AQTR 2004 (THETA 14)
2004 IEEE-TTTC - International Conference on Automation, Quality and Testing, Robotics
May 13 – 15, 2004, Cluj-Napoca, Romania

A SOLUTION FOR PROVIDING PSEUDO-DYNAMIC DISCOVERY FOR XML WEB SERVICES

Cristian Donciulescu, Luminita Vasiu

WITRC, School of Computing Science, Middlesex University, London, UK
Email: c.donciulescu@mdx.ac.uk, l.vasiu@mdx.ac.uk

Keywords: XML Web Services, security architecture

Abstract: Distributed computing reaches a new stage with the concept of XML Web Services. It is a stage that enables a high level integration between applications and business logic components, rather than objects, providing the opportunity to allow for further device integration capabilities, reaching closer to the final goal of achieving a seamless interaction between any two devices. However, for the time being, there are several hindrances in the way of web services. This article describes a solution to one of these problems – dynamic discovery.

1. BACKGROUND CONCEPTS AND INDUSTRY EFFORTS

XML Web services have been around for some time now, however, their potential is still largely unused due to certain problems that remain unsolved in their specification. The biggest such problem was and still is to this day the lack of security mechanisms available for developers. Most web services available now are simple pieces of code, without the need for security. The few XML Web Services used in industry have custom security implementations that usually take more time to build than the actual service functionality [4, 6].

While attempts to solve the 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 one of more than 30 specifications developed or under development under the umbrella of Web Services Interoperability Organisation (WSI). Every 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 more than one standard coming up, combining everything into a comprehensive security solution might be very difficult [3].

One issue that remains without an answer even if all the standards will be completed and used is what we call dynamic discovery. At present, developers needing to use a XML Web Service must search a UDDI registry for human understandable terms that will point them to some specific service they need. This process resembles very much with a regular search in an Internet search engine. While a developer will probably find what is needed eventually, this idea of static discovery is fundamentally flawed. There are a few issues that eventually lead to security gaps that make the software using a web service very unreliable.

First, the XML Web Service address (or rather the address of its contract) is usually stored within the code, implying that if the XML Web Service changes addresses, which will eventually happen at some point, the code must be recompiled. While for web applications that have thin clients (browsers) this is not a tragedy, for other applications this would lead to the necessity of a forced update. Coupled with the strong possibility that the developers of the software might not even be aware of the change of addresses, the result would most certainly be that the application will not run anymore. This could be addressed by using configuration files that users can modify in order to maintain the locations of the XML Web Services the application uses accurate. However, one of the advantages of the Web Services technology is that these services can remain hidden, so that users are not aware which services are used or even that they are used and this approach would not allow that to happen. Also, if a chain of Web Services is used (which can happen without the knowledge of the developers, since they can just as well use only the last link in the chain), a change of address in a previous link can lead to the application not working without any apparent reason.

Second, if the service becomes unavailable for any reason (network downtime, hacker attack, etc), there is no way for the application to switch to another service that performs the same tasks automatically, unless a rather complex and custom implementation approach is taken. That approach is not feasible; it would probably outweigh the development effort for the business logic of the service which would defeat the purpose of using Web Services in the first place.

Third, how can a developer know which service to actually choose? Reading a description of a web service is much like listening to a commercial. You know what the vendors of that web service want you to know. [8]

A mechanism that would allow for developers to specify in one way or another which *kind* of service they need and that would choose from a multitude of Web Services the best Web Service to be used at the moment of the request would be enormously beneficial. Not only that it would solve the problems outlined above, but it would also simplify other security tasks as well, such as load balancing incoming requests between similar web services. Specifying which kind of service is required in such a way that human and machine could understand the concept is not easy, but there are a few approaches that could be investigated.

This article tries to provide a solution for some of the issues above by adding to a Web Services Management Architecture that was designed for offering plug&play security for Web Services.

Under these circumstances detailed above, it is our belief that this architecture that would allow developers to build and deploy web services and that would take all the security issues outside the actual development process would be a feasible solution to the problem. The architectural requirements have already been drawn out [1] and various tests are being developed at the moment to determine its feasibility.

2. THE WEB SERVICES MANAGEMENT ARCHITECTURE (WSMA)

The WSMA is an architecture for securing XML Web Services without the service developer being involved in writing any security related code embedded in the Web Service itself (Donciulescu, Vasiu, 2004). The WSMA architecture is designed to achieve the basic tasks outlined below, however, as shown in this article, it can be easily extended for solving many security related issues.

- The WSMA architecture is designed to perform the following actions [2]:
- To control authorization of incoming requests to web services it manages;
- To maintain access rights for clients accessing services;
- To verify the identity of a client;
- To allow the deployment of a XML Web Service without any downtime;
- To manage situations where a Web service may become unavailable;
- To manage data integrity tasks such as encrypting and signing outgoing messages;
- To maintain information about the functionality of managed Web services;
- To be able to CRUD security environments which manage Web Services;
- To be able to inherit permissions and settings from other trusted environments and have the possibility to trust other environments

The WSMA architecture will create a sort of a "bubble" around the XML Web Services managed. It will be able to isolate them from the outside world and act as a kind of intermediary between the clients and the managed services. This could be accomplished by a mechanism similar to a firewall that intercepts all incoming SOAP requests. When receiving a message, the WSMA will perform the following steps:

- Decrypt the message using a private key (assuming the client had the message encrypted);
- Verify the identity of the client;
- Identify the service for which the client made the request. If the service resides within the current context, verify if the client has the rights to access that service (or even the particular method within the service).If the service is not found within the current context the architecture should pass the SOAP request to all the other trusted contexts;
- Allow the service to run or wait for the other contexts to return a response and grab the result (the response SOAP message);
- Encrypt the response for the client if necessary;
- Send the response back;

The above sequence of events is based on the following assumptions:

- There is a possibility to administer the individual local security context completely independent from the others;
- The local context can inherit all or some security settings from another context and from only one other context;
- The local context can trust any number of other contexts and can lend some or all its settings to those contexts;
- There is a mechanism that allows a client to address a request for a service to a parent context that will redirect it to its destination. Such a mechanism is possible to be established by providing each context with the possibility to publish the WSDL contracts of the XML Web Services managed on its parent.
- The architecture described above is in fact a tree of contexts that trust their parents and can be managed locally by independent administrative entities. Each context can manage any number of XML Web Services. A context has total control over all the services it manages. The administrative entity of a context it is able to manage at least the following aspects for the services it controls:
- Two configurable levels of authentication for clients: at the XML Web Service's level and at the method level for each particular XML Web Service;
- Whether or not the SOAP requests should be accepted unencrypted for each particular managed XML Web Service;
- Whether the SOAP responses should be sent encrypted all the time, only on request, only to specific clients or never
- The possibility to build contingency plans for the situation when a XML Web Service becomes unavailable.

These plans could include a standard SOAP response that the client can handle within the parameters expected for each request or the possibility to specify alternative hosts for the same XML Web Service or even alternative services performing the same task. This could be taken forward in the future by creating a possibility to specify equivalence between different XML Web Services that perform similar tasks. For now, specifying alternative services could be accomplished by building interfaces between them. An interface language based on XML of course could be beneficial at this point. The context should allow adding, removing or modifying information about the managed XML Web Services on-the-fly, without the need to take the context down at any time. This approach is discussed further in this article.

3. WSMA AND DYNAMIC DISCOVERY

The process of discovering a web service is at the moment restricted to a human search into a UDDI directory. The search is based on the description of that service and is done similar to a regular search engine search, with the disadvantages previously described. A dynamic discovery specification was just released (Feb. 2004) by Microsoft in collaboration with BEA Systems and several other players on the Web Services market, it is still just a specification. Further work will be required to assess how to use this specification with WSMA.

The WSMA is already designed to manage security aspects of web services and can be extended to perform description and discovery services. Each management context would act like a small UDDI directory for the web services it manages. The context tree would then become a UDDI directory in itself and could be linked to actual UDDI registries for easy reference. Such a service would complement the structure of the WSMA architecture perfectly; however this discovery system will still be a static one, but would allow services

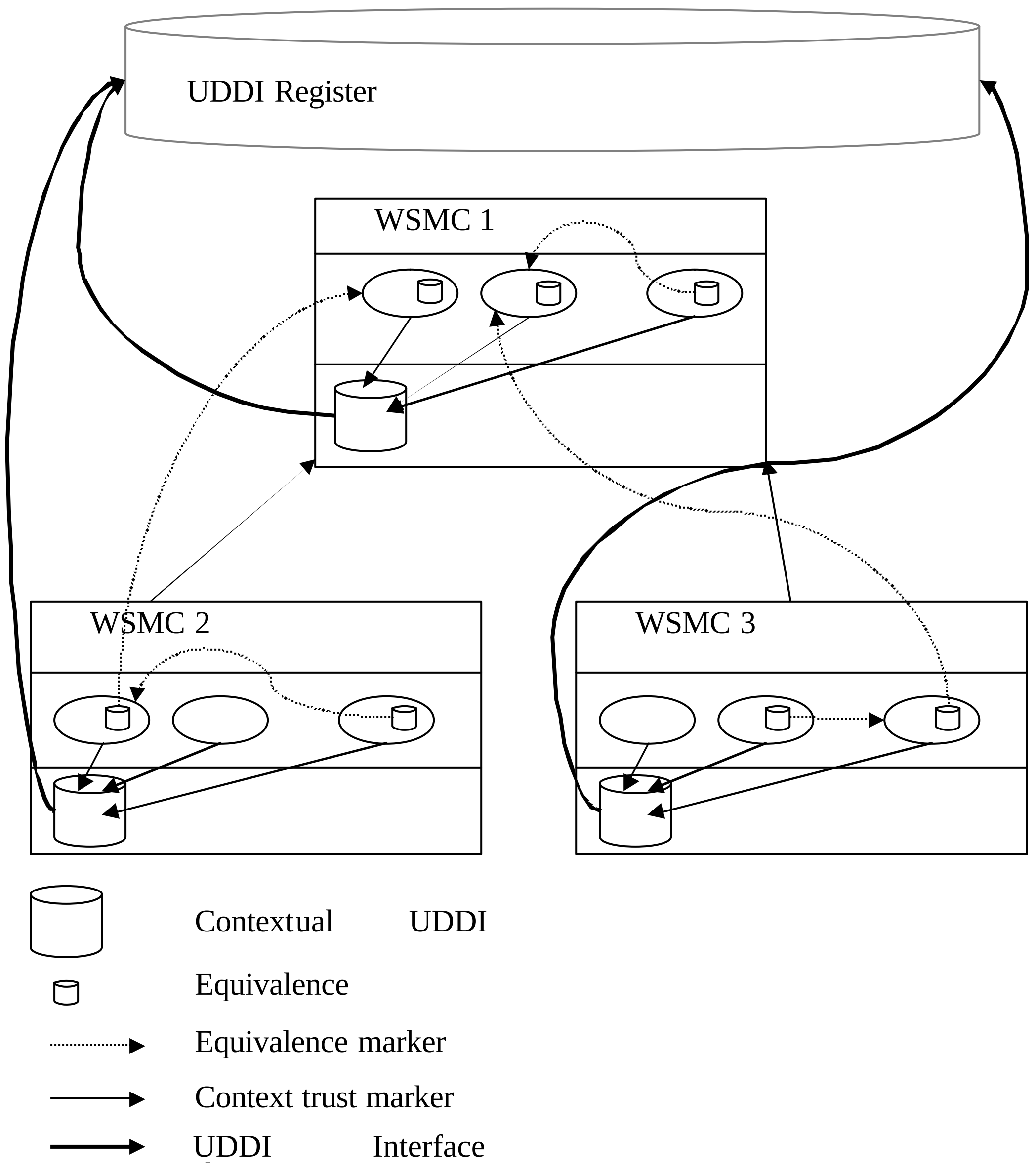

Figure 1 - The WSMA Architecture with the pseudo-dynamic discovery extension

to be monitored by their contexts. If a fault would be found, the context could report this to service users so they would take appropriate action.

In order to achieve dynamic discovery, further efforts have to be made to create a comprehensive method of defining web service equivalence, perhaps even at method level. An initial solution, until an XML-based equivalence service would become standard, would be to set up an equivalence system within a WSMA tree. That would allow each service to specify alternatives for each or some or all of its methods. Also, contexts could be able to specify alternatives for entire business processes. The WSMA architecture with the dynamic discovery component would look like below:

Figure 1 shows three XML Web Services Security Contexts (WSMC). WSMC2 and WSMC3 trust WSMA1. Each context has a small UDDI repository attached to it. It stores a list of web services published in that context. Services that are managed by the context can reside in the contextual repository, but this is not compulsory in any way. Each contextual repository (CR) must have a link to a normal UDDI register, which acts like a buffer between developers and applications on one side and the WSMA on the other side. This configuration is enough to simulate a UDDI structure on the WSMA. However, for achieving a pseudo-dynamic discovery, each web service managed can have an equivalence repository (ER) attached to it. This is another type of UDDI register that can store two types of equivalence:

- Service equivalence
- Method equivalence

The service equivalence represents a list of exactly the same Web Service, running on different environments. This is useful for performing load-balancing for example, or for dynamically switching to backup services in case the main one fails for any reason. While not all Web Services can rely on this mechanism, there are quite a few situations that might benefit from it.

The method equivalence represents a list of alternative methods from different types of Web Services. An alternative method must have similar inputs and outputs from the business logic point of view, although the signatures of the functions need not be identical. The following example illustrates two alternative methods from different Web Services:

```
[WebMethod] //belonging to Bank A
public int CrefitCardValidation(string ccNumber, string ccName, string endMonth,
string endYear, string ccType){..}

[WebMethod] //belonging to Bank B
public bool verifyCC(string name, string number, string endMth, string endYr,
string type){..}
```

The two methods above perform the same business process, however their signatures are slightly different, as it would be natural when the two web services have been developed by different programming teams. The equivalence repository of service A would be storing a the equivalence information between its CreditCardValidation method and the verifyCC method of service B. Also, the service B ER could, but this is not implied, store the similar information about service A. A mapping of the parameters and types must also be specified, through simple XML documents that in our opinion would not cause implementation issues. Whenever the call to the service A would fail for any reason, the WSMA would automatically reroute the call to service B. Evidently, a trust relationship, managed by the WSMA should be established between the two services if they reside in separate contexts.

This model can be expanded to encompass situations where a more complex business process could have an equivalence with another. However, even in the situation above, several Web Services referring each-other as shown would probably work much more efficient and error-free than on their own. Using the same equivalence, WSMA could perform automatic load-balancing tasks, preventing DDNS attacks, while maintaining the security of the whole system.

Taking the idea a little further, each context can monitor the services managed and based on this monitoring can decide which service should perform a task at a certain moment in time. This would assure developers and application users that at any given moment the most efficient (and secure) service is used.

4. DYNAMIC DISCOVERY ADVANTAGES

An architecture as described above, which manages the description and discovery of the managed services would solve (apart from the security problems which it was originally designed for) some of the issues discussed at the beginning of this article:

Even if the address of the web service would still be kept as is, either in the code or a configuration file, in the event the service called becomes unavailable for any reason, the call would be automatically redirected by the WSMA to another similar service specified in the ER of the original service. The call could travel through a chain of services until it would be answered and the response would get back to the client without the client knowing that the response is not coming from the service that was actually called. This in turn will eliminate the problem of the application failing simply because the service used is down.

There would be an embedded load-balancing mechanism within the architecture that would lessen the possibility of a successful denial-of-service attack on a critical web service. In fact, this approach would actually allow for decentralizing a Web Service, which is a great benefit for any software product. A decentralised architecture is much more difficult to attack, so allowing for this type of discovery would address some security issues as well as being convenient for both developers and users.

The WSMA architecture can monitor the performance of the managed Web Services and make decisions to which service to use at certain moments. Furthermore, based on this monitoring developers will have a benchmarking tool for assessing which service offers the best performance over time.

This approach maintains full compatibility with existing UDDI registries

In time, a true dynamic equivalence could be developed, so that the decision on the alternatives for existing services could be taken automatically by the WSMA

5. FURTHER WORK

The architecture described tries to address several security aspects of Web Services. Designed originally for performing authorisation, authentication and other common security tasks, this article shows how the architecture can be extended to address more issues, like dynamic discovery and load balancing.

While the architecture is only in the design and initial testing phase and there are still many questions to be answered before it can be implemented, we believe there are no insurmountable obstacles to be overcome. Further testing is required for evaluating the best methods for implementing various parts of the WSMA.

The approach shown for a pseudo-dynamic discovery is not the only possible one. Further investigation will take place to establish other approaches and to evaluate their feasibility.

6. REFERENCES

[1] Donciulescu C., Vasiu. L, 2004. A requirement for a XML Web Services Security Architecture.
[2] Yang, A., 2002. “XML Web Services Security Issues” – article, http://www.xwss.org/articlesThread.jsp?forum=34&thread=648
[3] Maler E. inverview, 2003. The future of Web Services Security - http://java.sun.com/features/2003/03/webservices-qa.html
[4] The Stencil Group, 2003. “Web Services Rules: Lessons learned from early adopters”. http://www.stencilgroup.com/ideas_scope_wsindex.html
[5] The Stencil Group, 2002. “Understanding Web Services management” http://www.stencilgroup.com/ideas_scope_200206wsmgt.html
[6] Robins B., 2003. “There is no Web Services yellow brick road” http://searchwebservices.techtarget.com/originalContent/1,289142,sid26_gci883333,00.html
[7] Thelin, J., 2002. “Web Services and remote references – an intimidating task?”, http://www.webservicesarchitect.com/content/articles/thelin01.asp
[8] Samtani, G, Sadhwani, D, 2003. Web Services and Peer to Peer computing”, http://www.webservicesarchitect.com/content/articles/samtani05.asp