SECURING XML WEB SERVICES – A PLUG & PLAY APPROACH

Cristian Donciulescu, Luminita Vasiu
Westminster University, London, UK

Keywords: XML Web Services, security, Web Services architecture, plug and play architecture

Abstract: XML Web Services have the potential of becoming the underlying technology on which the future of the Internet is built. However, several hurdles stand in the way before that can happen. Security issues, for one, are slowing down the rate at which industry adopts the use of Web Services and in spite of the coordinated efforts made to correct those problems, the actual results can be years away. This paper is aimed at providing a more immediate solution for Web Services in the form of an architectural design, not only regarding security aspects, but rather enabling a wider range of e-commerce supporting features which are of utmost importance for a wide acceptance of this technology within the industry.

1 CONTEXT

The XML Web services technology has been around for some time now, however, its potential is still largely unused due to certain problems that were not addressed by their original specification and by several additions that have appeared over time. The biggest such problem was and still is to this day the lack of security mechanisms available for developers. Because of this, the vast majority of web services available now are simple pieces of code, without the need for security. The few XML Web Services used commercially in industry have custom security implementations that usually take more time to build than the actual service functionality (Trivendi, 2002; Rosenberg, 2004).

While attempts to solve this stringent problem have been made for some time now by big players on the software market, such as Microsoft and IBM, the most notable result was the WS-Security specification. This specification is only the basis for more than 30 other specifications developed or under development under the umbrella of Web Services Interoperability Organisation (WS-I). Every such standard is designed to cover some particular aspect of security for XML Web Services.

With the entire standards development effort going on, it is small wonder that developers are still confused and unable to unleash the full potential of the technology and with much more than one standard coming up, combining everything into a comprehensive security solution might be very difficult (Bergholz, 2004).

Of course, the issue of time is always present. The delay in adopting the technology on a wide scale can only erode its credibility in the IT community. This delay might cause further delays resulted from the lack of trust in the security that would be offered as a specification add-on rather than as part of a comprehensive initial set of specifications. Technology offers a recent example on how an initial impression made by the lack of a sufficient security mechanism carries on even after the problem was addressed. The WiFi technology is still avoided by many because of that delay in implementing strong security in the first place.

This paper presents a solution that could be used to speed up the wide-scale adoption of the XML web services technology by providing a straight-forward way of securing services, as well as opening the way for further enabling the technology for e-commerce use.

2 MAIN ISSUES WITH WEB SERVICES

As discussed above, providing developers some straight-forward way of securing services is the main concern at the moment. However, the technology

Donciulescu C. and Vasiu L. (2005).
SECURING XML WEB SERVICES – A PLUG & PLAY APPROACH.
In *Proceedings of the Second International Conference on e-Business and Telecommunication Networks*, pages 25-32
DOI: 10.5220/0001419500250032
Copyright © SciTePress

has more than one hurdle before it can really become a fundamental building block of the Internet, as many IT specialists envisioned it when its specification was first published.

The main problems that we have identified with Web Services are:

- Privacy – web services communicate via SOAP XML documents that are simple text streams sent over the network.
- Routing/messaging – specifications have been drafted and are in the final stages of being approved.
- Transaction handling – there is no implemented way as of yet to perform any sort of transactional operations for services, which is an essential for any business operation. Specifications are being developed now (Trivendi, 2002)
- There is no way to do any performance/tuning/audit on web services. Developers cannot know which service offers a better performance and reliability. There is no planned way of doing this in the near future, although there is some related research in the field (Bravetti, 2004)
- Dynamic discovery – there is no way of automatically discovering which service performs a specific type of task, nor there is a way of switching between implementations of similar web services (Bergholz, 2004). Further, there is no way of finding a web service based its functionality rather than its description
- Complexity of the upcoming security model - the specifications that are now being drafted under the patronage of WS-I address different security issues and are built to be used as modular components added to service messages. These specifications are supposed to work and integrate very well with each-other. For example, a Web Service that needs to have its users authenticated would use WS-Security along with higher level standards like WS-Authorization and maybe WS-Trust. Making all these standards interact will prove in our opinion to be a difficult task. Added complexity means higher chance of failure in any type of system, and as it stands now, the complexity of the security policies that will be applied is very high (Rosenberg, 2004)

Discovering web services is essential, after all a service that cannot be found by clients is useless. The problem is that the mechanism available for service discovery is very basic and does not allow clients to perform any sort dynamic discovery. A service is basically embedded in an application at development time and there is no default way of changing it after the application is shipped to clients. There is no way to even switch automatically to another address if the service moves. While this issue might not seem security related at all, it proves to be extremely important regarding service availability and reliability. We will also show that while everybody agrees that dynamic discovery is necessary, the industry hasn't even agreed to what we should understand by dynamic discovery (Bergholz, 2004; Vinosky, 2004). By dynamic discovery we refer here at the "ability to describe systems in which clients search through registries to first discover and then invoke services supporting the capabilities they require" (Vinosky, 2004).

The architecture that will be presented in this article proposes some solutions to some of these problems.

3 THE PROPOSED SOLUTION

The approach proposed here relies on the fact that all communication involving web services takes place through the SOAP protocol; therefore the SOAP message would necessarily be the main engine that would carry any payload for a potential attack. Moreover, controlling SOAP message flow, by identifying the message source and destination as well as monitoring the specific Web Service to where each message is addressed can prove to be an effective way of managing certain services that run in a controlled environment.

The basic architectural element we rely on controlling the flow of SOAP messages is the security context. A security context is meant to group together a set of security policies that are applied to all the services that will be managed inside that context. Each context is administered by an administrative entity, be it one or more individuals. A context is set to act as a buffer between the service provider (Web Service) and requestor (Web Service Client).

A context is comprised of the following basic elements, based on the initial design resulted after the research conducted. They are presented in the relative order in which an incoming message would reach them:

- A bidirectional envelope that intercepts the incoming and outgoing SOAP requests and responses as they arrive to and from the Web Services that are managed. The interception would take place much in the same way a firewall intercepts packages. There are firewalls that can already recognize and block SOAP messages. If needed, the SOAP message would be decrypted at this level using probably a private context key. Encryption/decryption would take place here if required. While SSL could be used here to secure the communication channel, it is not always feasible, especially since both endpoints must have a digital certificate. Due to the relatively high cost of the digital certificates it is highly unlikely that the clients will employ them on a large scale (Cremonini, 2003).

- An access control list which retains permissions for groups/users (CACL). The administrative entity of the context has the exclusive right to modify CACL.
- An authenticator, sitting behind the envelope. The authenticator receives the messages intercepted by the envelope and interprets the information about the message source. Message authentication takes place here. The authenticator has access to the security context's access control list.
- The message is passed further down to an audit/logging unit that retains the information about the requestor, time of request, etc.
- A load balancing unit that forwards the message to the actual web service that is the intended destination. There are two ways this mechanism can be implemented in our opinion (using a centralized or decentralized approach) and this issue will be addressed later.
- A database for storing CACL and logging/auditing information
- Finally, a context manages zero or more web services belonging into two categories. These categories will be described when discussing about context inheritance, for now we will mention that they are either the managed services or proxy ones.

A security context can trust another context and only one. Multiple trust relationships are possible, however it has been decided that the trust relationship complexity would be too high, at least for a first version of the architecture.

A trust relationship between two contexts allows the "child" to trust a "parent". A child-parent trust relationship implies that the security policies given by the parent security context are applicable to the child. That is, the child inherits an ACL from the parent. The child ACL takes precedence over the parent ACL.

The implementation of such a trust relationship implies generating proxy web services. A proxy web service is defined as being a Web Service whose function is to forward all incoming requests to another service that has a similar signature (WSDL contract) as the proxy. It is theoretically possible to generate proxy services based on the contracts of real web services because generating proxy objects from WSDL contracts is basic functionality in all the Web Service supporting infrastructures. Such a proxy would be generated by the child context and published on the parent context in a trust relationship. The publishing process would require human supervision at least in the current design

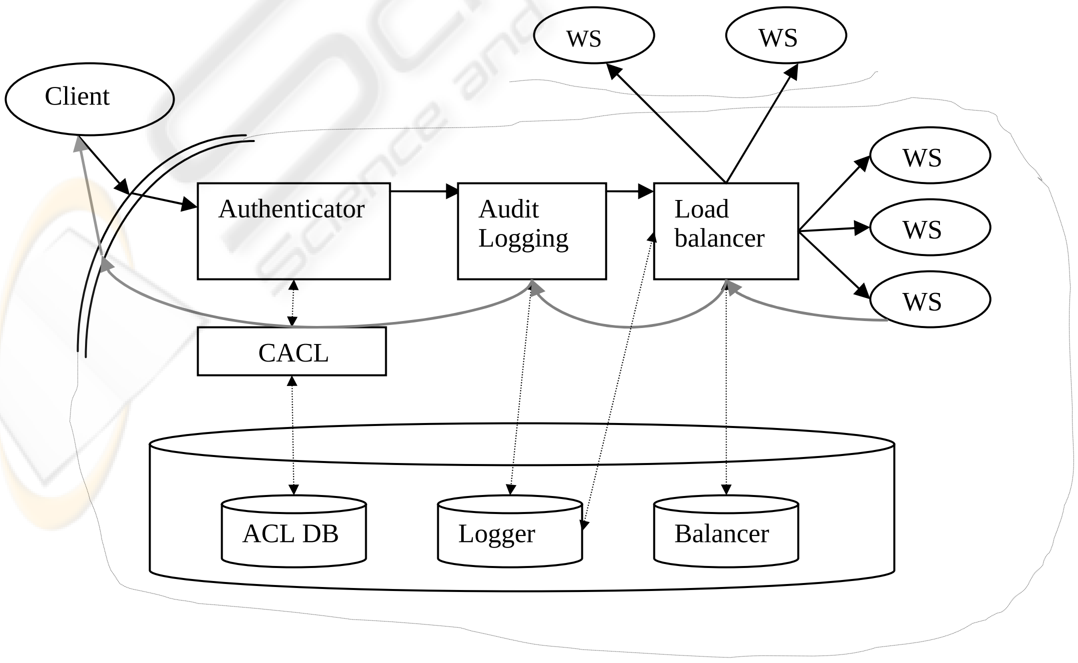

Figure 1: Security context architecture with centralized load balancer

phase, in order to minimize the security risks of such operation. The proxy services will become clients for the original services, but they will have the right to make calls to the original services because the child context in which those reside trusts the context in which the proxies reside. There are plans to automate the process completely. The publishing process would require of course some security privileges to be given to the child context by the administrative entity of the parent.

This implication generates a special type of web service that resides in a parent context: a proxy service, as described above. This means that a context can have two types of services: proxies and regular services. A proxy service can be made discoverable by clients or not, depending on the security policies implemented in the security context it resides in.

As previously mentioned, there are two ways of implementing a load balancing mechanism. The two options available are depicted in Figures 1 and 2.

The first method of providing a load balancing mechanism is depicted in Figure 1. The client sends a SOAP request to the service which was discovered before by any means. The message is intercepted by the envelope and if necessary decrypted using a context private key. The decrypted message is passed through to the authenticator unit which checks the message sender against a context access control list (CACL). Provided that the CACL permits the originator to interrogate the given Web Service, the authenticator passes the message to an audit/logging unit that will store whatever information is necessary about the request, the web service it was addressed to and so forth. The authentication can be based either on the content of the SOAP message or on the information that can be gathered about its origin, although a combination of the two would provide the most accurate results. Research is taking place for identifying the best way of achieving a comprehensive authentication model.

The message is then passed into a central load balancing unit that decides which service will actually perform the business logic associated with the request. This decision can be based on a multitude of factors, such as:

- The number of identical web services available to service a request at a certain moment. Using WS-Addressing it is (theoretically) possible for a client to make a general request to a service that has many instances running without having the knowledge of this. However, the solution is only provisional since the standard is not final yet. At the time when this paper is written (October 2005), the WS-Addressing standard has been submitted for review (W3C, 2004).
- The information available from the audit unit, detailing the usage history and present load on a specific service instance.
- The information available in the balancer database which typically would describe an addressing table for each service inside the context.

This load balancing solution can be very easily expanded to a pseudo-dynamic discovery solution. Using an XML equivalence language, either proprietary (at first) and then a standardized one, the balancer database can not only hold information about identical web service instances running in different points on the network, but also information about similar services that perform the same functions but have other contracts. This situation could be very frequent, the classic example of a credit card validation service being eloquent. Using a semantic language, such as OWL-S this mechanism could be expanded into a fully fledged dynamic discovery system, where services are invoked based on their functionality rather then their description (daml.org, 2004).

Once the message is relayed to the Web Service that is intended to service it, the service will execute the business logic required and will return the SOAP response to the load balancer. The load balancer is the only client the service will actually answer to. While this might seem bad practice from a performance stand-point, it is essential that the response is returned by the service to the party that requested it in the first place, for maintaining compatibility with the basic Web Services specifications.

The load balancer will forward the message to the logging/audit unit that will again record information such as response time and service instance that performed the business logic if necessary. This in turn will be used again in routing further requests. The information will also be useful for developers in deciding which service to use, if it will be made public in UDDI registries.

The audit unit will pass the response directly to the client, but the envelope will again intercept that request and perform an encryption on it if dictated by the security policies enforced by the context. Finally, the response is forwarded to the client.

Evidently, the whole process described above is transparent for the client, an essential requirement for maintaining conformance with the Web Services specification.

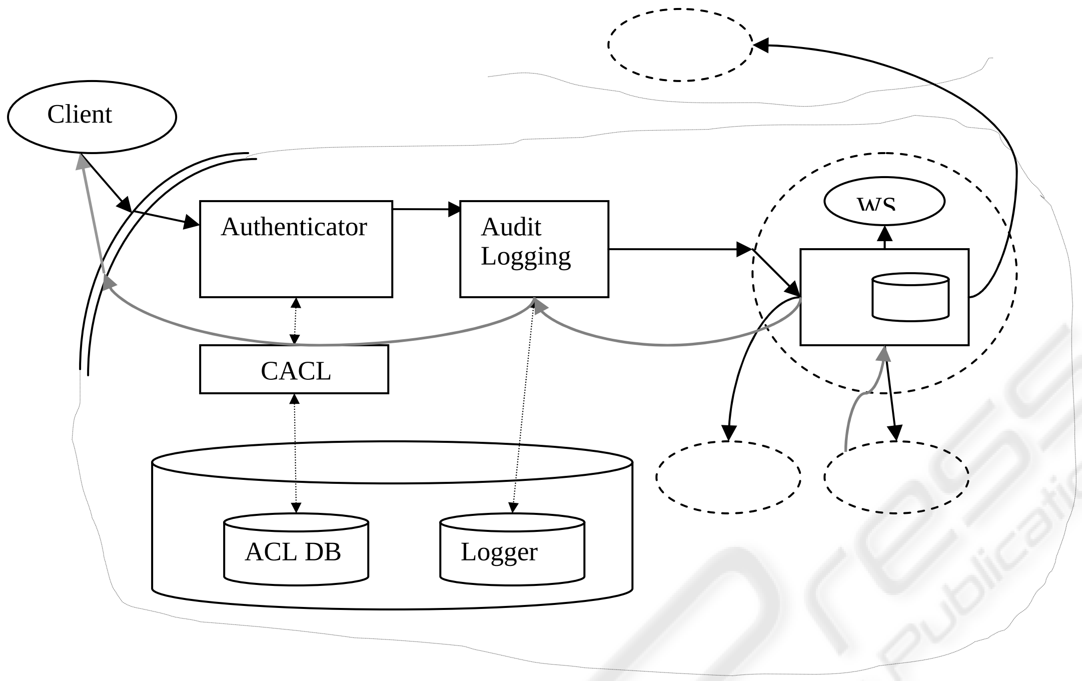

Figure 2: Security context architecture with decentralized load balancing

The second approach for implementing a security context architecture with load balancing is slightly different and presented above in Figure 2. In this approach, each managed service has its own load balancer. Each service has a repository attached which stores information about the alternative services that can perform the same business processes that it does. The service balancer would check if the service is available to process a request and forward it to the service or to an alternative of that service.

The message path through the security context would be relatively similar to the previous discussed approach; however, the audit/logging unit would pass the request directly to the web service which was chosen by the client as destination. The message would then be intercepted again by a secondary envelope, which would direct it to the service load balancer. If the service is available, the business logic would be executed and the response passed back to the balancer and would follow the path back to the client similar to the centralized approach. If the service is unavailable, the request would be passed to an alternate service (or more than one for that matter, although that approach would not work in all cases so we did not take it into account on the grounds that it cannot be generalized). The process would repeat itself until a service would eventually perform the request and return a response.

The advantage of this approach is that the load balancer becomes decentralized. This means that given a decent network of services, the request would travel through the network without problems until it would be serviced, without the need for a huge list of alternates for each service. Also, if a service does not have any alternates specified, in other words if it is working like a standalone service, the balancer does not interfere at all in the whole process, being completely bypassed and reducing network stress and processing time. Also, each balancer can decide whether its own service is available or not to service a specific request, this functionality being easier to fine-tune on this approach.

The downside of the security context being architected this way is that it is more difficult to manage, although it would provide finer control detail. Also, the balancer would most probably not have all the information that a centralized approach would be able to use. For example, the auditing and logging information would be more difficult to access and gather, mostly time-wise. Another possible downside is that due to this distributed nature, it would be difficult and unfeasible to use an actual database for storing the balancer related information, thus losing the advantages offered by a modern DBMS. Finally, we believe that it would be more beneficial to have all the information relating to a context into the same data source.

Taking into consideration the advantages and disadvantages presented, it is our opinion that the first approach is more feasible for implementing, although the decentralized version would be more versatile. However, further testing will still be carried out to determine the feasibility of the decentralized model, but, at this time we are leaning towards using the centralized one.

The following diagram (Figure 3) presents a top-level view of the architecture, above the security context level. The diagram presents a theoretical and reduced version of the architecture, with the purpose of demonstrating the trust relationships between security contexts. Depicted are three contexts, marked as WSMC(A) 1, 2 and 3. Each context has an access control list associated with it.

WSMC2 and WSMC3 trust WSMC1, which implies that they inherit ACL1.

Each context has a number of managed services, depicted with a continuous line and context 1 has some proxy services (dotted lines) which it manages also, since it is trusted by WSMC 2 and WSMC3.

Of course, the trust-inheritance tree could be and is desirable to be much larger than this. An optimal size is difficult to predict, although some testing is planned on this issue, in later stages of the project. The following section will describe various scenarios of security situations in which clients invoke services residing in the three contexts and the responses received.

4 CONCLUSION

The architecture described was designed with the purpose of providing security features for XML Web Services, features that are not available at present and that will most probably be missing or hard to implement in the foreseeable future as well. The design has been carefully steered towards being compatible with the XML Web Services basic standards and future developments. We do believe that it is compatible with upcoming specifications and, of course, with the WS-I Basic profile.

This is achieved by not interfering with the inner workings of the managed web services or the SOAP request/response messages' content. The main advantages that we see available to the technology once the architecture would be used on a decent scale are outlined below:

- WSMA basically provides a plug & play security mechanism for web services. As the architecture is designed, a service with no security features whatsoever can be placed into a context (or more than one) and it would "inherit" all the security policies defined by the administrator of that context
- The plug & play functionality would allow developers to focus on the business logic of most services, rather than writing complicated, security-related code which is error-prone and often leaves security gaps behind and takes up

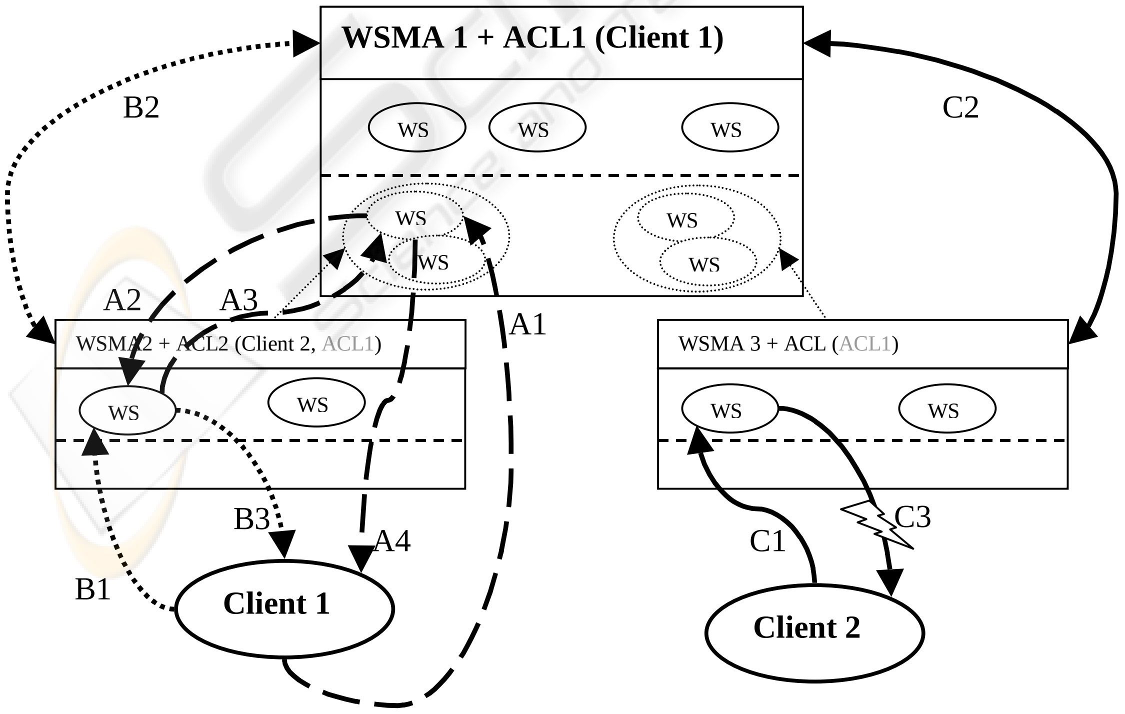

Figure 3: Top level view of the WSMA architecture

valuable time

- WSMA separates completely the business logic implementation of the service from the security related one
- WSMA can be used with any Web Service, both legacy and new ones being accommodated
- Because of the context-oriented architecture, an entire business process could be deployed inside a security context, providing clients access to the entire range of services that comprise that process. This allows for the possibility that in a business process which is composed of many services, the client must only be authenticated once, rather than with every service at a time, decreasing the overall response time for that business process
- The authenticator can be customized to allow multiple security policies, depending on the specific needs addressed by the security context
- The WSMA allows specifying method level access rights for the web services managed by a context. This functionality is not possible at present, nor will it be made available by any of the upcoming specifications. However, it is usable and desirable, considering that Web Services can expose multiple methods at a time, with different security policies needed on them. For example, a service that interrogates and updates a customer database would want to expose the interrogation methods to a set of its clients and the update methods to a subset of those clients. At present, the only way of implementing this kind of functionality is providing two separate services
- The architecture is designed in such a way that it allows for on-the-fly deployment (creation or update) of web services into a security context. This means that the services can be updated without the context being unavailable to the clients at any time
- The administrative entity of the context can modify the security policies at any time without any downtime for the context or the services managed
- The policies for all the services managed can be modified all at once, thus allowing for the access rights on an entire business process to be modifiable at once
- Each security context has complete control over the service managed, thus making the administration of those services an easy task
- WSMA allows a chain of trust to be built, which, in turn, allows a client to have access to XML Web Services for which specific security settings have not been specified. In other words, a client is not authenticated by the service, but by the context
- The architecture provides a mechanism for performing various audits and logging on the managed Web Services, which gives developers a benchmarking tool for selecting the proper web service for their application
- A load balancing mechanism is provided which helps insure that one service will always be available to service requests, even at the busiest times
- A pseudo-dynamic mechanism for discovery is provided, which allows a request to be serviced by another service than the one originally intended, provided that the business logic performed by the alternate service is similar to the business logic performed by the original one
- A truly semantic discovery model for web services can be envisaged subject to integrating a semantic language such as OWL-S into the Web Services architecture
- The architecture is open to implementing any of the specifications related to Web Services as soon as they would become a WS-I requirement, thus ensuring interoperability with services and applications that are not using WSMA

5 CURRENT AND FURTHER WORK

While the big picture on the WSM architecture design is relatively finalized, there still are some problems that need to be addressed, or questions that need to be answered before starting any implementation. The issues being worked on now are listed below:

- It is unclear how the WSMA would behave when the trust inheritance tree becomes big. The response time to a request is quite important and the load balancing mechanism would slow things down a little. If the request must follow a long network path until it is answered, the delay might become significant. It is important then to determine the actual optimal size of the inheritance tree, or else the number of forwards a SOAP message would be allowed to pass through before the client gets an exception
- Also, in a big trust inheritance tree, every time a service is updated in a child context, the publishing of the proxy service triggers an update that must take place in every parent context above the child. It is necessary to test the required resources this update would need in different context sizes. If this update is too expensive, resource-wise, a solution should be found to either reduce its cost, frequency or the

update should be limited to only a number of parent contexts.

- There must be a way for the client to receive an error if the request cannot be serviced for some reason. This response, if sent, should of course be an XML SOAP document. It is unclear whether the client should actually receive the response or not. There are pros and cons to either approach. If the client receives a response every time the request is unavailable, even for lack of proper credentials, there is a security risk of favoring denial of service attacks (see **Erro! A origem da referência não foi encontrada.**). Not returning a message implies that a client must always wait for a timeout error, on a timeout interval specified locally. There is of course a middle way of sending messages back only in certain conditions, but the approach that will be taken is not clear
- The model of the inheritance system for the trust tree has not been decided yet. We are oscillating between a simple approach, in which a service can be visible or not and a more complex one in which each service would have a visibility flag attached which could specify information about the number of levels the service should be visible, and related issues. For the first implementation we are leaning towards the simple approach
- In the future, we are looking at implementing support for transactions into the WSM architecture, which would make WSMA an truly e-commerce ready deployment platform for web services

An evaluation of existing implementations of open source firewalls will be carried out in order to assess whether one of those could be used for the building of the security context envelope, since its functionality resembles that of a firewall quite a lot. Hope is to find a solution that would be easily adaptable or convertible for our purposes. However, if such a solution will not be found, a custom implementation can be made (Cremonini, 2003).

REFERENCES

Trivendi R., 2002. Professional Web Services Security. Wrox Press Inc., ASIN: 1861007655

Rosenberg J, 2004. Securing Web Services with WS-Security: Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption. Pearson Higher Education. ISBN: 0672326515

Bergholz, A., Chidlovskii, B., 2004. Learning query languages of Web Interfaces, *Proceedings of the 2004 ACM symposium on Applied computing*

Bravetti, M., Lucchi, R., 2004. Web Services for E-Commerce: guaranteeing security access and quality of service, *Proceedings of the 2004 ACM symposium on applied computing*

Vinosky S., IONA technologies, 2004. White paper – Web Services and Dynamic Discovery - *http://www.iona.com/devcenter/articles/stevev/1101sv.htm*

Cremonini M., 2003. Security for Web Services: An XML based approach to combine firewalls and web services security specifications. *Proceedings of the 2003 ACM workshop on XML security*

MSDN Library, 2004, *http://msdn.microsoft.com/webservices/understanding/specs/default.aspx*

W3C - Web Services Addressing (WS-Addressing), 2004 -http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810

DAML Services, 2004 - http://www.daml.org/services/owl-s