Future Internet **2012**, *4*, 430-450; doi:10.3390/fi4020430

OPEN ACCESS

future internet

ISSN 1999-5903

www.mdpi.com/journal/futureinternet

Article

A Survey of Patterns for Web Services Security and Reliability Standards

Eduardo B. Fernandez *, Ola Ajaj, Ingrid Buckley, Nelly Delessy-Gassant, Keiko Hashizume and Maria M. Larrondo-Petrie

Florida Atlantic University, 777 Glades, Boca Raton, FL 33431, USA; E-Mails: oajaj@fau.edu (O.A.); ibuckley@fau.edu (I.B.); gassantn@gram.edu (N.D.-G.); ahashizu@fau.edu (K.H.); petrie@fau.edu (M.M.L.-P.)

* Author to whom correspondence should be addressed; E-Mail: ed@cse.fau.edu; Tel.: +1-561-297-3466; Fax: +1-561-297-2800.

Received: 16 January 2012; in revised form: 26 March 2012 / Accepted: 16 April 2012 / Published: 20 April 2012

Abstract: An important aspect for the acceptance of Service-Oriented Architectures is having convenient ways to help designers build secure applications. Numerous standards define ways to apply security in web services. However, these standards are rather complex and sometimes overlap, which makes them hard to use and may produce inconsistencies. Representing them as patterns makes them easier to understand, to compare to other patterns, to discover inconsistencies, and to use them to build secure web services applications. Security patterns abstract the key aspects of a security mechanism and can thus be applied by non-experts. We survey here our work on security patterns for web services and their standards and we put them in perspective with respect to each other and to more fundamental patterns. We also consider other patterns for web services security. All the patterns described here have been previously published, we only show here one of them in detail as an illustration of our style for writing patterns. Our main purpose here is to enumerate them, show their use, and show how they relate to each other.

Keywords: web services security; web services standards; security patterns; secure distributed systems; secure SOA; misuse patterns

---

1. Introduction

Service-Oriented Architectures (SOA) and web services are special cases of distributed systems. Distributed systems are typically heterogeneous systems that are accessible to a wide variety of users, including institution partners, customers, or mobile employees, which introduces a large variety of security threats. To protect its assets, an organization needs to define security policies, which are high-level guidelines that specify the states in which the system is considered to be secure. These policies need to be enforced by security mechanisms. In large organizations, the policies may be issued by different actors making their management difficult. Moreover, they need to be enforced for a variety of resources. To make things more difficult they may have to follow regulations. A way to allow interoperability and apply security is the use of standards which define architectures to enforce that all participants will follow the same rules in their interactions.

Security mechanisms and standards can be conveniently described using security patterns. A *security pattern* extends the idea of design pattern [1] to describe security mechanisms that can handle some threats [2]. The use of patterns is very convenient for software developers who need to add security to their applications but are not experts on security. Security patterns abstract the key aspects of a security mechanism and can thus be applied by non-experts. They can be used also for guiding product developers who need to follow specific standards. Another use is for evaluating existing systems. We have also found them very useful to teach security concepts. A good catalog is fundamental for the use of patterns and it is important to add more patterns to the existing collections [2,3]. Web services is an area where patterns have proven useful. In particular, there are many web services security standards, which are rather complex and sometimes overlap; representing them as patterns makes them easier to understand and to compare to other patterns. We survey here our work on security patterns for web services and their standards and we put them in perspective with respect to each other and to more fundamental patterns. Our objective here is not to describe each pattern in detail, for that the reader is directed to the references, but to show which patterns exist and how they relate to each other. We are not presenting here new patterns but surveying and evaluating our previous patterns.

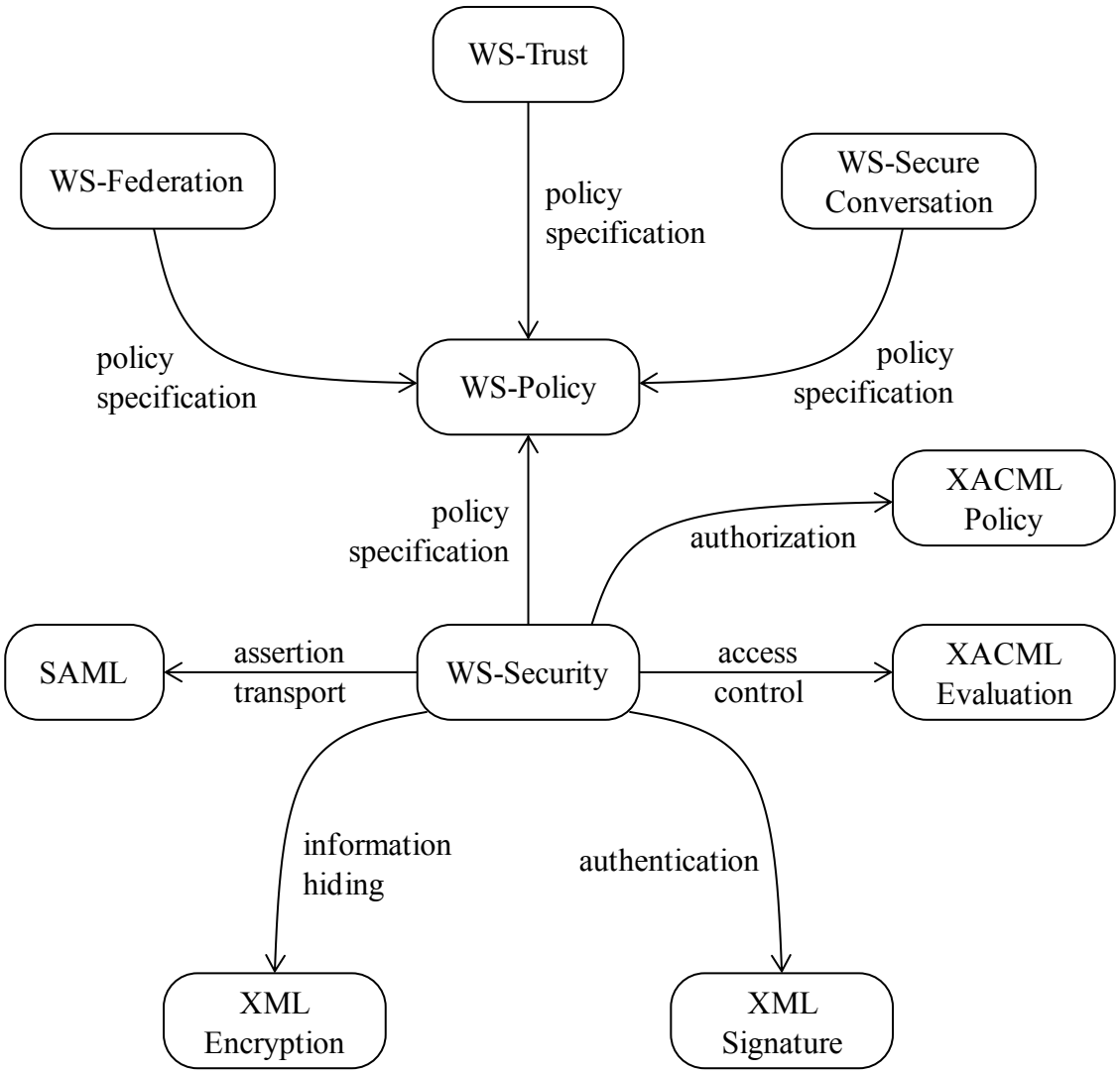
The patterns presented here are part of an ongoing catalog of security patterns [3] and we relate them to other patterns using pattern diagrams. As far as we know, ours is the only catalog of web services security standards. However, pattern catalogs are not very useful without a way to apply them to build secure systems. We have developed a general secure systems development methodology [4], which in principle applies also to web services; we specialized it for SOA [5].

As indicated, web services standards are rather complex and verbose and it is not easy for designers and users to understand their key points. Web services standards are typically long documents, e.g., the XACML 3.0 Core Specification is 150 pages long, written to be comprehensive but not easy to understand, and using a combination of XML, UML, and natural language. 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 [6]. Inversely, a product vendor can use the standards to guide the development of the product. By expressing standards as patterns, we can compare them and understand them better. For example, we can discover overlapping and inconsistent aspects between them. A standard defines a generic architecture and this is a basic

feature of any pattern; it can then be confirmed as a best practice by looking at products that implement the standard (and implicitly the pattern). There are many security standards for web services [7] defined by several committees, including W3C, OASIS, and IETF.

Figure 1 shows a pattern diagram describing the relationships between the patterns for web services security standards. Pattern diagrams such as this one were introduced in [8]. In fact, we also use their template to describe our patterns. The rounded squares represent patterns, while arrows indicate what is brought to a pattern by the pattern at the other end (point of the arrow); for example, WS-Security uses policies specified by WS-Policy. Our group has written patterns for all these standards, except WS-Secure Conversation and WS-Federation, which are ongoing work. We do not know of any other patterns for web services security standards.

Figure 1. Pattern diagram for web services security standards.



The patterns described here are specialized versions of more fundamental and more general patterns. For example, XACML is a specialization and extension of the Authorization pattern [9]. As such it carries the general properties of an Authorization pattern and adds aspects specific to XML access control. The new aspects may themselves be patterns, e.g., the Composite pattern [1] appears

frequently in these models to indicate recursive composition. Identifying patterns as part of a more complex pattern makes it easier to understand the functions of the complex model.

We introduce in each section a short summary of the patterns and we indicate where the complete pattern has been published; all of them will be collected in [3]. Section 2 presents fundamental patterns for access control, necessary to understand the more specialized patterns. Section 3 considers access control and policies for web services. Section 4 discusses patterns for cryptographic standards, starting again from fundamental patterns. Section 5 considers identity in distributed systems, and Section 6 looks at wireless web services security patterns. Section 7 is about web services reliability patterns while Section 8 discusses misuse patterns for web services. A misuse pattern describes a possible attack. Section 9 shows a complete pattern, WS-Trust, to illustrate one of our patterns, Section 10 considers related work, while some conclusions and possible future work are given in Section 11.

2. Fundamental Patterns for Access Control

Access control models generally represent a few types of security policies and provide a formalization of these policies using some ad hoc notation. Four basic access control models are commonly used and they may be extended to include content and context-based access control, delegation of rights, hierarchical structurings of subjects (including roles), objects, or access types [10], temporal constraints, *etc.* Access control models can be defined for different architectural levels, including application, database systems, operating systems, and firewalls [11].

Access control models fall into two basic categories: mandatory models, where users' rights are defined by administrators and data may be labeled to indicate its sensitivity, and discretionary, where users administer the data items they create and own. Within this classification, there are several models for access control to information that embody general (application independent) policies. The most common are:

- The *Multilevel model* organizes the data using security levels. This model is usually implemented as a mandatory model where its entities are labeled indicating their levels. There are separate models for confidentiality and integrity and accesses are decided by enforcing two principles or generic rules. This model is able to reach a high degree of security, although it can be too rigid for some applications. Usually, it is not possible to structure the variety of entities involved in complex applications into strictly hierarchical structures. However, they can be useful for structuring the architecture of systems.
- The *Access Matrix* describes access by subjects (actors, entities) to protected objects in specific ways (access types) [10]. It is more flexible than the multilevel model and it can be made even more flexible and precise using predicates and other extensions. However, it is intrinsically a discretionary model in which users own the data objects and may grant access to other subjects. It is not clear who owns the medical or financial information and the discretionary property reduces security because it may not be possible to decide who will acquire a specific right. This model is usually implemented using Access Control Lists (lists of the subjects that can access a given object) or Capabilities (tickets that allow a process to access some objects), described in two patterns:

- **Access Control List [12]**: controls access to objects by indicating which subjects can access an object and in what way. There is usually an ACL associated with each object.
- **Capability [12]**: controls access to objects by providing a credential or ticket to be given to a subject for accessing an object in a specific way.

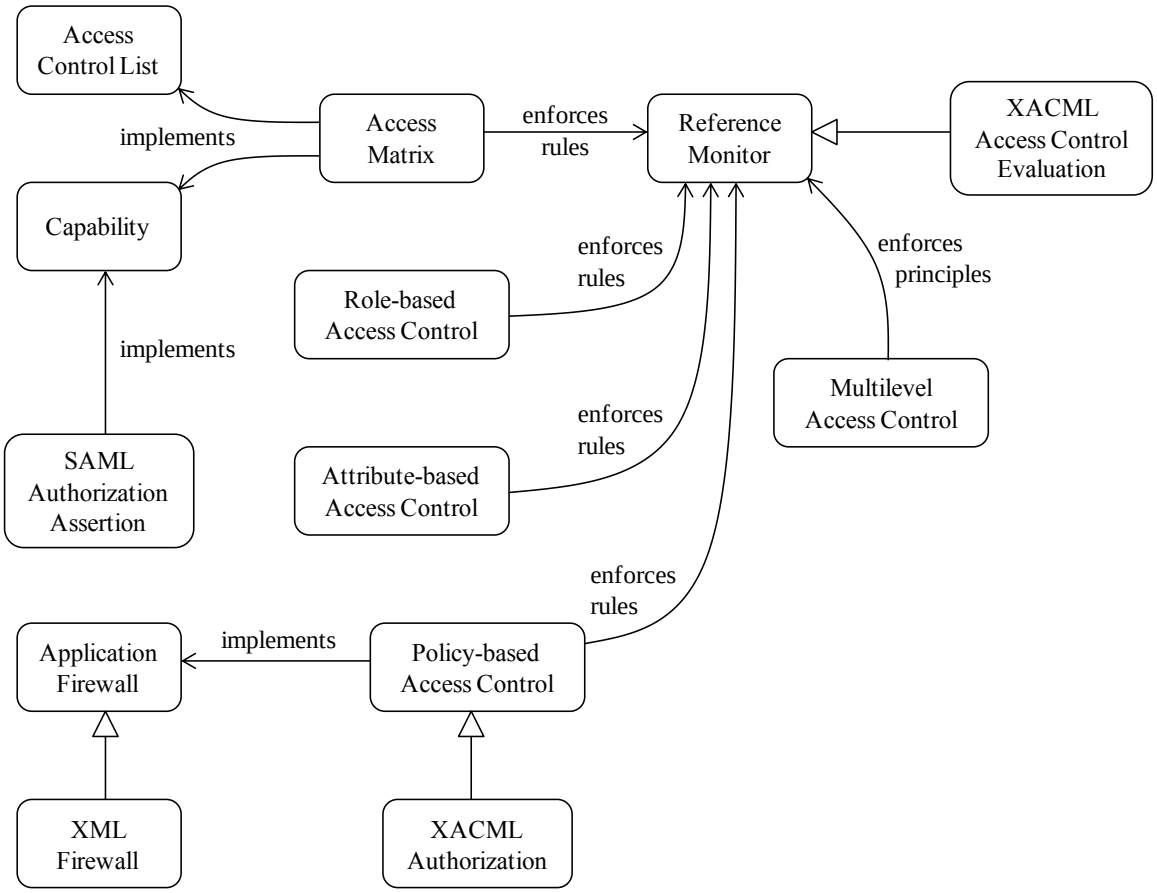
- *Role-Based Access Control* (RBAC), collects users into roles based on their tasks or functions and assigns rights to each role. Some of these models have their roles structured as hierarchies, which may simplify administration.
- *Attribute-Based Access Control* (ABAC). This model controls access based on properties of subjects or objects. It is used in environments where some or all subjects may not be pre-registered and where the effect of the context is important to decide access.

These models have different ways of expressing their access constraints but they use a similar abstract concept of *Reference Monitor* to enforce them. The Reference Monitor intercepts every access request and decides access based on the specific rules that apply to the request.

We have presented patterns for all these models [2,3,13] and for the Reference Monitor [2,3]. We can generalize them into a model that includes a set of policies enforced by a common Reference Monitor:

- **Policy-Based Access Control [12]**: models how to decide if a subject is authorized to access an object according to policies defined in a policy repository.

Figure 2. Pattern diagram for access control.



Standard models, such as the Access Matrix and RBAC (Role-Based Access Control), are represented in Figure 2, along with Attribute-Based Access control and Policy-Based Access control. The two latter models are more suitable in the case of distributed systems. All of the models use a Reference Monitor to enforce access decisions. ACL (Access Control List) and Capability are implementation-oriented patterns; they implement the Access Matrix or RBAC model. More specifically for web services, XACML (eXtensible Access Control Markup Language) Access Control Evaluation implements the Reference Monitor and the Policy-Based Access Control pattern, and the XACML Policy Language implements the Policy-Based Access Control pattern. SAML Authorization Assertion is a kind of Capability. The corresponding web services security patterns are shown in Section 3.

3. Access Control and Policies for Web Services

For distributed systems we can enforce policies by controlling inputs and outputs to the application. This leads to the Application Firewall. Similarly, we can control the sending and receiving between web services of XML documents using an XML Firewall.

Application Firewall [14]: The application firewall filters calls and responses to/from enterprise applications, based on an institution access control policies. It can be considered a concrete form of the Reference Monitor.

XML Firewall [14]: Filters XML messages to/from enterprise applications, based on business access control policies and the content of the message. It is even a more specialized version of the Reference Monitor.

Our pattern on Policy-Based Access control described above incorporates the concepts of distributed authorization at an abstract level. We also produced specific patterns for the two access control aspects of XACML, as well as their specialization for web services:

XACML Authorization [15]: XACML describes authorization rules that can be composed in a variety of ways.

XACML Access Control Evaluation [15]: This pattern decides if a request is authorized to access a resource according to policies defined by the XACML Authorization pattern; that is, it is a Reference Monitor for XACML.

More specific patterns for access control are defined for web services:

WSPL [15]: Enables an organization to represent access control policies for its web services in a standard manner. It also enables a web services consumer to express its requirements in a standard manner.

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, including security, reliability, and others. WS-Policy also provides a way to check the requests made by requestors in order to verify that they satisfy their assertions and their conditions before interacting with the web service:

WS-Policy [16]: Describes how to express requirements that are needed or supported by a web service. For instance, it can indicate that a specific signature algorithm must be used when adding a digital signature.

Web services interact with users and other web services. Sometimes users and web services are not predefined and known to each other and a trust relationship must be established before any interaction happens between the participants. This relationship can be defined by exchanging security tokens such as certificates or other proofs of identity or attributes. WS-Trust is a standard to support the establishment of trust relationships between web services. Trust depends also on reputation but this is an aspect not included in the standard. Trust is based on security and other policies to enable requesting and obtaining credentials within different trust domains. Both parties need to determine if they can "trust" the asserted credentials of the other party. The goal of the WS-Trust standard is to enable applications to construct trusted message exchanges. This trust is realized through the exchange and brokering of security tokens:

WS-Trust [17]: Provides a framework for requesting and issuing security tokens, and to broker trust relationships [18]. It uses WS-Security to transfer the required security tokens, using XML Signature and Encryption to provide confidentiality. This standard may use WS-Policy to specify which security tokens are required at the target.

Other patterns for distributed security include:

SAML [18]: Defines a standard protocol to exchange authentication and authorization assertions. It may use WS-Security standard to protect assertions while they are being transmitted.

WS-Federation: Defines mechanisms to allow different security domains to federate [19]. It describes how federated trust scenarios can be constructed using WS-Security, WS-Policy, WS-Trust, and WS-SecureConversation

WS-SecureConversation: Defines mechanisms to allow security context establishment and sharing, and session key derivation [20]. This specification uses WS-Security, WS-Trust and WS-Policy to negotiate and issue session keys.

4. Cryptography for Web Services

WS-Security is a standard that describes how messages using the SOAP protocol can have integrity, authentication, and confidentiality. WS-Security is a flexible protocol that supports different formats of authentication security tokens, different encryption technologies, and different signature formats. WS-Security does not define new security mechanisms, but it leverages existing standards such as XML Encryption, XML Signature, and Security Tokens e.g., Kerberos Tickets and X.509 certificates. We have described this standard in the form of a pattern:

WS-Security [21]: Defines how to secure SOAP messages applying XML security technologies such as XML Encryption and XML Signature. It also defines how to embed different security tokens. Security tokens provides authentication by proving one's identity (certificates or SAML assertions are examples).

Web services that exchange XML messages can be target of attacks. Some security standards have been developed to apply mechanisms that reduce security risks, one of these is. XML Signature. This standard is a joint effort between the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (ITEF). XML Signature defines how to digitally sign an entire XML message, part of an XML message, or an external object. XML Signature also includes hashing, but the pattern name follows the name of the standard. Because XML documents can have the same contents but in

different layouts, we need to convert the documents into a canonical form before we apply digital signatures. Note that XML Signature solves the same problem as the Digital Signature with Hashing pattern but in a more specialized context. The XML Signature pattern, a specialization of the Digital Signature with Hashing, is used to secure XML messages. We first defined three basic patterns to describe some cryptographic aspects:

Symmetric Encryption [22]: protects message confidentiality by making a message unreadable to those that do not have access to the key. Symmetric encryption uses the same key for encryption and decryption.

Digital Signature with Hashing [23]: allows a principal to prove that a message was originated from it. It also provides message integrity by indicating whether a message was altered during transmission.

XML Encryption [22]

This standard describes a process to apply encryption functions to data, keeping a correct XML syntax.

XML Signature [23]

Allows a principal to prove that a message was originated from it. It also provides message integrity by defining whether a message was altered during transmission. The XML Signature standard [24] describes the syntax and the process of generating and validating digital signatures for authenticating XML documents. XML Signature also provides message integrity. It requires canonicalization before hashing and signing.

5. Patterns for Identity [25]

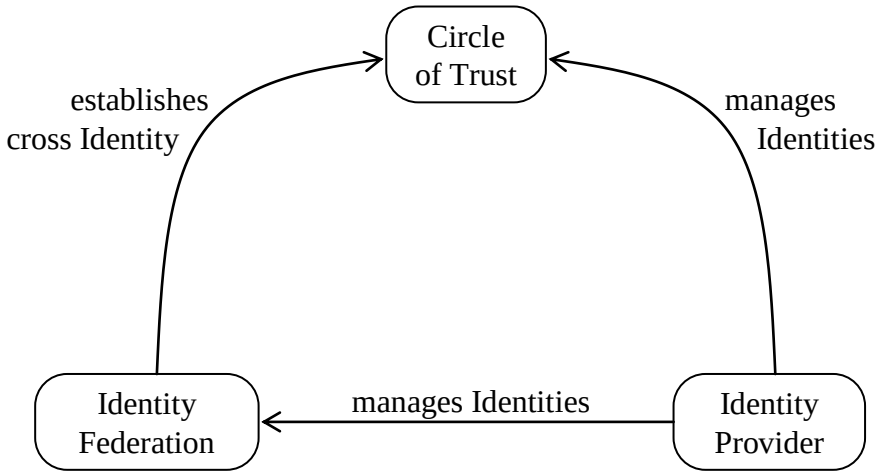
A large amount of work has been done about the propagation of identity information. In particular, web services standards have been published that deal with identity management and trust. We add support to the traditional models by defining architectural patterns for identity management. These patterns can then be used directly in the software development cycle, as proposed by different methodologies. Figure 3 relates our patterns for identity management, which include:

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

Identity Provider: The Identity Provider pattern allows the centralization of the administration of subjects' identity information for a security domain.

Identity Federation: The Identity Federation pattern allows the formation of a dynamically created identity within an identity federation consisting of several service providers. Therefore, identity and security information about a subject can be transmitted in a transparent way for the user among service providers from different security domains.

Figure 3. Relationships between identity patterns.



6. Wireless Web Services

Web services are also becoming important for use in mobile devices. The concept of dynamic access to web services allied with the flexibility of wireless accesses makes it possible to envisage a new type of applications, where the mobility of the user supplies the application with context elements. Possible applications include disaster management, location services, advertising, *etc.* Web services security standards are used for the secure design of the communications between a web service and a mobile client and for the storage of the web service and its data. However, because those standards are designed to be flexible, they are also complex and verbose. On their part, wireless devices have specific technological constraints as well as their own standards. To use web services standards in mobile devices, it is necessary to adapt these standards to consider the limitations of portable devices. We have shown the use of patterns as a way to adapt web services security standards to the wireless environment [26]. We have proposed two approaches for adapting standards to wireless systems. We showed the use of XACML for standard subsetting, where parts of the standard which are considered too wasteful of resources or unnecessary for this environment are left out. Another approach is to start from the functions themselves and see which ones are the ones needed for some targeted applications and adapt the standard to these functions. As an illustration for this latter approach the PAOS pattern, is a wireless-oriented version of the traditional HTTP binding for SOAP for web services.

Liberty Alliance PAOS Service [26]: An adapted version of the traditional HTTP binding for SOAP, suitable for mobile devices.

It is important that wireless devices use security models based on standards. This approach allows the incorporation of wireless devices into a unified enterprise policy. In this way, policies for wireless devices can be designed as part of the overall design as we do in our methodology [4], and not in an ad hoc way. Approaches that use separate policy models for wireless devices result in systems that will not be interoperable with other systems.

7. Patterns for Web Services Reliability

The WS-Reliability and WS-Reliable Messaging Standards are defined by OASIS and the former has borrowed from the ebXML Message Service Specification 2.0 technology. WS-Reliability is a

SOAP-based specification that fulfills reliable messaging requirements critical to some applications of web services [27]. The WS-Reliability standard utilizes quality of service (QOS) contracts, and uses conditions attached to the invocation of a set of operations; namely deliver, submit, respond and notify [Oas04]. To perform reliable delivery it uses the concept of Reliable Message Processor (RMP). The WS-Reliable Messaging standard provides guaranteed delivery, message ordering and duplicate elimination [28]. To support interoperable web services, a SOAP binding is defined within this specification. However the protocol depends upon other web services specifications for identification of service endpoint addresses and policies [28].

WS-Reliability and WS-Reliable Messaging specifications offer the same basic service, which is sending messages in a reliable manner. However, the two protocols utilize different means of performing this service. WS-Reliability has a binding to HTTP whereas WS-Reliable Messaging is transport independent allowing it to be implemented using different network technologies. In order to support interoperable web services, a SOAP binding is defined within both patterns. The specifications mandate that an agreement be made before communication can be done between endpoints. However the WS-Reliable Messaging explicitly states that endpoint referencing, establishment of trust and policy exchange are to be included in the agreement. Endpoint reference explicitly states the address where a reliable message should be sent. Establishment of trust is achieved with an enforced agreement and policy exchange facilitates the updating of quality of service terms and conditions. WS-Reliability does not explicitly dictate the terms of the contract.

WS-Reliability [29]: Ensures that a notification is always sent in response to a failure, it also provides guaranteed message delivery, message ordering, and duplicate elimination whenever messages are sent from one entity to another.

WS-Reliable Messaging [29]: Ensures guaranteed receipt in response to each message sent; it also provides, message state disposition, ordered delivery, and duplicate elimination whenever messages are sent between endpoints.

8. Misuse Patterns

We introduced the concept of misuse patterns in [30]. They describe from the point of view of the attacker, how a type of attack is performed (what system units it uses and how), proposes ways of stopping the attack by enumerating possible security patterns that can be applied for this purpose, and helps analyze the attack once it has happened by indicating where we can find forensics data as well as what type of data. We have published several misuse patterns for different types of attacks. One of them is for a common attack in web services:

Spoofing web services [31]: A web service spoofing misuse tries to impersonate the identity of a user, and then with the user ́s credentials makes requests in his name, with the intention of accessing specific web services.

9. A Complete Pattern: WS-Trust

9.1. 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, applications can engage in secure communication after establishing trust.

9.2. 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. *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 a partner is trusted or not, 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.

9.3. 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 does not necessarily correspond 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.

9.4. 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 security 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 contains 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. This implies that the policy itself should be protected.
- ***Message integrity***: The data to be transferred between the partners through messages may be private data that need to be protected. Attackers may try to modify or replace these messages.
- ***Time Validity***: For protection purposes, any interactions or means of communications (including the trust relationships) between the web services should have a time limit, that determines for how long the trust relationship is valid.

9.5. Solution

We define explicitly an artifact (security token) that implies trust. This artifact implies what kinds of assertions are required to make trustworthy interactions between the involved web services. We should verify the claims and information sent by the requester in order to obtain the required security token that becomes a proof enough to establish a trust relationship with its target partners.

9.5.1. Structure

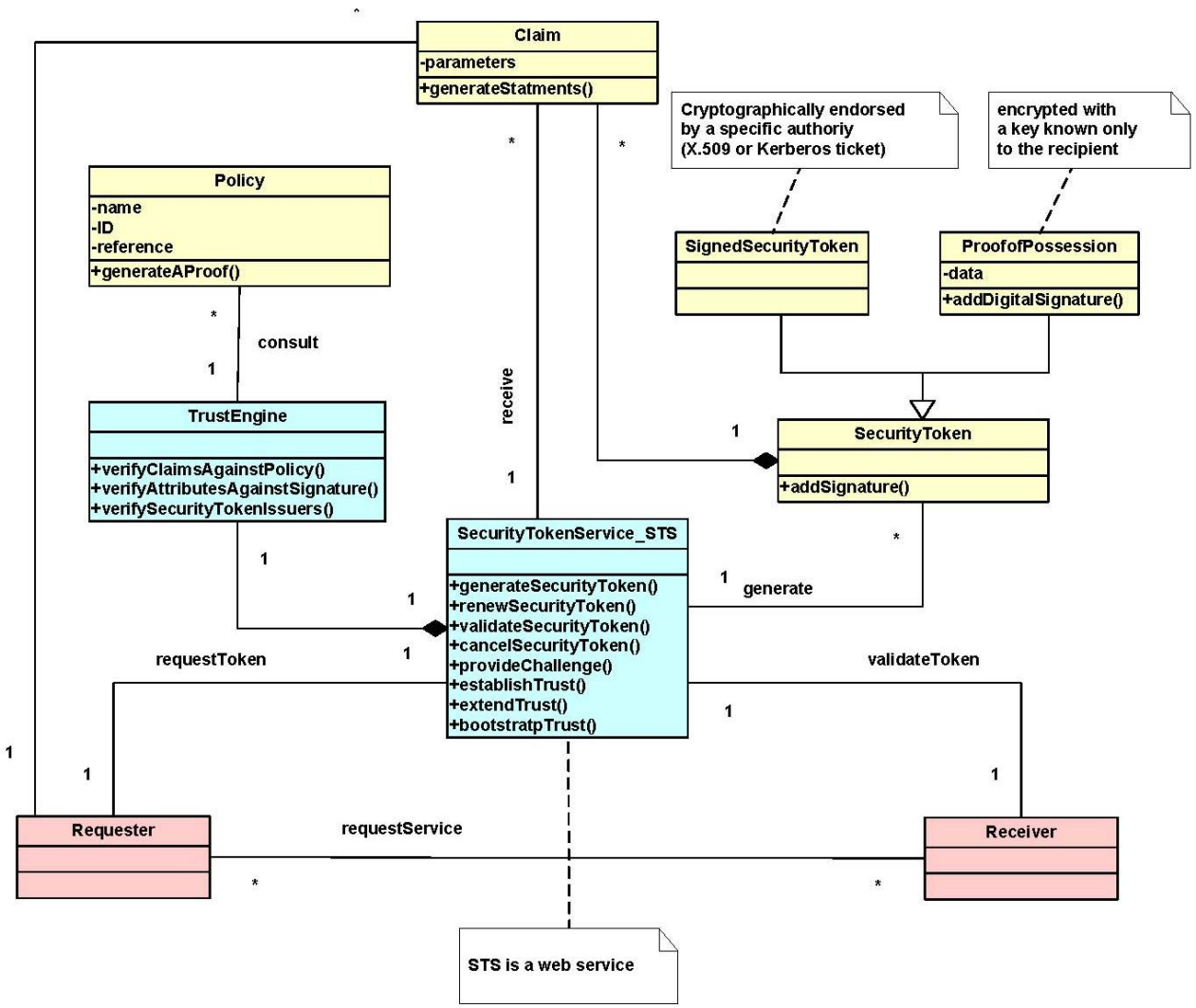
Figure 5 describes the structure of this pattern. **Claim** is a statement made about the attributes of a client, service or other resource (e.g., name, identity, key, group, privilege, capability, *etc.*). Claims are assertions, for example: "I am Joman", "I am an authenticated user and I am authorized to print in printer P". Claims are used to validate the requests made by a sender and need to be verified. 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.

The **Security Token Service (STS)** is a web service that issues security tokens. It makes decisions based on evidence that it trusts. The **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 expressed as a security token. 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.

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 assertions 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 5. Class Diagram for the WS-Trust Pattern.



9.5.2. 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 (Figure 6):

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:

a. The requester requests a security token by sending the required *claims* and *RequestType* signed by a *Signature* to the STS. The signature verifies that the request is legitimate.
b. The STS contacts the Trust Engine to check the requester's claims.

c. The Trust Engine contacts the web service's policy to verify the claims including attributes and security token issuers of the requester.
d. Once approved, the STS creates a security token containing the requested claims.
e. 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.

Access a resource using a token (Figure 7):

Summary: A STS allows the use of resources by establishing 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 notifies the receiver that the security token is valid and verified.
e. The receiver gives the requester a token that implies 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 6. Sequence Diagram creating a security token.

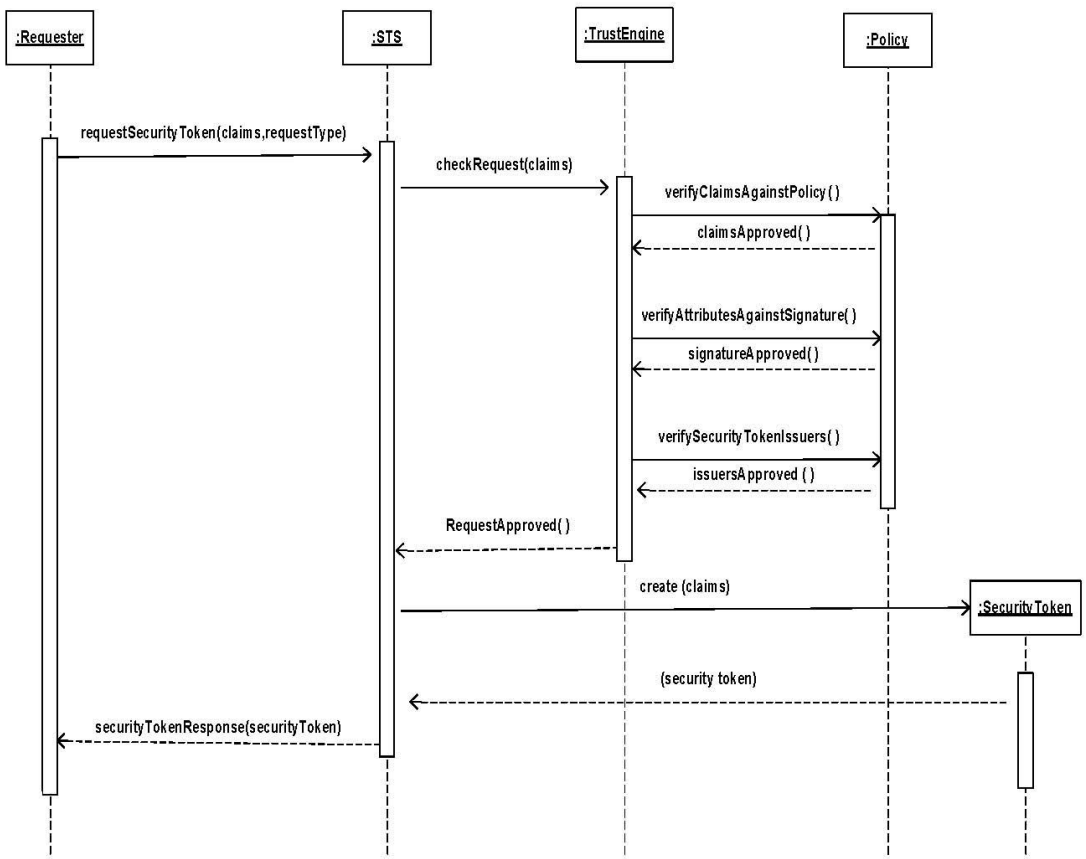
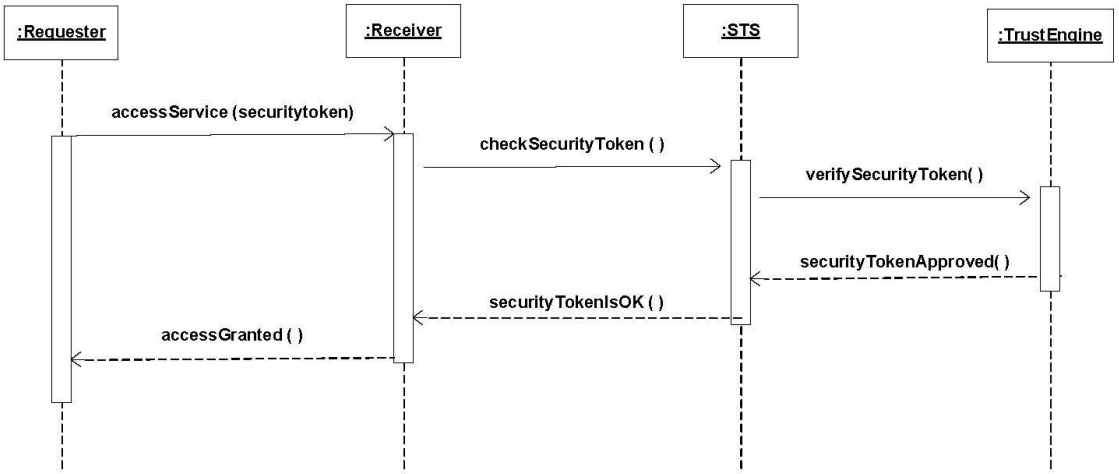


Figure 7. Sequence Diagram accessing a resource using a token.



9.6. 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 approval, the receiver establishes a valid trust relationship with the receiver that lasts as long as the security token is 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 aspects 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).*This is our use case Create a security token in the Dynamics section.
  - The renewal process: formed as *RequestSecurityToken (RequestType, RenewTarget).*
  - The cancel process: formed *RequestSecurityToken (RequestType, CancelTarget).* By the way, the cancelled *token is no longer valid for authentication and* authorization.
  - The validate process: formed as *RequestSecurityToken (RequestType, ValidateTarget).*

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 [32]. It was authored by Microsoft, IBM, Verisign, and RSA Security and was approved by OASIS as a standard in March 2007.

9.7. 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 trying 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.

9.8. Known Uses

- DataPower's XS40 XML Security Gateway [33] 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 [34] enforces WS* and WS-I standards to centralize security and access requirements in policies that can be run as a shared service in front of applications.
- Vordel Security Token Service [35] 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 between web services..
- PingTrust, a standalone WS-Trust Security Token Server [36] creates and validates security tokens that are bound into SOAP messages according to the Web Services Security (WSS) standard.

9.9. 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), and signing (where all private elements should be included in the scope of the signature and where this signature must include a timestamp).
- ***Trust.*** With this solution, we have the choice of implementing the 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 [24] to authenticate sensitive information, and WS-Metadata Exchange [37].
- 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 constraints in the parameters of a security token 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 [17].

9.10. Related Patterns

- The *Trust* Analysis Pattern, [38]. The objective of this pattern is to provide a conceptual model that embodies the abstract aspects of trust to make it applicable to different domains and applications.
- The *Credential* Pattern [39]. This pattern addresses the problem of exchanging data between trust boundaries and how to resolve the problem of authenticating and authorizing a principal's identity over different systems.
- The *Circle of Trust* pattern allows the formation of trust relationships among service providers in order for their subjects to access an integrated and more secure environment [25]. The WS-Trust pattern could be used to establish trust between providers.

10. Related Work

Many patterns have been identified in the web services community, at various level of granularity. For example, [40] and [41] propose patterns for web services composition, while [42] and [43] identify security patterns. Reference [44] describes a pattern for implementing SOAP message processing. However, most of the proposed security patterns are low level patterns. They are effectively implementation patterns that give solutions to concrete problems in terms of specific technologies. Erl has written a whole book on patterns for web services, which includes some security patterns [45]. However, his patterns are rather abstract, e.g., Brokered Authentication, and do not consider any aspects of standards; they are also mostly descriptive. Microsoft has a catalog of web services security patterns; while more detailed than Erl's they also are mostly descriptive and describe low-level aspects, not standards [46]. Sun book of patterns [47] applies mostly to J2EE and their web services patterns are of the same type as Erl and Microsoft. At present, only our patterns describe web service security standards.

11. Conclusions

As indicated earlier, expressing web services security standards as patterns has several benefits for designers and users of web services. We have surveyed here our work in expressing most web services standards as patterns. Our (almost complete) pattern language should be useful for designers of web service-based systems to incorporate security in their designs. They should also be valuable for users

of web services to build secure interactions with existing systems. To make the patterns clearer, we developed first more abstract patterns, from which we derived specific patterns for XML and web services. In this way, designers and users can navigate our pattern diagrams like a map starting from abstract concepts to find more specialized patterns. The Consequences section of the patterns enumerating advantages and disadvantages helps the application of a particular pattern. Our UML models provide a precise description which is still sufficiently abstract; they do not delve into low-level details such as specific WSDL descriptions and protocol aspects. Our example (Section 9) illustrates the level of detail provided by our patterns. The complete collection of all these patterns can be found in [3].

Our future work will produce patterns for other web services standards such as WS-Secure Conversation that depend on WS-Policy as a prerequisite foundation. This will give provide designers with a catalog of most of the important standards needed for web services interaction [40]. The catalog and the pattern diagrams could be part of the tools of a Secure System Development Environment to support a methodology as the one we proposed in [4].

Acknowledgements

The referees provided valuable comments that significantly helped improve this paper.

References

1. Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*; Addison-Wesley: Reading, MA, USA, 1994.
2. Schumacher, M.; Fernandez, E.B.; Hybertson, D.; Buschmann, F.; Sommerlad, P. *Security Patterns: Integrating Security and Systems Engineering*; Wiley: Hoboken, NJ, USA, 2006.
3. Fernandez, E.B. *Security Patterns in Practice: Building Secure Architectures Using Software Patterns*; John Wiley & Sons: Hoboken, NJ, USA, 2012.
4. Fernandez, E.B.; Larrondo-Petrie, M.M.; Sorgente, T.; VanHilst, M. A Methodology to Develop Secure Systems Using Patterns. In *Integrating Security and Software Engineering: Advances and Future Vision*; Mouratidis, H., Giorgini, P., (Eds.); IGI Gloga: Hershey, PA, USA, 2006; pp. 107–126.
5. Delessy, N.; Fernandez, E.B. A Pattern-driven Security Process for SOA Applications. In *Proceedings of the 3rd International Conference on Availability, Reliability, and Security (ARES 2008)*, Barcelona, Spain, 4–7 March 2008; pp. 416–421.
6. Fernandez, E.B.; Delessy, N. Using Patterns to Understand and Compare Web Services Security Products and Standards. In *Proceedings of the International Conference on Web Applications and Services (ICIW'06)*, Guadeloupe, French Caribbean, 23–25 February 2006.
7. Fernandez, E.B.; Hashizume, K.; Buckley, I.; Larrondo-Petrie, M.M.; VanHilst, M. Web Services Security: Standards and Products. In *Web Services Security Development and Architecture: Theoretical and Practical Issues*; IGI Global Group: Hershey, PA, USA, 2009; pp. 152–177.
8. Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Stal, M. *Pattern- Oriented Software Architecture*; Wiley: Hoboken, NJ, USA, 1996.

9. Fernandez, E.; Pan, R. A pattern Language for Security Models. In *Proceedings of the 8th Conference on Pattern Language of Programs (PLoP2001)*, Monticello, IL, USA, 11–15 September 2001.
10. De Capitani di Vimercati, S.; Samarati, P.; Jajodia, S. Policies, Models, and Languages for Access Control. In *Proceedings of the (Databases in Networked Information Systems)DNIS 2005*, Aizu-Wakamatsu, Japan, 28–30 March 2005; pp. 225–237.
11. De Capitani di Vimercati, S.D.; Paraboschi, S.; Samarati, P. Access control: Principles and solutions. *Softw Pract. Exp.* **2002**, *33*, 397–421.
12. Delessy, N.; Fernandez, E.B.; Larrondo-Petrie, M.M.; Wu, J. Patterns for Access Control in Distributed Systems. In *Proceedings of the 14th Pattern Languages of Programs Conference (PLoP2007)*, Monticello, IL, USA, 5–8 September 2007.
13. Priebe, T.; Fernandez, E.B.; Mehlau, J.I.; Pernul, G. A Pattern System for Access Control. In *Research Directions in Data and Applications Security XVIII*; Farkas, C., Samarati, P., Eds.; Springer: Berlin, Germany, 2004.
14. Delessy-Gassant, N.; Fernandez, E.B.; Rajput, S.; Larrondo-Petrie, M.M. Patterns for Application Firewalls. In *Proceedings of the Pattern Languages of Programs Conference (PLoP) 2004*, Monticello, IL, USA, 8–12 September 2004.
15. Delessy, N.; Fernandez, E.B. Patterns for the eXtensible Access Control Markup Language. In *Proceedings of the 12th Pattern Languages of Programs Conference (PLoP2005)*, Monticello, IL, USA, 7–10 September 2005.
16. Ajaj, O.; Fernandez, E.B. A Pattern for the WS-Policy Standard. In *Proceedings of the 8th Latin American Conference on Pattern Languages of Programs (SugarLoafPLoP 2010)*, Salvador, Bahia, Brazil, 23–26 September 2010.
17. OASIS, WS-Trust 1.4. Available online: http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/cd/ws-trust-1.4-spec-cd-01.pdf (accessed on 18 April 2012).
18. Fernandez, E.B.; Delessy, N.A; Larrondo-Petrie, M.M. Patterns for Web Services Security. In *Proceedings of the 21st International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Portland, OR, USA, 22–26 October 2006.
19. Lockhart, H.; Steve Andersen; Bohren, J.; Sverdlov, Y.; Hondo, M.; Maruyama, H.; Nadalin, A.; Nagaratnam, N.; Boubez, T.; Morrison, K.S.; *et al.* Web Services Federation Language (WS-Federation) Version 1.1. Available online: http://download.boulder.ibm.com/ ibmdl/pub/software/ dw/specs/ws-fed/WS-Federation-V1-1B.pdf?S_TACT=105AGX04&S_CMP=LP (accessed on April 18, 2012).
20. OASIS, WS-SecureConversation 1.3. Available online: http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/ws-secureconversation-1.3-os.html (accessed on April 18, 2012).
21. Hashizume, K.; Fernandez, E.B. A Pattern for WS-Security. *First IEEE Int. Workshop on Security Eng.Environments*, Shanghai, China, 17–19 December 2009.
22. Hashizume, K.; Fernandez, E.B. Symmetric Encryption and XML Encryption Patterns. In *Proceedings of the 16th Conference on Pattern Languages of Programs (PLoP 2009)*, Chicago, IL, USA, 28–30 August 2009.

23. Hashizume, K.; Fernandez, E.B.; Huang, S. Digital Signature with Hashing and XML Signature patterns. In *Proceedings of the 14th European Conference on Pattern Languages of Programs (EuroPLoP 2009)*, Bavaria, Germany, 8–12 July 2009.
24. XML Signature Syntax and Processing. Available online: http://www.w3.org/TR/xmldsig-core (accessed on April 18, 2012).
25. Delessy, N.; Fernandez, E.B.; Larrondo-Petrie, M.M. A Pattern Language for Identity Management. In *Proceedings of the 2nd IEEE International Multiconference on Computing in the Global Information Technology (ICCGI 2007)*, Guadeloupe, French Caribbean, 4–9 March 2007.
26. Delessy, N.; Fernandez, E.B. Adapting Web Services Security Standards for Mobile and Wireless Environments. *Lect. Notes Comput. Sci.* **2007**, *4537/2007*, 624–633.
27. W3C, Web Services Policy 1.5—Framework. Available online: http://www.w3.org/TR/ws-policy/ (accessed on 15 December 2011).
28. OASIS, W-S SecurityPolicy 1.2. 2007, Available online: http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.2/ws-securitypolicy.pdf (accessed on April 18, 2012).
29. Buckley, I.; Fernandez, E.B.; Rossi, G.; Sadjadi, M. Web Services Reliability Patterns. In *Proceedings of the 21st International Conference on Software Engineering and Knowledge Engineering (SEKE'2009)*, Boston, MA, USA, 1–3 July 2009.
30. Fernandez, E.B.; Pelaez, J.C.; Larrondo-Petrie, M.M. Attack patterns: A new forensic and design tool. *IFIP Int. Fed. Inf. Process.* **2007**, *242/2007*, 345–357.
31. Muñoz-Arteaga, J.; Fernandez, E.B.; Caudel, H. Misuse Pattern: Spoofing Web Services. In *Proceedings of the Asian Pattern Languages of Programs Conference, 2011*, Toyko, Japan, 5–7 October 2011.
32. Box, D. Available online: http://msdn.microsoft.com/en-us/library/aa479664.aspx (accessed on 15 December 2009).
33. IBM Corporation. WebSphere DataPower XML Security Gateway XS40. Available online: http://www-01.ibm.com/software/integration/datapower/xs40/ (accessed on 25 November 2009).
34. The SecureSpan XML Firewall. Available online: http://www.layer7tech.com/main/products/xml-firewall.html (accessed on 9 December 2009).
35. Vordel STS. Available online: http://www.vordel.com/solutions/security_token_services.html (accessed on 15 December 2009).
36. PingTrust, a standalone Security Token Server. Available online: http://www.pingidentity.com/about-us/news-press.cfm?customel_datapageid_1173=1404 (accessed on 15 December 2009).
37. Web Services Metadata Exchange. Available online: http://www.w3.org/TR/ws-gloss/ (accessed on 15 December 2009).
38. Fayad, M.E.; Hamza, H. "The Trust Analysis Pattern. In *Proceedings of the Fourth Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP 2004)*, Porto Das Dunas, Brazil, 10–13August 2004. Available online: http://sugarloafplop2004.ufc.br/acceptedPapers/ww/WW_1.pdf (accessed on 15 December 2009).
39. Morrison, P.; Fernandez, E.B. The credentials pattern. In *Proceedings of the 2006 conference on Pattern languages of programs (PLoP 2006)*, Portland, OR, USA, 21–23 October 2006. Available online: http://portal.acm.org/citation.cfm?id=1415472.1415483 (accessed on 15 December 2009).

40. Zirpins, C.; Lamersdorf, W.; Baier, T. Flexible Coordination of Service Interaction Patterns. In *Proceedings of the Second International Conference on Service-Oriented Computing (ICSOC'04)*, New York, NY, USA, 15–18 November 2004.
41. Benatallah, B.; Dumas, M.; Fauvet, M.-C.; Rabhi, F.A.; Sheng, Q.Z. Overview of some patterns for architecting and managing composite web services. *ACM SIGecom Exch.* **2002**, *3*, 9–16.
42. Tatsubori, M.; Imamura, T.; Nakamura, Y. Best-Practice Patterns and Tool Support for Configuring Secure Web Services Messaging. In *Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, San Diego, CA, USA, 6–9 June 2004.
43. Imamura, T.; Tatsubori, M. Patterns for Securing Web Services Messaging. In *Proceedings of the OOPSLA Workshop on Web Services and Service Oriented Architecture Best Practice and Patterns*, Anaheim, CA, USA, 26–31 November 2003.
44. Grushka, N.; Jensen, M.; LoIacono, L. A Design Pattern for Event-Based Processing of Security-enriched SOAP Messages. In *Proceedings of the International Conference on Availability, Reliability, and Security (ARES 2010)*, Krakow, Poland, 15–18 February 2010.
45. Erl, T. *SOA Design Patterns*; Prentice Hall: Upper Saddle River, NJ, USA, 2009.
46. Web Service Security Patterns—Community Technical Preview. 2010. Available online: http://msdn.microsoft.com/en-us/library/ff648183.aspx (accessed on 12 April 2012).
47. Steel, C.; Nagappan, R.; Lai, R. *Core Security Patterns: Best Strategies for J2EE, Web Services, and Identity Management*; Prentice Hall: Upper Saddle River, NJ, USA, 2005.