Smart Web Services: systems' integration using policy-driven automatic configuration

João C. C. Leitão[1], Miguel L. Pardal[1]

[1] Instituto Superior Técnico, Technical University of Lisbon
Av. Rovisco Pais, 1
1049-001 Lisboa, Portugal
{joaoleitao, miguel.pardal}@ist.utl.pt

Abstract: Web Services (WS) are an important tool for the integration of enterprise applications. With a growing set of WS related standards (WS-*), the technology has become increasingly more complicated to configure and manage, even more so when the Quality of Service (QoS) requirements of the system are changing. This paper presents the results of a study conducted on the ability of the major Web Services implementations to adapt to changing QoS attributes. Their shortcomings are then used as motivation for SmartSTEP, a proposal for a more advanced policy-driven automatic configuration solution.

Keywords: Web Services, Quality of Service, Information Systems Integration, Policy, Automatic Configuration, Java, STEP Framework.

1 Introduction

Enterprise applications have demanding requirements: many users, large volumes of data, ever-changing business rules, and multiple systems' integration interfaces to connect to other applications [1]. The fundamental challenge is *change* so there is great value in techniques that enable information systems to quickly adapt to changes in requirements.

Web Services (WS) [2] and Service-Oriented Architectures (SOA) [3] are a technology and architecture, respectively, which propose *services* as the building block for flexible information systems. WS technology is defined by multiple IETF, W3C, and OASIS standards.

A Web Service is defined as a network access endpoint to resources: data and business functions [2]. Although this endpoint can be accessed in many different ways, the most common is SOAP[1] [4], an extensible XML-based protocol for exchanging information in distributed environments.

1 Although SOAP was initially defined as Simple Object Access Protocol, the 1.2 version of the standard dropped this definition and simply refers to itself as SOAP.

The two major Web Services implementations are Windows Communication Foundation (WCF) [5] and Metro [6]. There are also open-source implementations, such as Apache Axis2 [7].

Recently, these projects have been focusing on the support of WS related standards (WS-*). These were created to extend the Web Services functionalities and capabilities and include standards like WS-Security [8].

Another important WS-* standard is WS-Policy [9], a framework for expressing policies that refer to capabilities, requirements or other characteristics of an entity.

This paper presents the results of an extensive analysis conducted on the Quality of Service (QoS) features of WCF, Metro and Axis2, with special interest in their configuration and limitations. To overcome the identified limitations, a proposal for a new approach and a real world use scenario of its capabilities are also described.

2 Service-oriented ideas

Erl [3] presents the eight principles of SOA: services share a formal contract, abstract underlying logic and are loosely coupled, autonomous, composable, reusable, stateless and discoverable.

In Web Services the formal contract is defined using an XML-based language known as WSDL (Web Services Description Language) [10]. This contract presents all the information that describes a service in a standard *machine-readable* format. The data types and message structures are described as XSD (XML schema definitions), which can be used by code generation tools to create appropriate representations in any supported programming language. This is called a *contract-first* approach, whereas the development of the service's implementation followed by the automatic generation of the WSDL is known as a *code-first* approach [11].

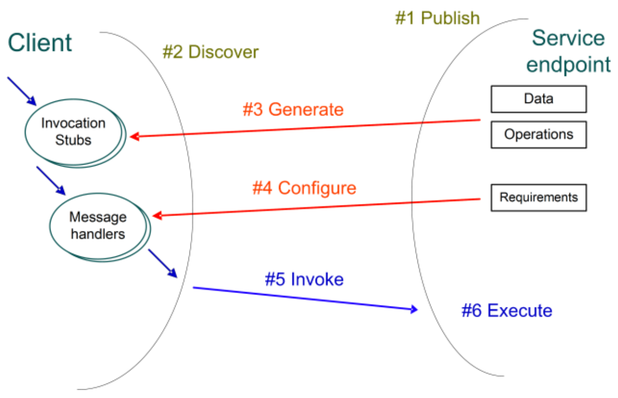

Figure 1. Web Service binding process [12]

Regardless of the chosen approach, the WSDL and other metadata must always be published: as an accessible resource (URL), a service endpoint as defined by

WS-MetadataExchange [13] or using a service metadata repository, like UDDI [14]. This is represented in Figure 1 as step #1 in the Web Service binding process.

Using these mechanisms the clients can retrieve the contract (step #2) and create the necessary code – *stubs* – to convert their data into the format specified by the service (step #3).

Services often have other QoS requirements that need to be met, like security or reliability. These requirements can be stated as policies, which can be used in step #4 to configure *message handlers*, components responsible for executing the required operations to meet the non-functional requirements. Examples include: message ciphering, security token validation, and transactional support.

After the successful configuration, the service can be invoked (step #5) and executed (step #6).

3. Web Services implementations

This section presents how the major WS implementations support WS-* standards and describes their configuration mechanisms. The section ends with a comparison table to summarize their configuration features.

3.1 WCF

WCF [5] is part of Microsoft's .Net Framework since version 3.0 (2006). It bundles several communication technologies, from .Net Remoting to Web Services, supporting several WS-* standards.

In WCF, the entire configuration is done in the *Web.config* file. Using a .Net specific XML-based syntax, one can define the features to use as well as any necessary parameters.

The code-first development is based on this configuration file and code annotations. Visual Studio[2] provides wizards that can be used to create or edit configurations. In a contract-first approach the configurations can be automatically created by code generation tools. Most code generation tools available can interpret policies defined in the WSDL as long as these are already supported by WCF's supported WS-* standards [15].

To extend the platform, one must extend or even override system classes [16]. This extensibility goes as far as creating elements to use in configuration files or defining a new WSDL generator to include custom policies.

3.2 Metro

Metro is Sun Microsystems' Java-based [17] Web Services stack. Version 2.0 was released on November, 2009. The Web Services Interoperability Technology (WSIT)

2 Product home page: http://www.microsoft.com/visualstudio/.

package is built on top of the JAX-WS 2.2 (Java API for XML Web Services) [6] core engine and implements the WS-* standards [18].

Metro's configuration is based on WSDL and *sun-jaxws.xml* configuration file that defines mappings between the contract and the service implementation.

The code-first approach is based on two sources: annotations on service implementation classes and a *wsit-*.xml* configuration file. This file is a simplified WSDL and defines the policies to apply to each supported element (messages, operations, endpoints).

In a contract-first approach, the entire configuration is based on the WSDL and its policies. Some of the platform's specific configuration is also policy-based, using system configuration policies. Metro's code generation tools only support system pre-defined policies, as required by the implemented standards.

Custom policies are ignored in compilation, but prevent the Web Service from being correctly deployed, as they are not recognized by the platform on initialization. In version 2.0 of Metro, custom policies are entirely unsupported.

Any additional behavior should be implemented using JAX-WS Handlers [6] or using the *DeclarativeTubelineAssembler* [19] feature. Another way of achieving similar results is by manipulating Metro's source code to attach a custom module.

One of the new features in Metro 2.0 is *dynamic reconfiguration*, which enables the remote management of a Web Service's policies at runtime. This feature is based on JMX (Java Management Extensions) [20], a Java technology that enables management and monitoring of applications, by dynamically loading and instantiating classes.

3.3 Axis2

Apache Axis2 follows a different approach from the other platforms. It contains the core functionality for Web Services, but the main WS-* standards are available as independent modules [21]. These modules can then be attached to the Axis2 core and used in applications as necessary.

The modules announce the policies they can handle, so that any defined policy can be handled by the proper module. This enables the creation of custom modules to handle any additional behaviors and respective policies. Additional application behaviors can be implemented in a custom *MessageReceiver*, a class that defines how messages should be handled.

Axis2 configuration is based on the *services.xml* file. In a code first approach, this configuration file should define the policies and their targets. In a contract-first development, the code generation tools provided by Axis2 are used to create the configuration file.

The Web Services hosting in Axis2 is also different from any other platform, as it is a Web Application itself. The very Web Services it supports are deployed as modules, which are simple JAR packages. These packages often use the AAR (Axis Archive) file extension, but do not differ from the normal JAR structure.

This is the base for another important feature of Axis2: *Hot Deployment*. Axis2 supports the deployment and initialization of services without having to restart the main Web Application. The attachment of new modules requires redeployment, but their association with the running applications can be made without restarting them.

3.4 Discussion

The following table summarizes the main features of the studied platforms.

Table 1. Features of existing Web Services implementations

Area	Feature	WCF	Metro	Axis2
Policies	WS-Policy	Yes	Yes	Yes
	Custom policies	Yes (1)	No	Yes
	Server-side policy alternatives	No	No	No
Configuration	WSDL-based configuration	No	Yes	Yes (2)
	Runtime policy configuration	No	Yes	Yes
	Automatic reconfiguration (3)	No	No	No
Extensibility	Extensible platform	Yes	Yes	Yes
	Modular platform	No	No	Yes
	Message handler extensibility	Yes (1)	Yes	Yes
	Message handler hot deployment	No	No	Yes (4)

(1). Requires WCF extensions
(2). WSDL and custom configurations
(3). Without user intervention
(4). If available in Axis2 Web Application

Figure 2 proposes a *dynamic binding spectrum* based on the moment where non-functional requirements can be changed, leading to the reconfiguration of message handlers.

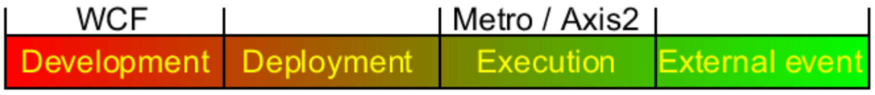

Figure 2. Dynamic binding spectrum

Development and *Deployment* are the implementation and loading phases of the application, respectively. The *Execution* phase includes any action made on the system by some part of the system itself (management interfaces are considered as part of the system so this feature is considered to be in this phase). Anything that acts on the system but is not initiated by it is considered as an *External event* (e.g. incoming message). A system with the ability to reconfigure itself as a reaction to this type of event is normally referred to as a self-adaptive system [22].

WCF is the easiest platform to develop secure and reliable services without great knowledge of WSDL or WS-Policy languages, mainly due to the IDE support. The main disadvantage lies in the lack of runtime configuration support.

Metro and Axis2 both support runtime configuration of policies through management interfaces, which is why they are placed in the execution phase of the spectrum.

Metro has many other features which make it one of the best equipped Java Web Services platforms in use. Other than the unsupported policy extensibility, there aren't many weaknesses. This and other features are planned for upcoming releases. The most important downside of Metro is that it requires some knowledge of WSDL and WS-Policy languages, even in code-first development.

Axis2 is clearly a platform with extensibility and customization in mind, as it can support virtually any feature. Although there are modules supporting the main WS-* standards, there are still many without a public stable implementation. The implementation of a module from scratch is a very complex procedure.

4 SmartSTEP

STEP Framework [23] is an academic open-source multi-layer Java enterprise application framework, with support for Web Applications (Servlet/JSP) and Web Services. Its main design goals are *simplicity* and *extensibility*. The framework's source code is intended to be small and simple enough to allow any developer to read it and understand it thoroughly, as part of a learning process.

SmartSTEP aims to support user-free automatic reconfiguration of QoS capabilities, thus achieving the last phase of the dynamic spectrum: reconfiguration based on an external event (see Figure 2). The proposed feature list is composed by all the features on Table 1, including those unsupported by all studied platforms, namely server-side policies, automatic reconfiguration and handler hot deployment.

4.2 Proposal

In the current version of the STEP framework, message handlers are supported using an *extension engine* [23]. This engine is conceptually similar to the JAX-WS Handlers, but it integrates with other STEP layers, namely the business logic layer (services). The engine executes several extensions sequentially, where each extension manipulates the message that results from the execution of the previous extensions. The execution ends when all required extensions where executed or an error is detected. Currently these extensions are configured using static property files, which prevent runtime modification of the extension sequence or even deployment of new extensions.

This proposal requires a more dynamic approach, so the extensions will be packaged as independent JAR files, following the modular approach used by Axis2. Each JAR will have a specific configuration file to identify a class responsible for the *auto-installation* of the extension.

The extension JAR's should then be placed in a directory that will be periodically checked for new files. Once a new JAR is detected, it will be loaded and the specified installation class will be invoked. This new approach enables the *runtime deployment* of extensions, which can then be used in different applications.

The extension execution sequence should also be dynamic, enabling the usage of a different sequence for different messages. This can be achieved through *factories*, classes that create an extension sequence given a message context. Basic factories should be implemented as part of the framework. The implementation of custom factories should also be supported, thus covering any special configuration scenario.

This feature makes *automatic reconfiguration* possible, as it can create a new extension sequence for each sent or received message.

Policies can be supported by mapping a *policy namespace* to an extension. This association should be done using configuration files, which can be *updated and reloaded in runtime*, without any code manipulation.

When policy alternatives are defined in the server contract, any received message should indicate the alternative used by the client in the outbound processing, so the server can use the correct inbound extension sequence. This indication should be a *header* in the SOAP message, taking advantage of the policy attachments specification [24]. The definition of *server-side policy alternatives* can be a useful feature when a server needs to support multiple configuration scenarios.

5 Use scenario

To demonstrate the usefulness of policy-driven automatic configuration as proposed for SmartSTEP, a real world use scenario was picked: insurance sales.

Many insurance salesmen spend much of their time outside the office, where the prospective clients are. In order to perform their tasks, they need to communicate with the office's main system, which must be prepared to deal with requests that originate inside or outside the corporate network. We will assume that requests from the inside use a different security scheme than the ones from the outside (Figure 3).

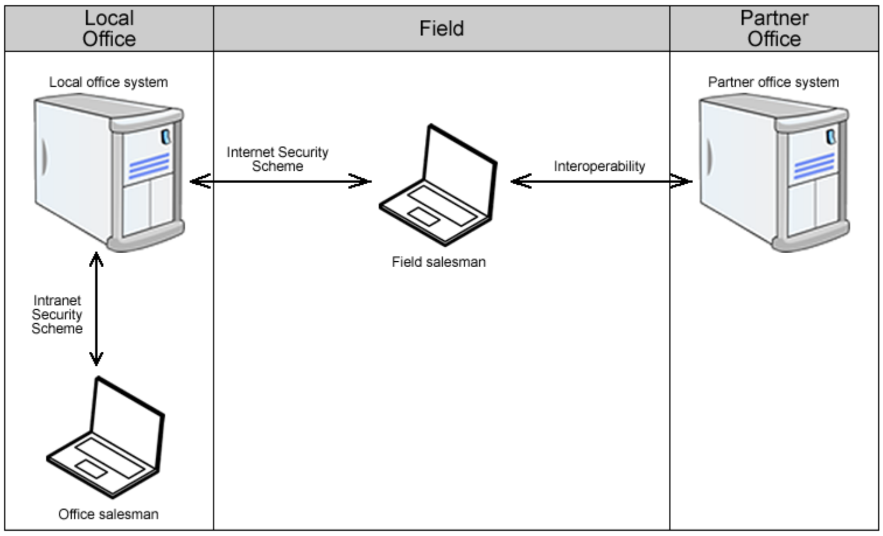

Figure 3. Real world scenario

This situation would require that the office system provided two or more connection points, each with different configurations, creating a significant administrative burden. With server-side policy alternatives, one could define multiple configuration scenarios for one connection point, which would be properly activated whenever a new request was received.

The choice is then made by the client application, which would be prepared to activate the necessary measures according to some environmental parameter. This is possible with automatic reconfiguration and the ability to customize the configuration process to consider the environment.

This customization applies not only to security, but to other requirements. For instance, an application running on a PDA does not have the same available resources as a laptop. These limitations could be considered to create other configuration profiles.

Another important requirement is interoperability (as seen in Figure 3). A salesman might need to retrieve information about a client from a business partner, which might require an unsupported feature. By installing a new extension, the new feature could be ready to use in minutes, without the need for professional technical assistance.

In other platforms, this simple scenario would require multiple applications or extensive configurations, and any update would require professional technicians. With SmartSTEP, applications would be flexible and powerful enough to infer all configurations and *adapt*.

6 Conclusion

Our study of the most popular WS implementations shows that even though they support many WS-* standards and configuration options, they are not as dynamic and extensible as one could wish for. WCF is completely static from the configuration point of view. Metro and Axis2 are more dynamic, but some of their mechanisms are hard to extend or even to work with.

SmartSTEP tries to incorporate the best ideas from the studied implementations into STEP, maintaining its main characteristics: simplicity and extensibility. All proposed features were extensively researched and are considered feasible given the time and complexity constraints.

6.1 Future Work

The next step in the SmartSTEP project is the implementation of the proposed features and their evaluation using the use scenario and performance metrics.

These features open new doors for STEP, making possible to implement new independent modules to support virtually any WS-* standard and possibly publish them in an on-line STEP extension repository, shared by the whole development community, creating new learning opportunities. This would not only be an interesting work from the extensibility point of view, but it can also help in achieving interoperability with other WS platforms.

Acknowledgments

The authors acknowledge the STEP designers and community for their contributions to the framework's source code.
Miguel L. Pardal is supported by a PhD fellowship from the Portuguese Foundation for Science and Technology FCT (SFRH/BD/45289/2008).

References

1. Hohpe, G., & Woolf, B.: *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.* Addison-Wesley Professional. (2003).
2. Alonso, G., Casati, F., Kuno, H., & Machiraju, V.: *Web Services - Concepts, Architectures and Applications.* Springer. (2004).
3. Erl, T.: *Service-Oriented Architecture (SOA): Concepts, Technology, and Design.* Prentice Hall. (2005).
4. Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., Nielsen, H. F., Karmarkar, A., et al.: *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. (2007). From http://www.w3.org/TR/soap12-part1/
5. Lowy, J.: *Programming WCF Services, Second Edition.* O'Reilly Media. (2008).
6. Kalin, M.: *Java Web Services: Up and Running.* O'Reilly Media. (2009).
7. Tong, K. K.: *Developing Web Services with Apache Axis2.* TipTec Development. (2008).
8. Nadalin, A., Kaler, C., Monzillo, R., & Hallam-Baker, P.: *Web Services Security: SOAP Message Security 1.1.* (2006). From http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-errata-os-SOAPMessageSecurity.pdf
9. Vedamuthu, A., Orchard, D., Hirsch, F., Hondo, M., Yendluri, P., Boubez, T., et al.: *Web Services Policy 1.5 - Framework.* (2007). From http://www.w3.org/TR/ws-policy/
10. Chinnici, R., Moreau, J.-J., Ryman, A., & Weerawarana, S.: *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. (2007). From http://www.w3.org/TR/wsdl20/
11. Sosnoski, D.: *"Code First" Web Services Reconsidered*. (2007). From http://www.infoq.com/articles/sosnoski-code-first
12. Pardal, M.: *Segurança de aplicações empresariais em arquitecturas de serviços.* (2006).
13. Davis, D., Malhotra, A., Warr, K., & Chou, W.: *Web Services Metadata Exchange (WS-MetadataExchange)*. (2009). From http://www.w3.org/TR/ws-metadata-exchange/
14. Bellwood, T., Capell, S., Clement, L., Colgrave, J., Dovey, M. J., Feygin, D., et al.: *UDDI Version 3.0.2*. (2004). From http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm
15. *Web Services Protocols Supported by System-Provided Interoperability Bindings*. From http://msdn.microsoft.com/en-us/library/ms730294.aspx
16. Skonnard, A.: *Extending WCF with Custom Behaviors*. (2007). From http://msdn.microsoft.com/en-us/magazine/cc163302.aspx

17. Arnold, K., Gosling, J., & Holmes, D.: *Java(TM) Programming Language, The (4th Edition).* Prentice Hall. (2005).
18. *Metro Specifications*. From https://metro.dev.java.net/guide/Metro_Specifications.html
19. *Declarative Tubeline Assembler One Pager*. From http://wikis.glassfish.org/metro/Wiki.jsp?page=DeclarativeTubelineAssemblerOnePager
20. Kreger, H., Harold, W., & Williamson, L.: *Java(TM) and JMX: Building Manageable Systems.* Addison-Wesley Professional. (2003).
21. *Apache Axis2 Modules*. From http://ws.apache.org/axis2/modules/index.html
22. Heuvel, W.-J. v., Weigand, H., & Hiel, M.: Configurable adapters: the substrate of self-adaptive web services. In *ICEC '07: Proceedings of the ninth international conference on Electronic commerce* (pp. 127-134). Minneapolis, MN, USA: ACM. (2007).
23. Pardal, M., Fernandes, S. M., Martins, J., & Pardal, J. P.: Customizing Web Services with Extensions in the STEP framework. In *International Journal of Web Services Practices, Vol.3, No.1-2* , 1-11. (2008).
24. Vedamuthu, A., Orchard, D., Hirsch, F., Hondo, M., Yendluri, P., Boubez, T., et al.: *Web Services Policy 1.5 - Attachment.* (2007). From http://www.w3.org/TR/ws-policy-attach/