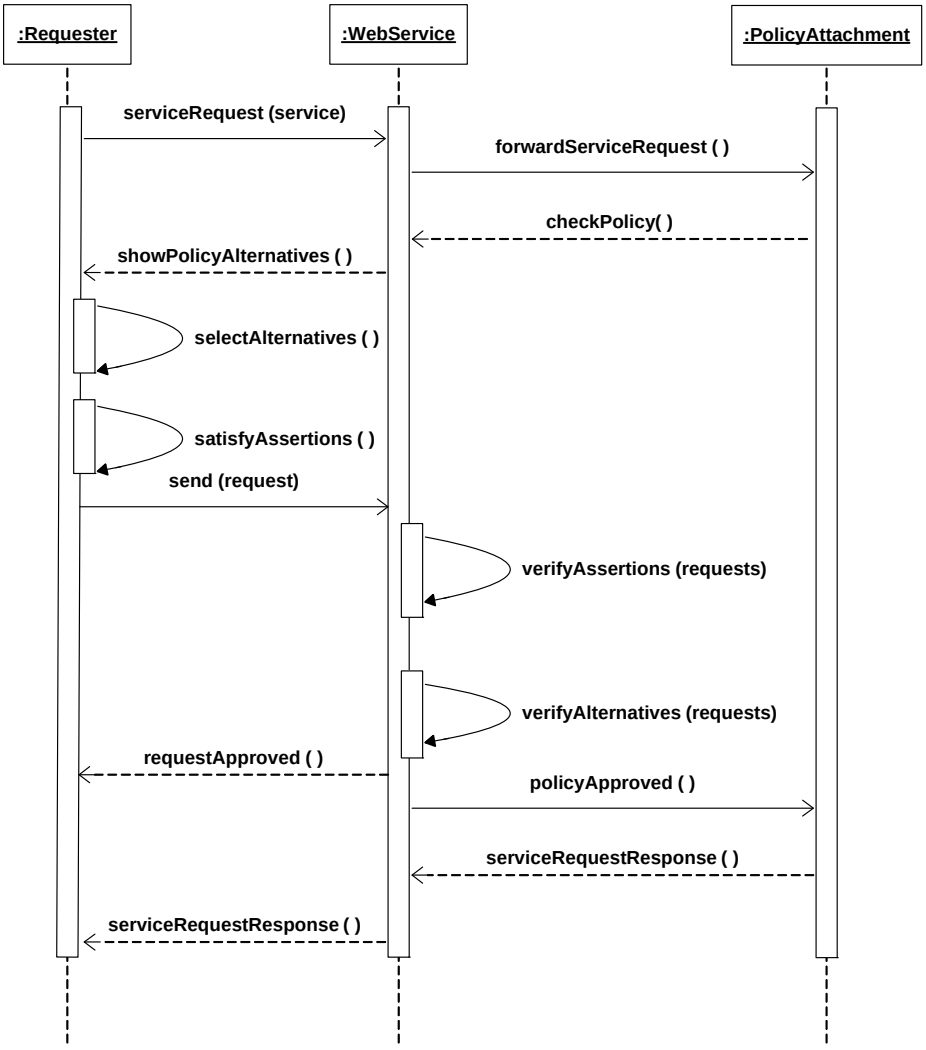
d. The web service shows its policy alternatives to the requester to comply with.

e. The requester selects from the alternatives then satisfies that alternative's assertions, the sends a request to be verified against the policy. The web service checks all possibilities that results and approves or denies.

f. The policy attachment will respond to the web service, which in turn forward it to the requester.

Postcondition: The Requestor can now use the web service after satisfying its policy conditions.

Figure 5: Sequence Diagram for requesting a service.



5.6. Implementation

The web-based system has decided to use WS-Policy to convey conditions on the interactions between entities (Provider, Broker, Requester, etc). This was supported by the fact that WS-Policy is a standard and several products support its use.

In order to assure effective implementation, we need to take in consideration the following:

1. A policy may or may not reference another policy (ies) depending on the level of authentication that is required.

2. A policy alternative may contain multiple assertions of the same type. Policy assertions within a policy alternative are not ordered. However, providers can write assertions that control the order in which behaviors are applied.

3. Policy Assertions are the main blocks of the policy that specify a particular behavior. Translating these assertions will qualify the behavior indicated by. For example, sp:AsymmetricBinding assertion is identified to support a specific reliable messaging mechanism, while sp:SignedParts assertion is used to indicate message-level security and sp:EncryptedParts assertion is used to indicate the parts of a message that require confidentiality.

4. A policy expression conveys policy in an interoperable form, either in a normal form (which is the most straightforward XML representation of the policy data model) or in an equivalent compact form (that is used to compactly express a policy with more description about definitions and outlines).

5. For security considerations, a policy Expression should not reference itself directly or indirectly.

5.7. Example Resolved

Ajiad now defined systematic rules to specify the way its web services should accessed in terms of who, when, and in what, as well as conditions

Ajiad's new web-based system now has more control over its services by applying prerequisite conditions and security constraints through policies. So, in order to use any service, all customers are required to comply with its policy conditions and satisfy its requirements (for example, by using the required security token types specified by the policy) and agree with its terms before using the web service.

5.8. Consequences

The WS-policy pattern presents the following advantages:

- **Data confidentiality of web services**: It's possible to secure the data of web services, since Policy providers now can use mechanisms from other web services specifications such as WS-Security [ibm09b], XML Digital Signature [w3c08] and WS-Metadata Exchange [w3c09] and that's by securing access to the policy,

requiring authentication for sensitive information and omitting sensitive information from the policy.

- **Message-exchange**: The pattern has the way to assure messages exchange between the partners by giving policy providers the ability to avoid older or weaker policy alternatives and gives the requestor the ability to verify the policy provider.

- **Policy Integrity**: Using the appropriate signing mechanism will protect the policy assertions from tampering, e.g. Requestors should discard a policy unless it is signed by the provider and presented with sufficient credentials.

- **Availability**: The pattern mitigate the chance of denial of services threats By enforcing the policy implementers to use a modal margin with defaults on number of policy alternatives, number of assertions in an alternative, depth of nested policy expressions.

The pattern also has some (possible) liabilities:

- WS-Policy is an immature specification which is still changing.

- The WS-Policy standard is a lengthy document with a lot of details that we left out to avoid making the pattern too complex. For more details check the WS-Policy Standard web page [w3c07a].

5.9. Known Uses

Some products that implement this pattern are:

- *The HP SOA Systinet Standard Edition* is a platform for SOA Governance. This SOA architecture tool provides the different levels of governance [hp09].

- *The Layer 7 SecureSpan XML Virtual Appliance* provides security and threat protection for internal and cloud-based XML and web services applications [vmw09].

- *Xtradyne's WS-DBC* is an XML/SOAP Firewall specifically designed for use in environments that demand performance, scalability, availability, and policy management [xtr05].

- *DataPower's XS40 XML Security Gateway* is a network appliance that operates as an XML proxy. It provides security functions for XML-based communications [dat05].

5.10. Related Patterns

- *A pattern language for security models* [Fer01] discusses three patterns that correspond to the most common models for security: Authorization, Role-Based Access Control, and Multilevel Security which can be applied in all the levels of the system.

- *Rule Object 2001: A Pattern Language for Adaptive and Scalable Business Rule Construction*. Due to a continuous refinement and update of the Rule Pattern Language (RPL), this paper provides and overviews the patterns and focus on Rule Object, Type Rule and Configurable Workflow as three representative patterns in greater detail. [Ars01]

- *Patterns for the eXtensible Access Control Markup Language* [Del05] presents three architectural patterns for XACML. The XACML Authorization pattern unifies the definition of authorization rules throughout an organization. WSPL is a specialization of XACML Authorization describes access control rules for web services. The XACML Access Control Evaluation pattern defines request/response syntax for access control decisions.

- *Patterns for Access Control in Distributed Systems* [Del07a] considers the access control in the distributed systems that has a variety of security threats, and describes patterns for it. The patterns handle different missions of describing how to decide if a subject is authorized to access an object, how to implement the Access Matrix or RBAC model and how to control access to objects.

5.11. Conclusion

Without a clear definition of how to use web services, their use could be chaotic. WS-Policy provides means of possible configurations of a web service by enforcing certain level of defined security.

Our future work is designing patterns for other web services standards such as WS-Trust and WS-SecureConversation that depend on WS-Policy as a prerequisite foundation. This will give us a good chance to analyze and discover how WS-Policy fits with other web services standards and how much it could simplify the implementation of these specifications in real-life business applications.

6. WS-TRUST PATTERN

6.1. Intent

WS-Trust defines a security token service and a trust engine which are used by web services to authenticate other web services. Using the functions defined in WS-Trust, after establishing trust applications can engage in secure communication.

6.2. Example

The *Ajiad* travel agency offers its travel services through several different business portals to provide travel tickets, hotel and car rental services to its customers. *Ajiad* needs to establish trust relationships with its partners through these portals.

The *Ajiad* supports different business relationships and needs to be able to determine which travel services to invoke for which customer. Without a well-defined structure, *Ajiad* will not be able to know if this partner is trusted or no, or to automate the trust relationships quickly and securely with its partners, which may lead to losing a

valuable business goal of offering integrated travel services as a part of the customer’s portal environment.

6.3. Context

Distributed applications need to establish secure and trusted relationships between them to perform some work in a web-service environment which may be unreliable and/or insecure (e.g. the Internet). The concept of "Trusting A" mainly means "considering true the assertions made by A", which is not necessarily corresponding to the intuitive idea of trust in its colloquial use.

WS-Security begins with the assumption that, if one of the parties uses a particular type of security token within the WS-Security header, then the other party will be able to interpret and process this token. A fundamental issue that WS-Security did not address is how two entities (a SOAP client and SOAP Service) can agree on the nature and characteristics of the security tokens that are the fundamentals of WS-Security.

6.4. Problem

Establishing security relationships is fundamental for the interoperation of distributed systems. Without applying relevant trust relationships expressed in the same way between the involved parties, web services have no means to assure security and interoperability in their integration. How can we define a way for the parties to trust each other’s credentials?

The possible solution is constrained by the following forces:

- ***Knowledge:*** In human relationships, we are concerned with first knowing a person before we trust her. That attitude applies also to web services. We need to have a structure that encapsulates some knowledge about the unit we intend to trust.

- ***Policy consideration*****:** The web service policy encloses all the required assertions and conditions that should be met to use that web service. The trust structure should consider this policy for verification purposes.

- ***Confidentiality and Integrity*****:** Policies may include sensitive information. Malicious consumers may acquire sensitive information, fingerprint the service and infer service vulnerabilities.

- ***Tamper-Freedom***: The data to be transferred between the partners through messages may be private data that need to be protected. Outsiders may try to tamper or replace these messages.

- ***Time Validity***: For protection purposes, any interactions or means of communications (including the trust relationships) between the web services are restricted using a time limit that determines for how long that interaction is valid.

6.5. Solution

We define explicitly an artifact that implies trust. This artifact implies what kinds of assertions are required to make trustworthy interactions between the involved web services.

We should consider the claims and information sent by the requester in order to obtain the required secure token that is proved enough to establish a trust relationship with its target partners. Once initiated, this solution defines the protocol to handle that trust relationship properly.

Structure

Figure 1 describes the structure of this pattern. **Claim** is a statement made about a client, service or other resource (e.g. name, identity, key, group, privilege, capability, etc.). Claims are assertions, for example: “I am Joman”, “I am an authenticated user and I am authorized to print in printer P”. Claims are used to validate the request.

Security Token Service (STS) is a web service that issues security tokens. It makes decisions based on evidence that it trusts. **STS** is responsible for generating security tokens and, providing challenges for the requester to ensure message freshness (the message has not been replayed and is currently valid), verification of authorized use

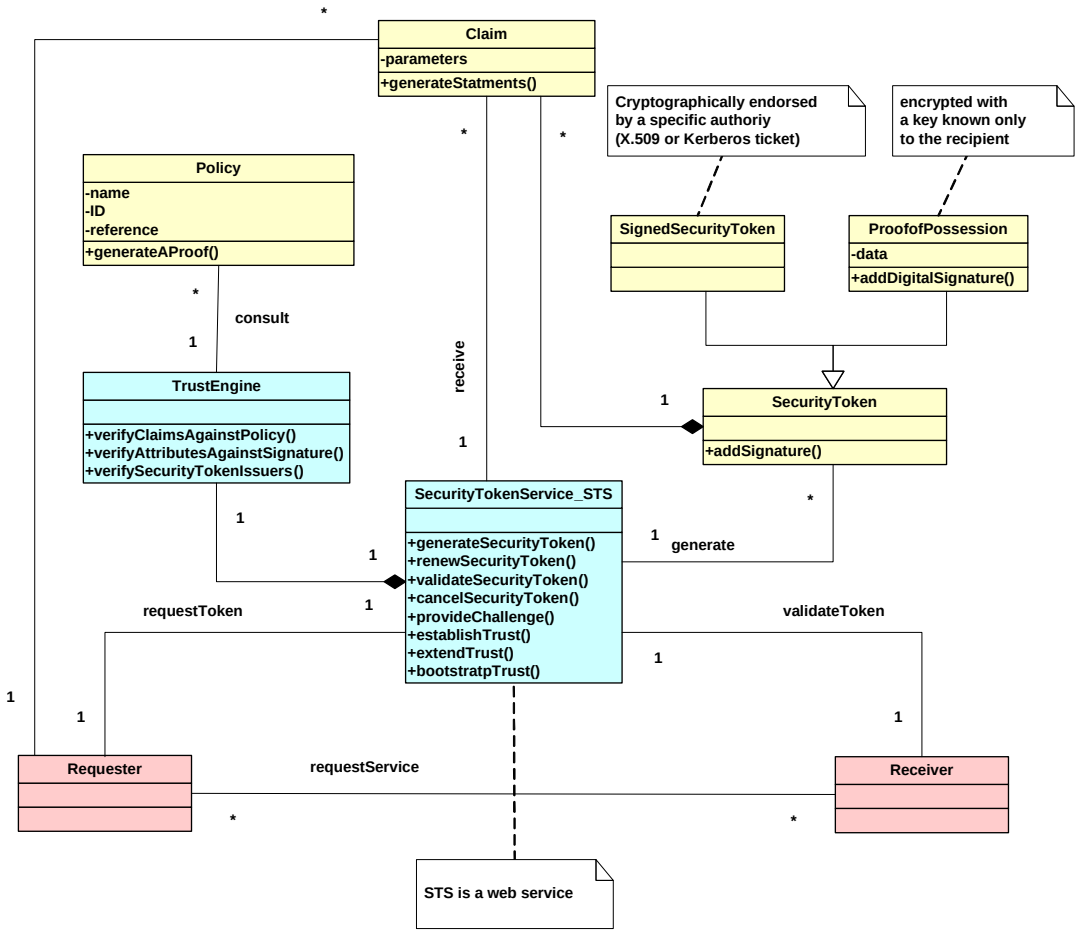
of a security token, and finally establishing, extending and removing trust in a domain of services. The **STS** is the heart of WS-Trust and forms the basis of trust brokering.

The main output of the **STS** is a trust relationship between the requester and the receiver. It represents the characteristic that one entity is willing to rely upon a second entity to execute a set of actions and/or to make set of assertions about a set of subjects and/or scopes in a secure, reliable and time-relevant manner.

A **Security Token** is a collection of claims. It is possible to add signatures to tokens. **Security Token** also is a generalization of two types: **Signed Security Token** that is cryptographically endorsed by a specific authority (e.g. an X.509 certificate or a Kerberos ticket) and **Proof-of-Possession (PoP) Token** that contains a secret *data* parameter that can be used to prove authorized use of an associated security token and provides the function of adding digital signature. Usually, the proof-of-possession information is encrypted with a key known only to the recipient of the POP token.

Each **STS** has a **Trust Engine** that evaluates the security-related aspects of a message using security mechanisms and includes policies to verify the requester's assertions. The **Trust Engine** is responsible for verifying security tokens and verifying claims against policies. A **Policy** is a collection of policy assertions that have their own name, references, and ID. Policies form the basic conditions to establish a trust relationship. Verifying the requester's claims against policy conditions generates an approval to use the target service. A policy may reference another policy (ies), in order to check the tokens sent by the requester or verified by the receiver.

Figure 6: Class Diagram for the WS-Trust Pattern



Dynamics

We describe the dynamic aspects of the WS-Trust using sequence diagrams for the use cases "*create security token*" and "*access a resource using a token*".

Create a security token (Figure7):

Summary: STS creates a security token using the claims provided by the requester.

Actors: A Requester

Precondition: The STS has the required policy to verify the requester claims and the requester provides parameters in form of *claims* and *RequestType* signed by a *signature*.

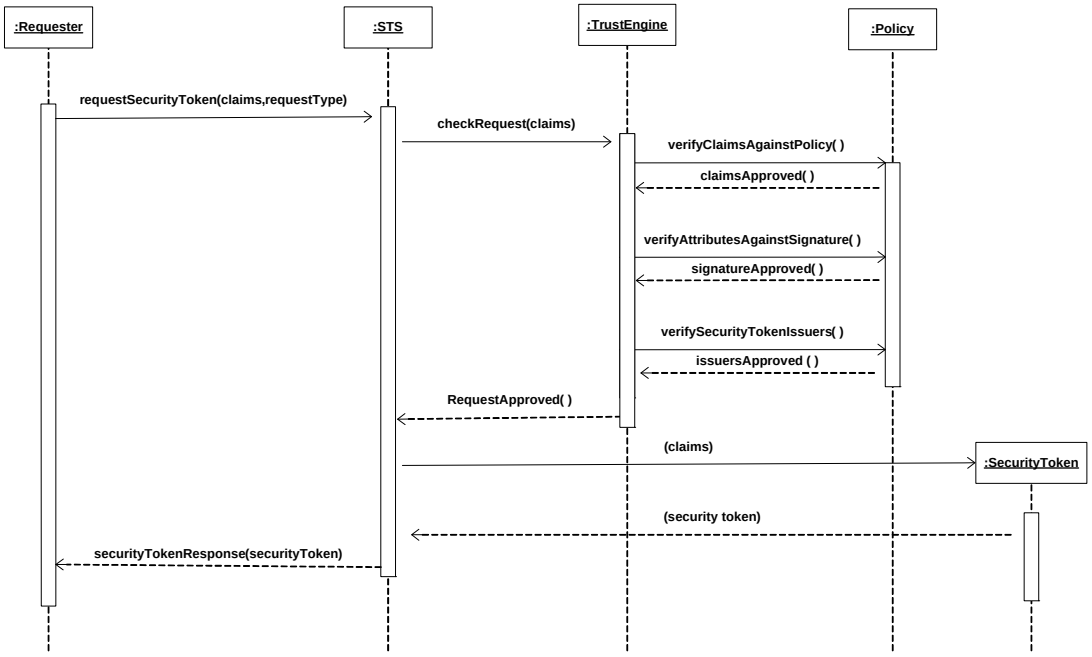
Description:

1. The requester requests a security token by sending the required parameters of *claims* and *RequestType* signed by a *Signature* to the STS. The signature verifies that the request is legitimate.
2. The STS contacts the Trust Engine to check the requester's claims.
3. The Trust Engine contacts the web service's policy to verify the claims including attributes and security token issuers of the requester.
4. Once approved, the STS create a new security token containing the requested claims.

5. The STS sends back its *SecurityTokenResponse* with a security token issued for the requester.

Postcondition: The requester has a security token that can be used to access resources in a trusted unit.

Figure 7: Sequence diagram for creating a security token



Access a resource using a token (Figure 8):

Summary: A STS establishes trust by verifying *proofOfClaims* sent by the requester.

Actors: A Requester

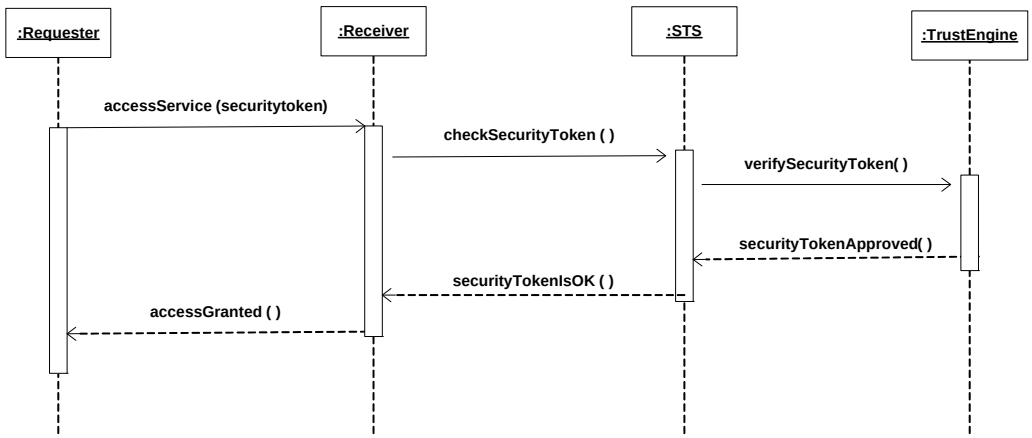
Precondition: The Trust Engine has the required policy to verify the requester' security token.

Description:

a. The requester asks for a service access by providing the required security token.
b. The receiver sends the security token to the STS for verification.
c. The STS use its Trust engine to verify the security token claims.
d. Once approved, the STS notify the receiver that the security token is valid and verified.
e. Then, the receiver gives the requester the right to use the service.

Postcondition: The requester has a security token that can be used to access services in a Receiver web service.

Figure 8: Sequence diagram for validating a security token



6.6. Implementation

In this solution, the concept of trust is realized by obtaining a security token from the web service (in our diagram, the Security Token Service) and submitting it to the receiver who in turn validates that security token through the same web service. Upon approving, the receiver establishes a valid trust relationship with the receiver that lasts as long as the security token is kept valid.

In order to assure effective implementation, we need to take in consideration the following:

- To communicate trust, a service requires proof, such as a signature to prove knowledge of a security token or set of security tokens. A service itself can generate tokens or it can rely on a separate STS to issue a security token with its own trust statement.

- Although the messages exchanged between the involved entities are protected by WS-Security; still three issues related to security tokens are possible: security token format incompatibility, security token trust, and namespace differences. The WS-Trust pattern addresses these issues by defining a request/response protocol (in which the client sends *RequestSecurityToken* and receives *RequestSecurityTokenResponse)* and introducing a Security Token Service (STS) which is another web service.

- Based on the credential provided by the requester, there are different aspect of requesting a security token (RST), each of which has a unique format that the requester should follow.

    - The issuance process: formed as *RequestSecurityToken (RequestType, Claims).*
    - The renewal process: formed as *RequestSecurityToken (RequestType, RenewTarget).*
    - The cancel process: formed *RequestSecurityToken (RequestType, CancelTarget).*

      The cancelled token is no longer valid for authentication and authorization.

    - The validate process: formed as *RequestSecurityToken (RequestType, ValidateTarget).*

.

6.7. Example Resolved

Ajiad now has the ability to automate its trust relationships with its partners by managing the registration tasks for all its partners and issuing customers a unique ID's. In this case, *Ajiad* provides a mediator between the customers and its participant partners and plays the role of negotiator and third-party player who is looking to satisfy both sides.

Ajiad now can offer a Security Token Service for its business partners, who may find useful ways to take advantage of credit processing and other services offered by *Ajiad*, which now has new business opportunities.

6.8. Consequences

The WS-Trust pattern presents the following *advantages:*

- ***Security.*** By extending the WS-Security mechanisms, we can handle security issues such as security tokens (the possibility of a token substitution attack), signing (where all private elements should be included in the scope of signature and that this signature must include a timestamp).

- ***Trust.*** With this solution, we have the choice of implementing WS-Policy framework to support trust partners by expressing and exchanging their statements of trust. The description of this expected behavior within the security space can also be expressed as a trust policy.

- ***Confidentiality.*** We can achieve confidentiality of users' information. Since Policy providers now can use mechanisms provided by other web services specifications such as WS-Security [ibm09b] to secure access to the policy, XML Digital Signature [w3c08] to authenticate sensitive information and WS-Metadata Exchange [w3c09].

- All the security tokens exchanged between the involved parties are signed and stamped with unique keys that are known only to the recipients.

- ***Time validity***. We can specify time constraint in the parameters of a security tokens issued by STS. This constraint will specify for how long that security token is valid. Upon expiring, the security token's holder may renew or cancel it.

The WS-Trust pattern presents the following *liabilities:*

- The efficiency of WS-Trust may suffer from the repeated round-trips for multiple token requests. We need to make an effort to reduce the number of messages exchanged.

- The WS-Trust standard is a lengthy document and several details were left to avoid making the pattern too complex. The interested reader can find more details in the WS-Trust Standard web page [oas09].

6.9. Known Uses

- *DataPower's XS40 XML Security Gateway* [dat05] is a device for securing web services that provides web services access control, message filtering and field-level encryption. It centralizes policy enforcement, supporting standards such as WS-Security, WS-Trust, WS-Policy and XACML.

- *SecureSpan™ XML Firewall* [lay09] enforces WS* and WS-I standards to centralize security and access requirements in policy that can be run as a shared service in front of applications.
- *Vordel Security Token Service* [vor09] is used to issue security tokens and to convert security tokens from one format to another. The security tokens created by an STS are bound to the messages travelling within a Service- Oriented Architecture (SOA).
- *PingTrust*, a standalone WS-Trust Security Token Server [pin06] creates and validates security tokens that are bound into SOAP messages per WS-Security standard.

6.10. Related Patterns

The *Trust* Analysis Pattern [Fay04] provides a conceptual model that embodies the main aspects of the trust concept to make it applicable to different domains and applications.

The *Credential* Pattern [Mor06] addresses the problem of exchanging data between trust boundaries and how to resolves the problem of authenticating and authorizing a principal's identity over different systems.

The *Circle of Trust* pattern [Del07a] allows the formation of trust relationships among service providers in order for their subjects to access an integrated and more secure environment.

6.11 Conclusions

This pattern describes how existing direct trust relationships may be used as the basis for brokering trust through the creation of security token issuance services. These security token issuance services build on WS-Security to transfer the requisite security tokens in a manner that ensures their integrity and confidentiality.

Our future work is designing patterns for other web services standards such as WS-Federation and WS-SecureConversation that depend on WS-Trust as a prerequisite foundation. This will give us a good chance to analyze and discover how WS-Trust fits with other web services standards and how much it could simplify the implementation of theses specifications in real-life business applications.

7. SECURITY CONSIDERATIONS FOR BPEL

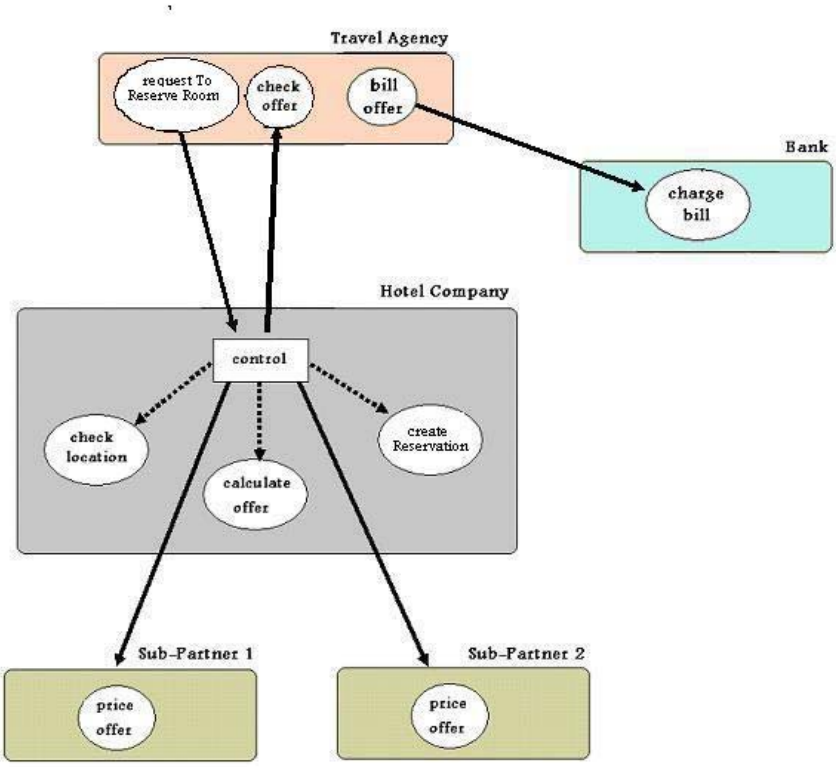
As a web service composition language, BPEL promises a convenient and effective means for application integration over the Internet in typical B2B interaction scenarios. BPEL a language for web services composition and several implementation of it are already developed.

However, for BPEL to keep its promises it is necessary to provide more support for security. Vendors and companies are looking to have their requirements of authentication, integrity and confidentiality satisfied. In this paper we will address security considerations in BPEL and how to enforce them as well as its interactions with other web services standards such as WS-Security and WS-Policy.

7.1. An Example for a Collaborative Business Process

In Figure 1, an example from the area of travel agency is shown that will be used to illustrate the security issues arising when we defined a BPEL process that would be shared across many partners.

Figure 9: An example of a collaborative business process



The application context of the distributed business process shown in Figure 1 is for a travel agency with a chain of several offices across the world. The travel agency has a partnership with a hotel company and a bank using web services. The travel agency defines a BPEL process to serve its customers. The process will reserve suites or rooms from the hotel and invokes the bank's payment web service to pay the transaction amount. The process is deployed by the travel agency and the result web service can be accessed from any of its distributed offices.

The business process is set up in a service-oriented computing (SOC) environment where all functions used for the application are provided as web services and the composition of web services is accomplished using BPEL.

7.2. Description of Example Business Process

In this example, a controlling BPEL process is executed in a system of the hotel denoted by ***control.*** In Figure 1, a ***requestToReserveRoom*** process of the travel agency that may itself be a web service invokes the web service offered by the ***control*** process at the hotel providing a list of rooms to be booked. Before placing a reservation, the travel agency expects a price offer accompanied by a commitment with respect to the reservation date.

Now, the ***control*** process invokes a ***check location*** web service for checking the availability of the available rooms in location. In order to do that, the list of reservations to be placed is passed to this web service. After checking availability in locations, two lists of rooms that need to be reserved are retrieved from sub-partners. Together with these lists, a transaction ID for the reservation in progress and the credentials required to invoke the respective web services of the two sub-partners are returned by ***check location***.

Upon receiving the response of ***check location***, the ***control*** process invokes the ***price offer*** web services of the sub-partners and provides the respective list of items to each of them. To get access to these web services, the ***control*** process will use the proper credentials (which may be entirely different for each web service) returned by ***check location***.

The ***price offer*** web service will check the availability of rooms on the list in order to return a list of prices and availability on location. The ***control*** process then invokes a ***calculate offer*** web service of the hotel to prepare an offer for the travel agency. For this purpose, the ***control*** process passes the lists returned from both sub-partners to the ***calculate offer*** web service together with the transaction ID that was returned to it before by the ***check location*** web service.

The ***calculate offer*** web service uses the transaction ID to identify the proper request of the travel agency and to find the information relating to this reservation in the

data base of the hotel provided there by the ***check location*** web service**.** For instance, information about items found to be available in location and reserved for this request by the ***check location*** web service could be identified by the ***calculate offer*** web service in the course of its processing. Finally, the offer is returned to the ***control*** process and will be passed to a ***check offer*** web service of the travel agency. This web service will return an 'OK' or 'Reject' response to the ***control*** process after having checked whether the offer would be acceptable or rejected.

Upon accepting the offer from the hotel company, the ***bill offer*** web service will invoke the ***charge bill*** web service for the bank to charge the customers the required fees and notify the travel agency of that to finalize the process.

The response from the ***check offer*** web service is passed to a ***create reservation*** web service by the ***control*** process. Depending on the type of response, this web service either completes the reservation processing within the hotel if the response was 'OK' or discards all intermediate information such as items reserved for this transaction ID if the response was 'Reject'.

After the ***create reservation*** web service has terminated its task, it returns a corresponding result to the ***control*** process that, in turn, provides this result to the ***reserve*** web service of the travel agency as a response to its own invocation, thereby completing the workflow of this business process.

Figure 2 (A BPEL Activity diagram for booking a flight) and Figure 3(A BPEL sequence diagram for reserving a room), will cover a small part of the big picture implied in Figure 1 (An Example of collaborative business process). Extending these diagrams on the entire web services provided by Ajiad travel agency will eases the process of listing the threats and vulnerabilities the BPEL have and eases addressing required security considerations.

Figure 10: A BPEL Activity diagram for booking a flight

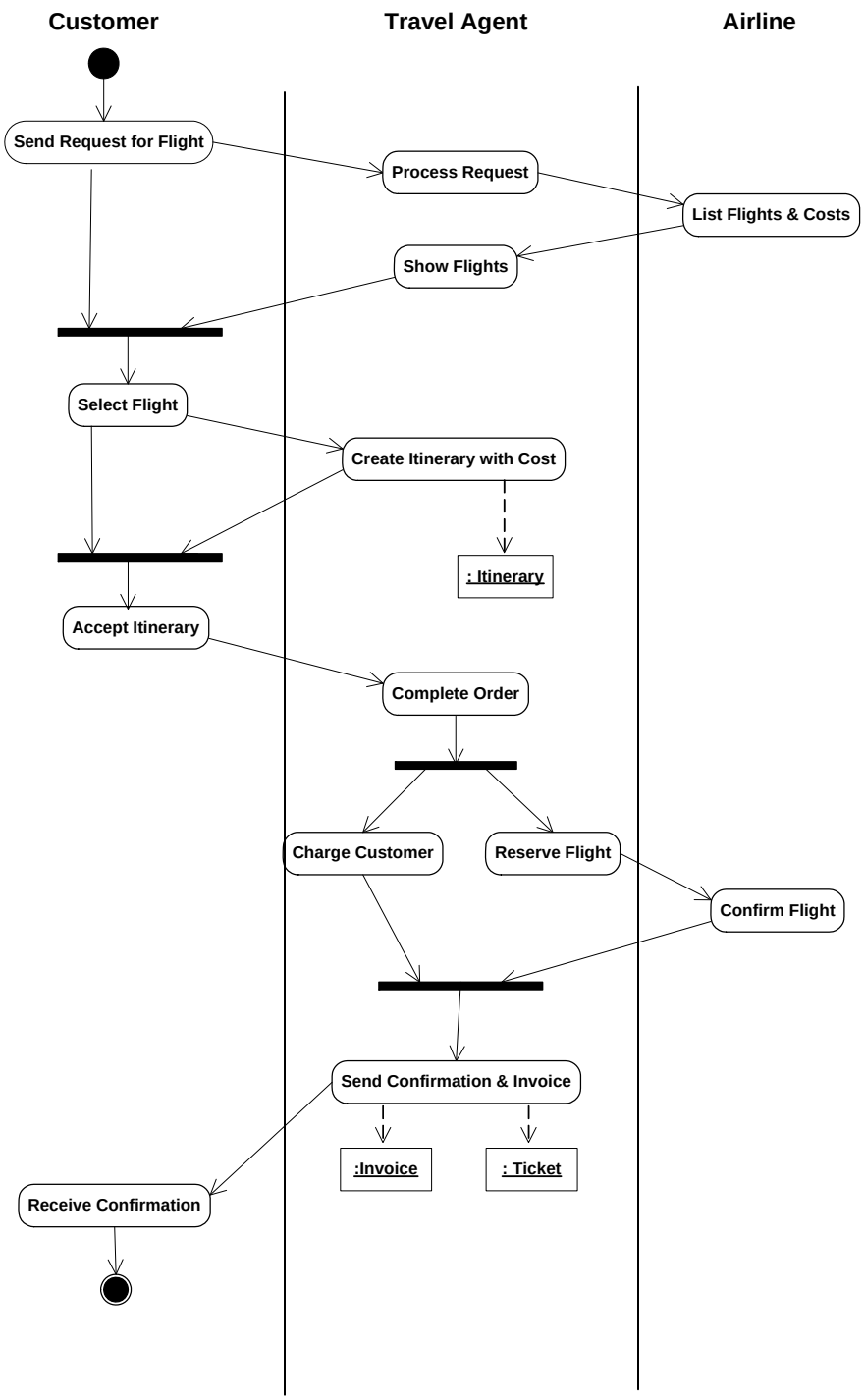
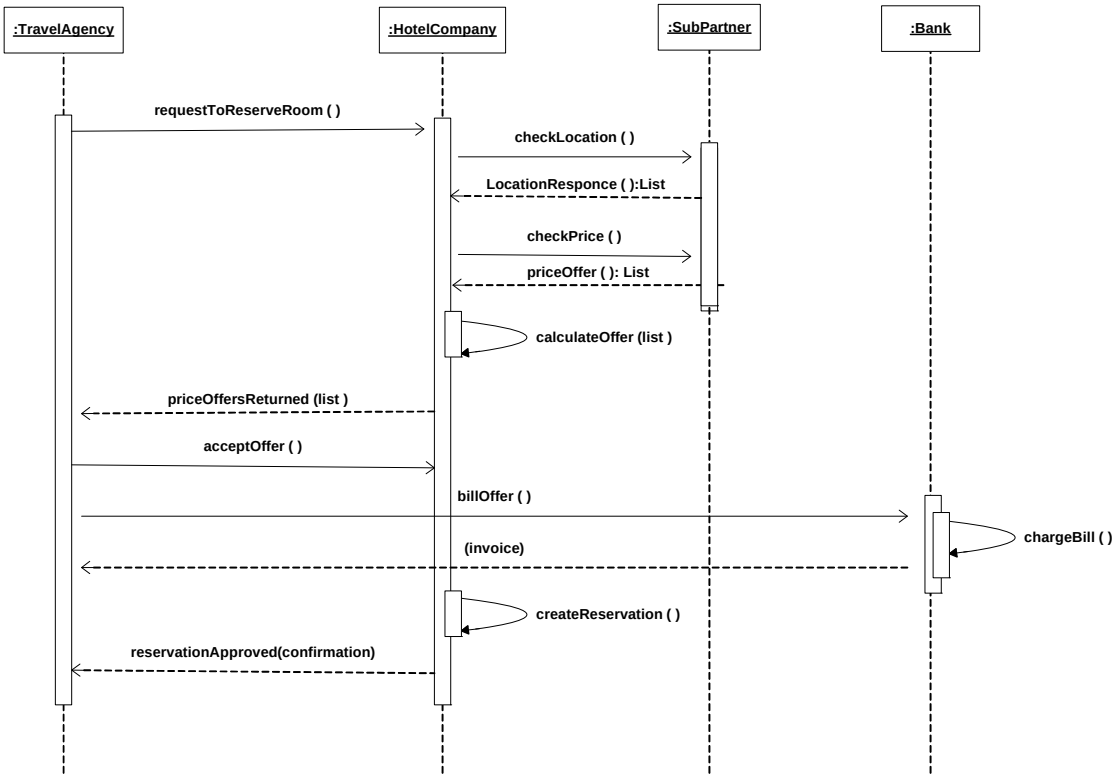


Figure 11: A BPEL sequence diagram for reserving a room



7.3. Composition and Security Specifications

In this section we give a short overview of BPEL and web service specifications related to policies.

BPEL is a workflow-based web service composition language. It specifies the composition as a process, which declares the web services participating in the composition (partners), data containers (variables), and a set of activities with specific

patterns of control and data flow. The building blocks of BPEL processes are activities. There are primitive activities such as <receive>, <invoke> and <replay> and structured activities such as <sequence> and <flow>. Structured activities manage the order of execution of their enclosed activities. BPEL processes can run on any BPEL-compliant orchestration engine. The engine orchestrates the invocations of the partner web services according to the process specification. For illustration, we present a skeleton of the BPEL process that corresponds to the travel agency scenario from Sec. 2. We omitted some parts due to place restrictions.

```
<process name="requesttoReserveRoomProcess"/>

    <partnerlinks>
        <partnerLink name="supplier".../>
        <partnerLink name="bank".../>
        <partnerLink name="HotelCompany".../>
    </partnerlinks>

    <variables>
        …
    </variables>

    <sequence name="Main">
```

```
<receive partnerlink="HotelCompany" operation="order" variable="orderqst"
createInstance="yes" …/>

<invoke partnerlink="supplier" operation="putOrder"
inputvariable="supplyrequest" …/>
…
<invoke partnerlink="bank" operation="pay" inputvariable="payrequest" …/>
…
<reply partnerlink=" HotelCompany" operation="order" variable="clientrspse"
…/>
</sequence>

</process>
```

Listing1. *The requestToReserveRoom* process in BPEL

7.4. Security Considerations

For illustrating the importance of implementing security requirements for web services composition, our considerations are addressed as the following; any operation from the partners (Hotel Company, Bank) should require ***authentication***. Since it's not acceptable for anyone who knows the web service link to have the ability to reserve a room or perform bank transactions. For that reason, the web service composer has to

know the ***security policy for the partner*** services before writing down the BPEL process that will invoke those services. To participate with the BPEL process, the security policy for hotel and bank web services must define which security mechanism (binary certificate, encryption algorithm, digital signature, etc.) they support.

Now the partner web service can verify the identity of the requestor, the next step is to decide what the requestor is allowed to do. For ***non repudiation*** issue, it's important that the office which did the reservation cannot deny do so and that nobody can misuse its identity for malicious activities. To fulfill this requirement, ***digital signature*** can be used. Another issue is ***data integrity***; we need to make sure that when the offices perform the reservation process, it's mandatory that nobody can expose the reservation and change its data. Appropriate mechanisms are also needed to avoid replay attacks, where attackers try to copy the reservation order and resend it again. For that kind of forces, a timestamp mechanism is applicable.

For the bank's payment web service, we consider more about ***confidentiality*** since sensitive data transferred between the travel agency and the bank should not be exposed by any third-part entity. The agreement to choose a key-based encryption and decryption mechanism is a good choice.

To interact with its partners The BPEL process implements three main activities; <invoke>, <receive> and <replay>. ***All the previous security mechanisms mentioned are valid for the <invoke> activity***. But the <receive> and <replay> activities have different

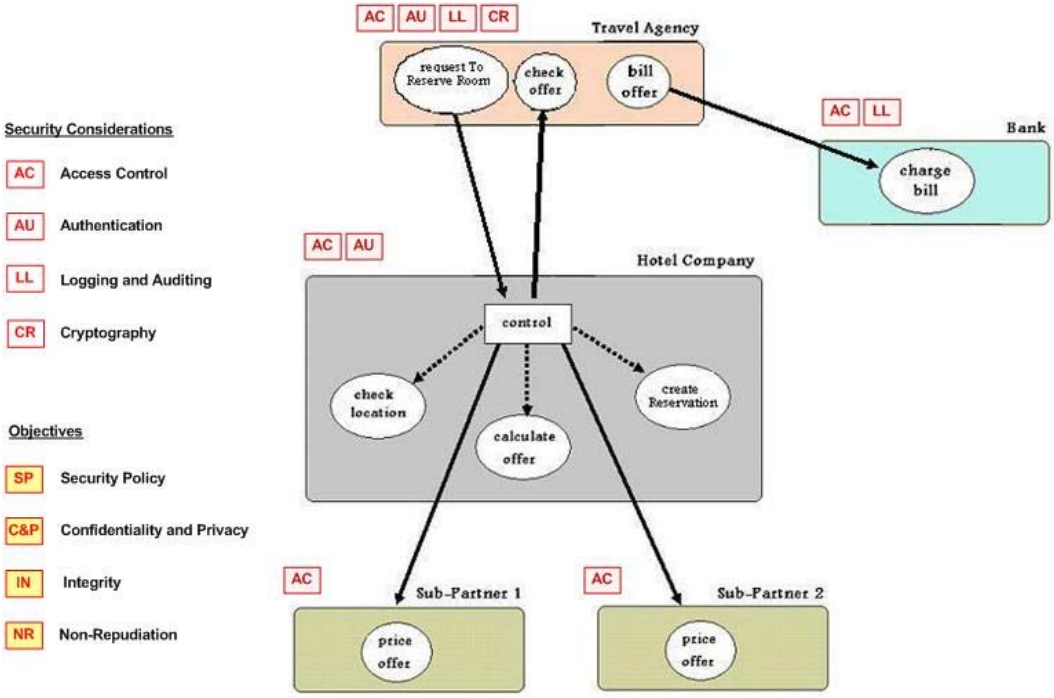
requirements on security specifications. For example; when the BPEL orchestration engine executes the <receive> activity, it will wait until it receive a client request that will match it to start processing. At this point, from the client's perspective, the composite web service is similar to any web service with its own interface. But from the security perspective, it's more about who has the power to invoke the composite web service and what kind of authentication, encryption or signing mechanisms are required by the composite web service. The WS-Policy specification is used to express these security mechanisms and the orchestration engine should enforce it.

To process the <replay> activity, the orchestration engine as required by the security policies has the mission to secure responses in both directions. For example, if a client chooses to use a specific encryption algorithm, then the security framework should encrypt its response with the client's choice of encryption algorithm.

Through its specifications, The BPEL process didn't present any means of security constraints and requirements for a given activity. Since BPEL through its activities tries to provide specific core functional aspects and any non-functional aspects should be considered or addressed by other specifications. If we try to extend BPEL by adding any non-functional features, then this will convert the BPEL to a complex language that is hard to maintain and evolve.

Figure 12 summarizes the discussion and addresses the security considerations for BPEL.

Figure 12: Addressing security considerations for BPEL



7.5. Security Restrictions in Executing Business Process

When the controlling BPEL script is brought in from the travel agency for execution in the domain of the hotel, the processing performed by the controlling business process will be subject to several restrictions derived from the security policies of the hotel.

These security policies may mandate that the list of items that are not in location and, therefore, have to be reserved from the sub-partners may not be disclosed to the travel agency and the respective competing sub-partners for strategic reasons. The same will be true for the credentials required for granting access to the ***fees*** web services of the sub-partners. This information may only be passed to the respective sub-partners, but neither to the competing sub-partner nor to the travel agency for obvious security reasons.

Other restrictions in the example of Figure 1 may require that the list containing prices and reservation dates or the rooms to be reserved returned by each of the sub-partners has to be passed unmodified to the ***calculate offer*** web service, the respective source of the lists may not be confused in order to allow for proper calculation of an offer to the travel agency, and, finally, this information may also not be disclosed to the travel agency or the competing sub-partners. Furthermore, it may be required that the offer containing prices and reservation dates returned from the ***calculate offer*** web service is passed to the ***requestToReserveRoom*** web service of the travel agency without any modification in order to prevent manipulation of this offer.

7.6. Security Issues with Remotely BPEL

Up to the moment, the discussion is focusing on the importance of non-disclosure of information passed between the web services of the control process to destinations outside the domain of the hotel. In our example this process is defined exactly by the travel agency who is restricted to get some of the information of the process. The latter restriction is of special interest where the values originate from other web services may passed to a web service.

Access control to web services and in particular role-based access control (RBAC) may only cover part of these restrictions. In other words, enforcement of nondisclosure of information outside the local domain would imply that access to the particular information would not be granted to any part outside the local domain. In our example, the travel agency is outside the local domain of the hotel and, as a consequence, would not be granted access to restricted information, and the result is we prevent the ***control*** process being remotely defined and deployed by the travel agency.

These restrictions derived from the security policies at the location where a remotely defined BPEL script will be executed can be enforced. Some of which may include that the restriction that the author of the BPEL script is prevented from getting knowledge about information handled by the business process he has specified.

7.7. Related Work

Security requirement analysis of business processes [Her06] introduced a framework that supports domain experts to define security requirements for business processes and to modify business processes in order to guarantee the business process requirements.

Capturing Security Requirements in Business Processes through a UML 2.0 Activity Diagrams Profile [Rod06] shows a micro process through which it is possible to specify and refine security requirements at a high level of abstraction, in a way that they can be incorporated into the development of a software system.

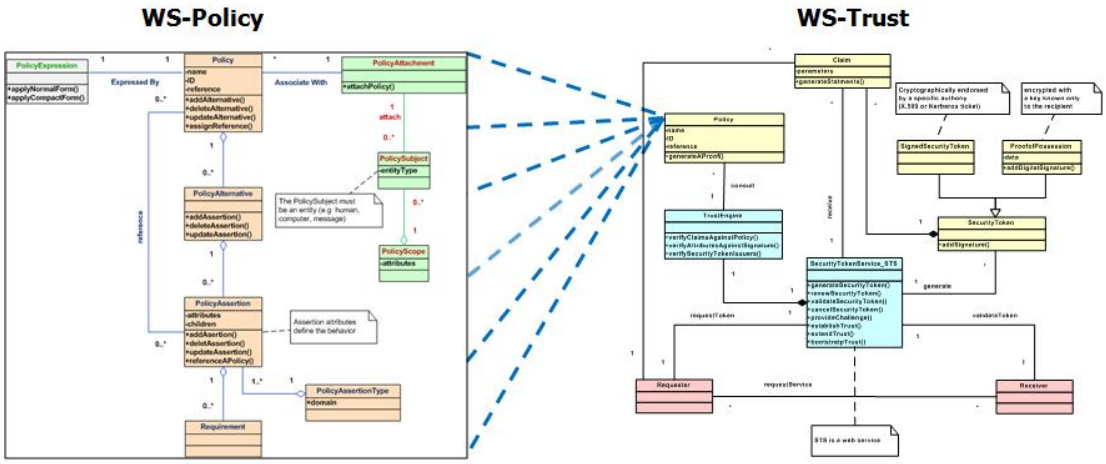
8. CONCLUSIONS AND FUTURE WORK

Our research on WS-Policy and WS-Trust patterns illustrates their value. Without a clear definition of how to use web services, their use could be chaotic. WS-Policy provides means of possible configurations of a web service and enforcing certain level of defined security and authentication. The WS-Trust pattern describes how to build trust relationships and how existing trust relationships may be used as the basis for brokering trust through the creation of security token issuance services. Security token issuance services build on WS-Security to transfer the requisite security tokens in a manner that ensures their integrity and confidentiality.

We wrote new patterns that describe the WS-Policy and WS-Trust standards. Our objective here is to provide an easy way for web services developers to leverage the use of web services standards in web services to make them more usable whilst conforming to industry standards. These patterns offer a detailed and convenient means for developers to include policies and trust aspects in web services.

Figure 13 shows how the patterns we developed in this research fit together and what degree of reusability this relation will provide.

Figure 13: Relation between WS-Policy and WS-Trust



We also analyzed some security aspects of BPEL, the language used to build application workflows. This analysis helped to put in respective the use of the WS-trust and WS-Policy patterns

In rewriting and creating new patterns for WS-Policy and WS-Trust, we observed that many of the patterns could be combined to form new patterns. In combining patterns this could add more useful properties to web services system and that more patterns could be added by examining the degree of need for their application and by identifying their strict relationships with previous patterns.

Additionally, our pattern diagram (Figure 1) highlights patterns that are useful but have not yet been written, and this indicates some of our future work. These patterns include WS-SecureConversation and WS-Federation. Future work will also include the development of further patterns so as to provide the designer with a catalog of patterns that can be used when developing web services-based systems.

For BPEL, our future work will list systematically the possible threats and attacks that could happen in web services workflows in order to define the appropriate and suitable countermeasures to stop or mitigate them

REFERENCES

[Ars01] A. Arsanjani, “Rule Object 2001: A Pattern Language for Adaptive and Scalable Business Rule Construction”, *the 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS39)*, Santa Barbara, California, July29 - August 03, 2001. http://www.computer.org/portal/web/csdl/proceedings/t#4

[Box02] D. Box, *Understanding GXA*, Microsoft Corporation, http://msdn.microsoft.com/en-us/library/aa479664.aspx - Last accessed on December 15, 2009.

[Bus96] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M.(1996). Pattern oriented software architecture: A system of patterns. West Sussex, England: Wiley.

[dat05] IBM Corporation, WebSphere DataPower XML Security Gateway XS40, http://www-01.ibm.com/software/integration/datapower/xs40/ – Last accessed at December, 15, 2009.

[Del05] N. Delessy, and E.B. Fernandez, "Patterns for the eXtensible Access Control Markup Language", in *Proceedings of the 12th Pattern Languages of Programs Conference (PLoP2005)*, Monticello, Illinois, USA, 7-10 September 2005. http://hillside.net/plop/2005/proceedings/ - Last accessed on December 23, 2009.

[Del07a] N. Delessy, E.B. Fernandez, M.M. Larrondo-Petrie, and J. Wu, "Patterns for access control in distributed systems", *Procs. of the 14th Pattern Languages of Programs Conference (PLoP2007)*, Monticello, Illinois, USA, September 5-8, 2007. http://hillside.net/plop/2007/index.php?nav=program - Last accessed on December 31, 2009.

[Del07b] N. Delessy, E.B.Fernandez, and M.M. Larrondo-Petrie, "A pattern language for identity management", Procs. *of the 2nd IEEE Int. Multiconference on Computing in the Global Information Technology (ICCGI 2007),* Guadeloupe, French Caribbean, March 4-9, 2007. http://www.iaria.org/conferences2007/ComICCGI07.html - Last accessed on March 25, 2010.

[Fay04] M.E. Fayad, and H. Hamza, "The Trust Analysis Pattern*", in Proceedings of the Fourth Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP 2004),* Porto Das Dunas, Ceara, Brazil, August 10-13, 2004. http://sugarloafplop2004.ufc.br/acceptedPapers/ww/WW_1.pdf - Last accessed on December 15, 2009.

[Fer01] E.B.Fernandez, and R. Pan, "A pattern language for security models". *The 8th Conference on Pattern Language of Programs* (PLoP2001), Monticello, Illinois, USA September 11-15, 2001. http://jerry.cs.uiuc.edu/~plop/plop2001/ - Last accessed on December 15, 2009.

[Fer06] E.B. Fernandez and N. Delessy, "Using patterns to understand and compare web services security products and standards", *Proceedings of the Int. Conference on Web Applications and Services (ICIW2006*), Guadeloupe, February 2006. IEEE Comp. Society, 2006.

[Fer09] E.B.Fernandez, E. Gudes, and M. Olivier, "The design of secure systems", under contract with Addison-Wesley.

[Gam94] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley, Boston, Mass., 1994.

[Has09a] K. Hashizume, E.B.Fernandez, and S. Huang, "Digital Signature with Hashing and XML Signature patterns", *Procs. of the 14th European Conf. on Pattern Languages of Programs, EuroPLoP 2009.* http://hillside.net/europlop/europlop2009/index.html - Last accessed on November 15, 2009.

[Has09b] K. Hashizume and E.B.Fernandez, "Symmetric Encryption and XML Encryption Patterns", *Procs. of the 16th Conf. on Pattern Languages of Programs (PLoP) 2009.*

[Has09c] K. Hashizume, E.B.Fernandez, and S. Huang, "The WS-Security pattern",

First Int. Workshop on Security Eng.Environments (SEE), Dec. 17-19, Shanghai, China.

[Her06] P. Herrmann and G. Herrmann, "Security requirement analysis of business processes", *Journal of Electronic Commerce Research*, Volume 6, Numbers 3-4, October, 2006, Pages305-335.

[hp09] Hewlett-Packard Development Company L.P., HP SOA Systinet, https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-130-27%5E1461_4000_100__&jumpid=reg_R1002_USEN – Last accessed at December, 15, 2009.

[ibm09a] Security in a Web Services World: A Proposed Architecture and Roadmap, http://download.boulder.ibm.com/ibmdl/pub/software/dw/library/ws-secmap.pdf - Last accessed on December 3, 2009.

[ibm09b] IBM Corporation, Web Services Security 2004, http://www.ibm.com/developerworks/library/specification/ws-secure/ – Last accessed at December, 15, 2009.

[lay09] Layer 7 Technologies, The SecureSpan XML Firewall, http://www.layer7tech.com/main/products/xml-firewall.html – Last accessed on December 09, 2009

[Mad03] Paul Madsen, WS-Trust: Interoperable Security for Web Services, http://www.xml.com/pub/a/ws/2003/06/24/ws-trust.html - Last accessed on November 30, 2009.

[mic06] Michael Cobban, SoftCare EC Solutions, "What is BPEL and why is it so important to my business? ", http://www.softcare.com/whitepapers/wp_whatis_bpel.php - Last accessed at March 9, 2010.

[mic07] Microsoft Corporation, .NET Framework Class Library, http://msdn.microsoft.com/en-us/library/ms951274.aspx – Last accessed at March, 13, 2010.

[Mof88] J.D. Moffett and M. Sloman, "The source of authority for commercial access Control", *Computer*, IEEE, February 1988, 59-69.

[Mor06] P. Morrison and E.B. Fernandez, "The credentials pattern", *in Proceedings of the 2006 conference on Pattern languages of programs (PLoP 2006)*, Portland, OR, USA. October 21–23, 2006. http://portal.acm.org/citation.cfm?id=1415472.1415483 - Last accessed on December 15, 2009.

[oas06] OASIS, Web Services Security: (WS-Security 2004), http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf - Last accessed on December 15, 2009

[oas07] OASIS, WS-BPEL TC, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf - Last accessed at March 5, 2010.

[oas09a] OASIS Standard, WS-Trust 1.4, http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/os/ws-trust-1.4-spec-os.pdf - Last accessed at February 9, 2010

[oas09b] OASIS Standard, WS-Federation 1.2, http://docs.oasisopen.org/wsfed/federation/v1.2/ws-federation.pdf - Last accessed at February 19, 2010.

[pin06] Ping Identity Corporation, PingTrust, a standalone Security Token Server, http://www.pingidentity.com/about-us/news-press.cfm?customel_datapageid_1173=1404 - Last accessed on December 15, 2009.

[Rod06] A. Rodriguez, E. Fernandez-Medina, and M. Piattini, "Capturing Security Requirements in Business Processes Through a UML 2.0 Activity Diagrams Profile", Advances in Conceptual Modeling - Theory and Practice, Springer Berlin / Heidelberg, October, 2006, Pages 32-42.

[Sch06] M. Schumacher, E.B. Fernandez, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security Patterns: Integrating security and systems engineering*, Wiley 2006.

[Sch83] Schlichting, R. D. & Schneider, F. B. (1983). Fail-Stop processors: an approach to designing fault tolerant computing systems. ACM Transactions on Computing Systems, 1(3), 222-238.

[sof09] Software AG, Application Modernization, http://www.softwareag.com/corporate/products/wm/am/default.asp – last accessed at March, 25, 2010.

[ssr10] Secure Systems Research Group (SSRG), Florida Atlantic University, http://www.security.ceecs.fau.edu/ - last accessed at February, 21, 2010.

[pin06] Ping Identity Corporation, PingTrust, a standalone Security Token Server, http://docs.oasisopen.org/wsfed/federation/v1.2/ws-federation.pdf - Last accessed on December 15, 2009.

[vmw09] VMware, Inc., SecureSpan XML Virtual Appliance, http://www.vmware.com/appliances/directory/249773 – Last accessed at December, 15, 2009.

[vor09] Vordel Limited, Vordel STS, http://www.vordel.com/solutions/security_token_services.html - Last accessed on December 15, 2009.

[w3c02a] W3C, XML Signature Syntax and Processing, February, 2002, http://www.w3.org/TR/xmldsig-core/ – Last accessed at March 9, 2010.

[w3c02b] W3C, XML Encryption Syntax and Processing, December, 2002, http://www.w3.org/TR/xmlenc-core/ - – Last accessed at March 9, 2010.

[w3c04] W3C Working Group Note 11, February, 2004, http://www.w3.org/TR/ws-gloss/ – Last accessed at April 21, 2010.

[w3c07a] Web Services Policy 1.5 - Framework, September, 2007, http://www.w3.org/TR/ws-policy/ – Last accessed at December, 15, 2009.

[w3c07b] Web Services SecurityPolicy 1.2, July, 2007, http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.2/ws-securitypolicy.pdf – Last accessed at March 9, 2010.

[w3c08] W3C Working Group, XML Signature Syntax and Processing (Second Edition) 2008, http://www.w3.org/TR/ws-gloss/ – Last accessed at December, 15, 2009.

[w3c09] W3C Working Draft 2009, Web Services Metadata Exchange, http://www.w3.org/TR/ws-gloss/ – Last accessed at December, 15, 2009.

[wik09] Wikipedia, Wikimedia Foundation, Inc. Business Process Execution Language, http://en.wikipedia.org/wiki/Business_Process_Execution_Language – Last accessed at March 9, 2010

[xtr05] PrismTech Corporation, Xtradyne's WS-DBC - the XML/SOAP Firewall for Enterprises, http://www.xtradyne.com/products/ws-dbc/ws-dbc.htm – Last accessed at December, 15, 2009.