Core mechanisms for Web Services extensions

Miguel Pardal
Instituto Superior Técnico
Department of Information Systems and Computer Engineering
miguel.pardal@dei.ist.utl.pt

Abstract

Organizations expect Web Services to make their information systems more agile*, so they can better adapt to changes in business requirements. Hence, this technology focuses on interoperability and flexibility giving developers the ability to* customize*,* reuse *and* enhance *Web Service functionalities as well as non-functional extensions such as security, transactions and reliable messaging.*

This paper describes the core mechanisms necessary to build Web Services extensions, regardless of the underlying platform. This contribution is based on the results of a comprehensive evaluation of existing implementations.

1. Introduction

The Internet allows an open and dynamic business environment, where information and communication technologies enable new and innovative ways to work and create value. In this *value web*, Organizations use sophisticated software to connect to their partners [14].

Enterprise applications for the Internet have heavy-duty requirements: users in high numbers and diverse profiles, large volumes of complex data, unsettled business rules, and several integration interfaces with other applications [7].

The main challenge of Enterprise applications is *change*: the needs of the customers change, businesses must change accordingly and so do their systems. Because of this, Enterprise applications benefit from being *agile*, i.e., adapting more easily to requirement changes.

Web Services (WS) [4] and Service-Oriented Architectures (SOA) [13] address the need for agility at the technology and architecture levels, respectively.

1.1. Web Services

A *Web Service* is an access endpoint to data and functional resources of Enterprise applications.

Web Services technology is defined by standards. Figure 1 shows WS-Map [17], a broad and vendor-independent standards index.

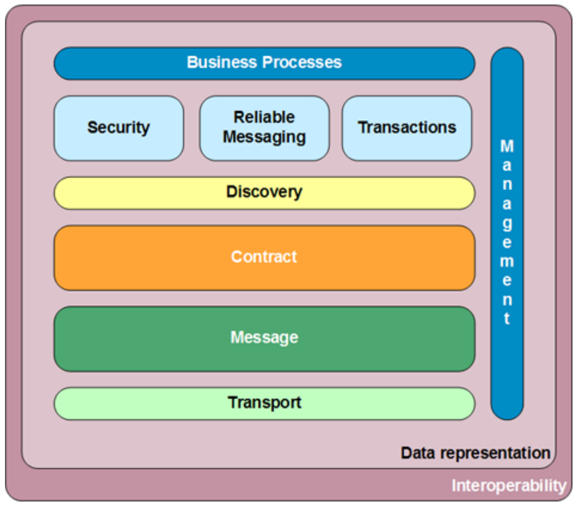

Figure 1. Web Services standards map

Figure 2 illustrates how a client interacts with a Web Service. The service endpoint is published in a directory. The client discovers the service location. The available data and operations are described in XSD (XML Schema Definition) [5] and WSDL (Web Services Description Language) [2]. The client generates invocation stubs that perform run-time data conversion to SOAP [10] message format. Additional requirements are described in WS-Policy [19]. *Extension libraries* are engaged to satisfy these requirements both on the client and on the server, and control data is added to the SOAP message headers. The client invokes the service (using a transport protocol, like HTTP [6] for example) and the service is executed!

XSD, WSDL, WS-Policy and SOAP all use XML [3] as their foundation for data representation.

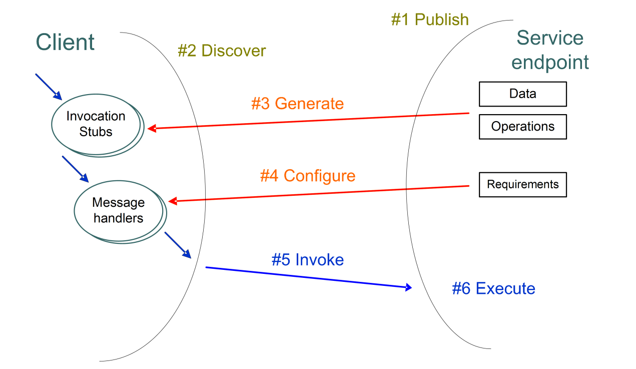

Figure 2. Web Services binding

1.2. Functional and non-functional requirements

Web Service requirements can be classified as *functional* or *non-functional*.

Informally, functional requirements say what a Web Service can do. Non-functional requirements say what properties hold when the Web Service is executed. Non-functional requirements include: security, transactions, reliable messaging, management, usability, and performance. The non-functional requirements of a Web Service can be contradictory, so they must be balanced during implementation.

Let's consider an example Web Service that gives access to an on-line inventory. A functional requirement is "The service allows reading data from the product inventory". A non-functional requirement is "The service interface must be simple to use" and another one is "The service must assure data is kept confidential from non-authorized users". There is a non-functional requirement conflict here, as the service would be easier to use if it didn't need a password, but it would be less secure.

Another issue is Web Service flexibility in non-functional requirements. For instance, service protection can be adjusted to data value: low value messages can use a weaker cipher algorithm than high value messages. Also, service protection can be adjusted to invocation circumstances: a request made from a client inside the corporate network can use a local security credential whereas an external client must use a cross-domain security credential.

1.3. Separating components and aspects

The functional requirements should be implemented as *components*, that can be structured and composed as generic procedures.

The non-functional requirements should be implemented as *aspects*, that allow additional procedures to be executed around or inside components' procedures.

This can be achieved with design patterns [9] that improve relations between components and overall program structure, or with new programming language paradigms, like AOP (Aspect Oriented Programming) [12].

1.4. Web Services extensions

Non-functional requirements can be specified with WS-Policy, but extension libraries are necessary for actual implementation.

The *problem* at hand is to identify the features a platform must have to enable Web Services extensions, keeping components and aspects orthogonal.

In question form: What are the core mechanisms required to support Web Services extensions such as security, transactions and reliable messaging?

2. Core mechanisms for Web Services extensions

We propose the following set of core mechanisms for Web Services extensions:

- Requirements declaration (Reqs);
- Configuration (Config);
- Execution contexts management (Ctx Mgmt);
- Message flow interception (Msg Flow Intercept);
- Operation execution interception (Op Exec Intercept).

2.1. Requirements declaration

The (non-functional) requirements declaration is done with a service policy.

This capability is needed, for instance, to declare that a Web Service can be invoked with transport security or with message security. It can also be used to declare an operation with transactional properties.

2.2. Configuration

The configuration selects the extension mechanisms to engage and the parameters to use or to request from the application in run-time (e.g. which digital certificate will be used to sign messages).

Ideally this configuration should be generated automatically from the client and server policies, after a negotiation.

This capability is needed to control the behavior of the extension libraries.

2.3. Execution contexts management

Execution contexts are an abstraction to organize state variables related to the Web Service. Contexts enable data sharing between the extension library and the rest of the service implementation. Some relevant context scopes are: Application, Session, Operation and Thread.

For instance, the session context allows the storing of a cryptographic key used to store the set of messages in the same security scope. It can also be used to store distributed transaction state, like: id, coordinator location, etc.

2.4. Message flow interception

The message flow interception provides access to the message's routing and contents (headers and body).

This capability allows, for instance, the forwarding of a rejected incoming message, sending it to a security node for reporting. It can also be used to retry sending a lost message, to achieve reliable messaging.

The message flows are usually sequential, but there are proposals, like SPEF (SOAP Profile Enabling Framework) [15], for more elaborate flows.

2.5. Operation execution interception

The operation execution interception allows decision points before the service code is actually executed. The business objects, data objects, stubs and other objects are created in factories that can be customized to return different implementations according to the desired behavior.

Using this feature it's possible, for instance, to implement generic security authorization mechanisms. It's also possible to transparently return transactional or reliable transport implementations of remote Web Service stubs to a client application.

2.6. Overview and dependencies

Figure 3 gives an overview of all the mechanisms for Web Services extensions and shows the dependencies between them.

3. Related work

The presented mechanisms for Web Services extensions were drafted from the results and evaluation of a study [18] about the following implementations:

- WSE 3 (Web Services Enhancements 3) for Microsoft Dot Net 2 [16];
- WSS4J (Web Services Security for Java), for Apache Axis2, for Java [1];

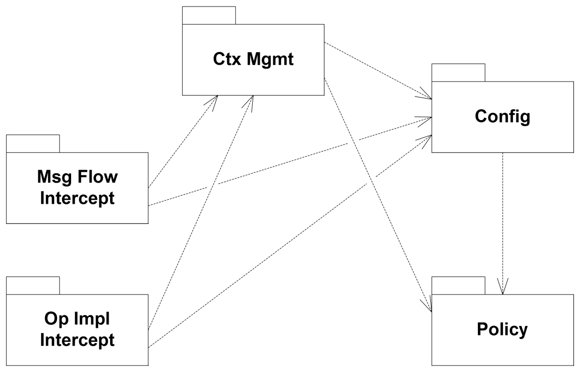

Figure 3. Core extension mechanisms package diagram

- XWSS (XML and Web Services Security), for JAX-WS 2, for Java [21].

The study included extensive tests for each implementation and the development of a prototype for a business case-study.

The selected implementations were biased towards security, but additional tests were performed for transactions and reliable messaging usage scenarios.

After the conclusion of the prototype, a proof-of-concept implementation of the mechanisms was developed for testing and further evaluation in a laboratory project for a Distributed Systems course.

3.1. Case-study prototype

The chosen case-study was "real-estate contracts" and the main functionality supported by the prototype was "signing of sale agreement between seller and buyer".

The full business process and informational entities were modeled using a service-oriented extension of a methodology proposed by Guerra and Pardal [11] for enterprise architecture. The prototype use-cases and interaction diagrams were modeled using UML [8]. The prototype specification and development explicitly accounted for binding, invocation and key distribution, as briefly illustrated in figure 4, and detailed in [18].

3.2. Proof-of-concept implementation

The Web Services extension mechanisms were implemented leveraging existing open-source libraries, primarily JAX-WS (Java API for Web Services) 2 [21].

Requirements declaration was implemented with WS-Policy provided by Apache Commons Policy 1.0 [1].

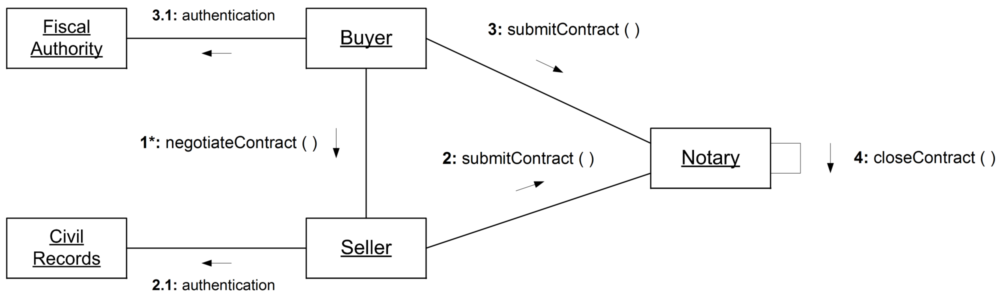

Figure 4. Prototype collaboration diagram

Message flow interception was based on JAX-WS handlers.

Configuration, *execution contexts* and *operation execution interception* were implemented using singleton and factory design patterns [9] with additional custom coding.

3.3. Field tests

The Web Services extension mechanisms implementation was used by 300+ students in a Distributed Systems course's laboratory project, requiring the implementation of a web services application and extension libraries for Web Services security and transactions.

The security extension library supported encryption, MAC (Message Authentication Code) and digital signature [20].

The distributed transactions extension library implemented a "two-phase commit" consensus protocol for transactions with relaxed isolation [22].

The final results were compared with results from a previous course, with similar goals and contents, but without the mechanisms statement and the proof-of-concept implementation.

The new projects were better at separating the application specific code from the extension code than the old ones. Also, the success rate improved from 78.2% to 86.7%, meaning that with the same available time, the students were able to improve the capabilities of the extension.

The results also showed that a change of XML programming approach from data-binding to data-wrapping could yield better extension performance and more control, with less data conversions between XML and objects (and vice-versa).

The field tests showed that the identified mechanisms are *necessary and sufficient* for the development of Web Services extensions.

4. Conclusions and future work

The main goal of *Web Services tools* should be to simplify XML programming. Development tools should focus on Web Service contracts, with direct specification of data schemas, functions and policy, rather than Java or Dot Net centric approaches that map their own concepts to XML, making the contracts less explicit and therefore more difficult to manage and maintain.

This paper presented the *core mechanisms to build Web Services extensions*, regardless of the underlying platform. The clear identification of these mechanisms and its mapping to different Web Services implementations makes it simpler for developers to focus on the extension's added value and to abstract the platform (e.g. Java, Dot Net) specifics.

"Security report" is an example extension for Web Services security. This extension library would produce a report stating all performed message security processing and the used parameters (e.g. keys, certificates). Then, the application could just read the report (oblivious to all the specifics of the standards) and make a final trust decision: accept the request or reject it.

Whenever possible, non-functional requirements should be implemented by extension libraries that share context with applications through meaningful abstractions, to delegate decisions in a simple and effective way. The "security report" proposal is an example of such an extension.

A clear view of the mechanisms regardless of platform, decreases the "cost-of-entry" into extension development, broadens the number of developers that can try new extension ideas and encourages competition and best-of-breed selections, that can further advance the state of the art of Web Services technology.

References

[1] Apache. Securing soap messages with wss4j, 2006.

[2] D. Booth and C. K. Liu. Web services description language (wsdl) version 2.0. W3C, Hewlett-Packard, SAP Labs, 2005.

[3] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (xml) 1.0 (third edition). W3C, Textuality and Netscape, Microsoft, Sun Microsystems, 2004.

[4] F. Curbera, F. Leymann, T. Storey, D. Ferguson, and S. Weerawarana. *Web Services Platform Architecture: Soap, WSDL, WS-Policy, WS-Addressing, WS-Bpel, WS-Reliable Messaging and More*. Prentice Hall, 2005.

[5] D. C. Fallside and P. Walmsley. Xml schema part 0: Primer second edition. W3C, October 2004.

[6] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1. IETF, June 1999.

[7] M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, and R. Stafford. *Patterns of Enterprise Application Architecture*. Addison Wesley, 2002.

[8] M. Fowler and K. Scott. *UML Distilled*. Addison-Wesley, 1999.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[10] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, and H. F. Nielsen. Soap version 1.2 part 1: Messaging framework. W3C, Microsoft, Sun Microsystems, IBM, Canon, June 2003.

[11] M. Guerra, M. Pardal, and M. M. da Silva. An integration methodology based on the enterprise architecture. In *Proc. of the 2004 Conference of the UK Academy for Information Systems (UKAIS 2004)*, May 2004.

[12] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland*, volume LNCS 1241. Springer-Verlag, June 1997.

[13] D. Krafzig, K. Banke, and D. Slama. *Enterprise SOA: Service-Oriented Architecture Best Practices*. Prentice Hall PTR, November 2004.

[14] K. Laudon and J. Laudon. *Management Information Systems*. Pearson Prentice-Hall, 2002.

[15] H. B. Malek and J. Durand. A soap container model for e-business messaging requirements. In M. Kitsuregawa, editor, *Proceedings of WISE 2005*, volume LNCS 3806, page 643652. Springer-Verlag, 2005.

[16] Microsoft. Microsoft web services enhancements (wse) 3.0 documentation, 2005.

[17] M. Pardal. Ws-map: Web services standards map. WWW, November 2006.

[18] M. F. L. Pardal. Security of enterprise applications in service architectures. Master's thesis, Instituto Superior Técnico, September 2006.

[19] J. Schlimmer. Web services policy framework (wspolicy) version 1.2. Microsoft, IBM, VeriSign, Sonic Software, SAP, BEA Systems, March 2006. Editor.

[20] R. E. Smith. *Internet Cryptography*. Addison Wesley, 1997.

[21] Sun. Java web services developer pack. Sun Microsystems Web Site, 2006.

[22] A. S. Tanenbaum and M. van Steen. *Distributed Systems - principles and paradigms*. Prentice Hall, 2003.