Customizing Web Services with Extensions in the STEP framework

Miguel Pardal, Sérgio Miguel Fernandes, Jorge Martins and Joana Paulo Pardal

***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's design principles focus on interoperability and flexibility to give developers the ability to customize, reuse and enhance functionalities as well as non-functionalities such as security, transactions and reliable messaging.**

In particular, an effective Web Services customization must give application developers simple and expressive ways to program the changes they need without losing any capabilities available in the platform.

We propose customization with Web Services Extensions and present the concept, its core mechanisms and its implementation on the STEP Framework, an open-source multi-layer Java enterprise application framework.

***Index Terms*—Customization, Java, STEP Framework, Web Services.**

I. Introduction

THE internet allows an open and dynamic business environment, where information and communication technologies enable new and innovative ways to collaborate and create value. Organizations use sophisticated software to connect to their partners [1].

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 [2].

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

Manuscript received June 6, 2008. Joana Paulo Pardal is supported by SFRH/BD/30791/2006 PhD fellowship from FCT. Sérgio Miguel Fernandes is supported by SFRH/BD/41857/2007 PhD fellowship from FCT.

M. Pardal is a Lecturer at Department of Computer Science and Engineering (DEI) of Instituto Superior Técnico (IST), Technical University of Lisbon (UTL), Av. Rovisco Pais, 1049-001 Lisboa, Portugal; phone: +351 919 473 933; fax: +351 213 145 843; (e-mail: miguel.pardal@ist.utl.pt).

S. Miguel Fernandes is a Researcher at Software Engineering Group of INESC-ID and a Lecturer at DEI (e-mail: sergio.fernandes@inesc-id.pt).

J. Martins is a Researcher at Software Engineering Group of INESC-ID and a Lecturer at DEI (e-mail: jorge.b.martins@inesc-id.pt).

J. Paulo Pardal is a Researcher at Spoken Language Systems Laboratory (L²F) of INESC-ID and a Lecturer at DEI (e-mail: joana.pardal@l2f.inesc-id.pt).

Web Services (WS) [3] and Service-Oriented Architectures (SOA) [4] address the need for agility at the technology and architecture levels, respectively.

II. Web Services

WS technology is designed for the implementation of Enterprise applications guided by service-oriented principles [5]: Formal contract; Loose coupling; Encapsulation; Composability; Reusability; Autonomy; and Discoverability.

Abiding to all these principles during Enterprise application development is a significant investment in future reuse. The single most important principle is *using formal contracts* to ensure correct client-server integration.

A *WS* is an access endpoint to data and functional resources of Enterprise applications. Fig. 1 exemplifies how a client application interacts with a WS.

The WS endpoint is published in an UDDI [6] directory where a client discovers its location. The available data and operations are described in XSD (XML Schema Definition) [7] and WSDL (Web Services Description Language) [8]. The client generates invocation stubs that perform run-time data conversion to SOAP [9] message format. Additional requirements are described in WS-Policy [10]. 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 [11]) and the service is executed.

XSD, WSDL, WS-Policy and SOAP all are based on XML [12].

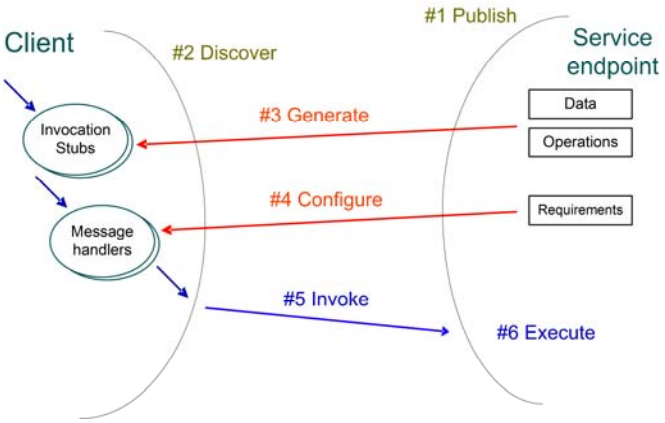

Fig. 1. Web Services client-server interaction.

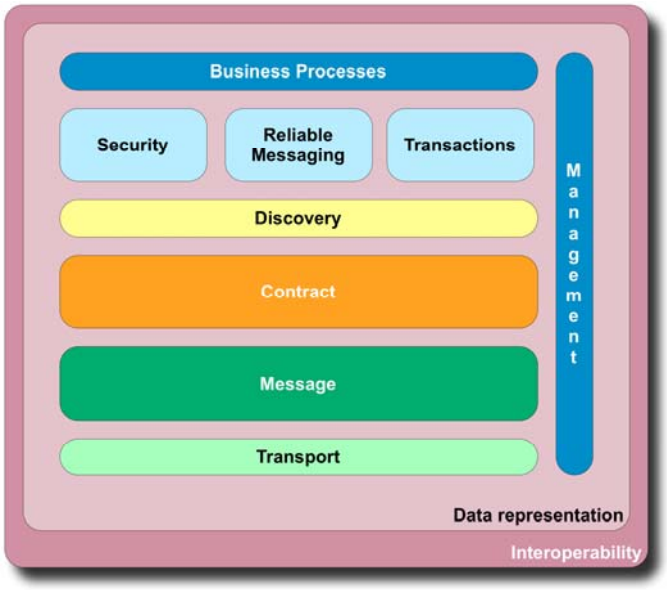

Fig. 2. Web Services standards map.

A. Standards

WS technology is defined by *standards* that leverage Internet network protocols and other open standards. Fig. 2 shows WS-Map [13], a broad and vendor-independent standards index.

WS-Map categorizes WS standards in a set of *categories* that give a sense of the technology's broad scope: Data representation; Transport; Message; Contract; Discovery; Security; Transactions; Management; and Interoperability.

The *data representation* standards address the problem of heterogeneous data representation i.e., how to represent data in a format that is equally understood by everyone. XML is at the base of all other WS standards.

The *transport* standards define ways to establish a communication channel between a client and a WS. The communication can be synchronous or asynchronous, meaning that the client blocks waiting for an answer from the remote WS or not, respectively.

The *message* standards define the structure of the communication units and the ways they can be exchanged between services. SOAP is also a fundamental standard, as it enables message level extension using SOAP headers.

The *contract* standards accurately describe the data, functions, policy and other resources of a WS. They are used for client-server binding.

The *discovery* standards define ways to publish and discover services. UDDI [6] defines a WS directory. WS-MEX [14] is a protocol for WS self-description and meta-data access.

The main concerns for *security* standards are message protection, access control and configuration flexibility. WS-Security [15] states how to protect SOAP messages with XML Signature [16] and XML Encryption [17] and how to transport security-related tokens, like cryptographic keys, digital certificates, assertions, etc.

The *reliable messaging* standards address the reliability of message exchanges in a transport independent way. WS-Reliability [18] and WS-ReliableMessaging [19] are two proposals for assured delivery, duplicate elimination and correct ordering in WS messaging.

The *transactions* standards address the problem of providing well-defined semantics for the combined result of a group of WS operations on distributed resources. There are two proposals for transactions: WS-Coordination [20] and WS-CompositeApplicationFramework [21].

The *business process* standards define development concepts and tools at an abstraction level closer to the needs of business people. An example is WS-BPEL [22] for composing orchestrations of existing WS.

The *management* standards address the problem of keeping a Web Services infrastructure up-and-running.

Finally, the *interoperability* profiles are necessary because of the ambiguities in the standards that result in implementation differences. Each profile defines guidelines, example applications and compatibility test toolkits. The WS-Interoperability organization [23] brings together the main vendors of WS tools in defining interoperability profiles such as basic interaction [24] and security [25].

III. Web Services Customization Challenges

A *custom requirement* is an application-specific variation on a general requirement. Most of the implementation code required to satisfy it is the same as the general case, except for small tweaks.

For instance, consider a *digital signature library* that performs document signature and verification. A custom requirement would be to require the signing principal to belong to a subset of entities.

An effective customization tool must:

- Support functional and non-functional requirements;
- Allow configuration flexibility.

A. Functional and non-functional requirements

WS requirements can be classified as *functional* or *non-functional*.

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

Let's consider an example WS 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 secret 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.

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.

This can be achieved with design patterns [26] that improve relations between components and overall program structure, or with new programming language paradigms, like AOP (Aspect Oriented Programming) [27].

B. *Configuration flexibility*

Another issue is configuration flexibility when supporting non-functional requirements. For instance, service protection can be adjusted to data value: low value messages can use a weaker cipher algorithm than higher 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.

IV. WEB SERVICE EXTENSIONS

WS Extensions are a customization mechanism that provides *interception points* where application developers can add custom code and leverage the underlying WS implementation's capabilities.

First we describe a WS Extension example, then the core mechanisms required for it and finally the proof-of-concept implementation.

A. *Security Report Extension*

The *security report* is an example of a useful WS Extension.

Some applications prefer not to know about security, they just want it to be guaranteed. But others need to know what has been done, for instance, to store audit information in a database.

A security report is produced during WS security processing, containing all performed actions and all used parameters, in a simple, easy-to-use data schema. This effectively leverages the security implementation and enables context sharing through a meaningful abstraction, delegating security decisions in a simple and effective way.

B. *Core mechanisms*

The following mechanisms are required for WS Extensions:

- Configuration;
- Contexts management;
- Message interception;
- Operation interception.

Requirements declaration (Policy) is optional.

Fig. 3 shows the dependencies between packages.

1) *Requirements declaration*

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

The policy states additional requirements that must be fulfilled by the client and by the WS so that the interaction between them can occur as required.

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

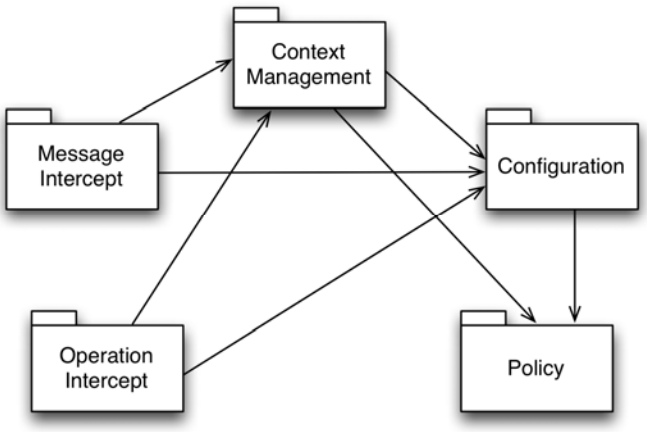

Fig. 3. Extension mechanisms package diagram.

The standard for policy declaration is WS-Policy [10]. A WS-Policy states a set of configuration alternatives supported by a service. The client has to support at least one of the alternatives.

WS Extensions can be a means to satisfy specific requirements stated by a policy.

2) *Configuration*

The configuration selects the extension 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 securely sign messages). This capability is needed to control the behavior of the extensions.

Ideally this configuration should be generated automatically from the client and server policies, after a negotiation. However, a simpler approach is to perform the configuration off-line and then rebuild both the client and the server. This approach also yields better performance, because the policy negotiation is performed in advance.

3) *Contexts management*

Execution contexts are an abstraction to organize state variables related to the WS. Contexts enable data sharing between the extension library and the rest of the application. Some relevant context scopes are: Application, Session, 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.

4) *Operation interception*

The operation execution interception allows interception points before the domain logic 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.

5) *Message interception*

The message flow interception provides access to the

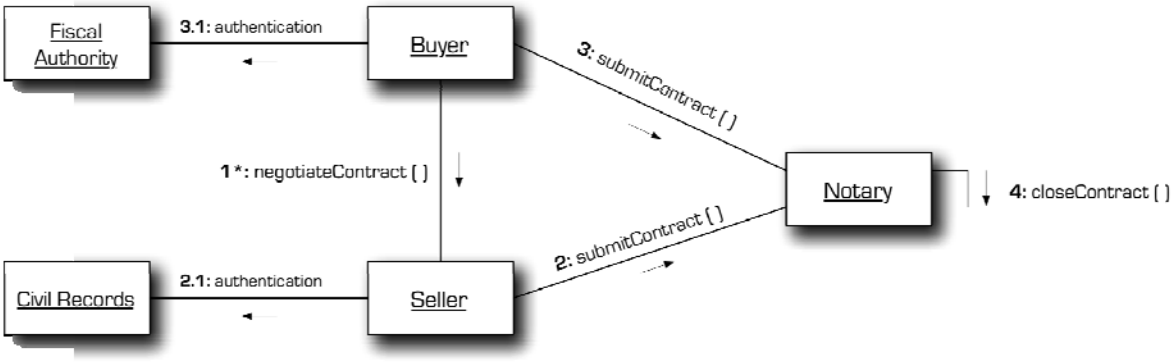

Fig. 4. Prototype collaboration diagram.

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) [28], for more elaborate flows.

C. *Proof-of-concept*

The presented mechanisms for WS Extensions were drafted from the results and evaluation of a study [29] about the following implementations:

- WSE 3 (Web Services Enhancements 3) for Microsoft .NET 2 [30];
- WSS4J (Web Services Security for Java) for Apache Axis2, for Java [31];
- XWSS (XML and Web Services Security) for JAX-WS 2, for Java [32].

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, an Extensions proof-of-concept implementation was developed for testing and further evaluation in a laboratory project for a Distributed Systems course.

1) *Case-study*

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 methodology [33] for enterprise architecture. The prototype use-cases and interaction diagrams were modeled using UML [34]. The prototype specification and development explicitly accounted for binding, invocation and key distribution, as briefly illustrated in Fig. 4, and detailed in [29].

2) *Implementation*

The WS extension mechanisms were implemented leveraging existing open-source libraries, primarily JAX-WS (Java API for Web Services) 2 [32].

Requirements declaration was implemented with WS-Policy provided by Apache Commons Policy 1.0 [31].

Message interception was based on JAX-WS handlers.

Configuration, execution contexts and *operation interception* were implemented using singleton and factory design patterns [26] with additional custom coding.

3) *Field tests*

The WS extension proof-of-concept implementation was used by 300+ students in a Distributed Systems course's laboratory project, requiring the implementation of a WS application and extension libraries for security and transactions.

The security extension library supported encryption, MAC (Message Authentication Code) and digital signature [35].

The distributed transactions extension library implemented a “two-phase commit” consensus protocol for transactions with relaxed isolation [36].

The final results were compared with results from a previous course, with similar goals and contents, but without Extensions.

The new projects were better at separating the application specific code from the customization code.

The field tests showed that the identified mechanisms were *necessary and sufficient* for the development of WS Extensions.

V. Extensions in the STEP Framework

After the proof-of-concept implementation, a more complete implementation was deemed necessary to further develop WS Extensions as a customization mechanism. A more complete implementation was built on top of the STEP Framework[1].

The Extension concept only makes sense in an application domain that is being extended. To properly intercept an application's messages and operation execution, we need a framework that provides a common architecture for applications.

A. *Framework*

The STEP Framework is an open-source, multi-layer Java enterprise application framework with support for Web Applications (Servlet/JSP) and Web Services.

The main design goals of STEP are simplicity and

[1] `http://sourceforge.net/projects/stepframework`

extensibility, and it's been designed for teaching purposes[2].

The STEP framework source code is intended to be small and simple enough to allow any developer to read it and understand it thoroughly, seeing how the multiple layers are implemented in practice.

In its layers, STEP leverages other open-source projects, such as Hibernate[3] for data persistence, Stripes[4] for the web layer, Sun's JAX-WS reference implementation [5] for Web Services, etc. These specific libraries are used in the current version, but different ones have been used before (e.g. Struts[6], OJB[7]) and different ones will probably be used in the future.

The framework aims to be a learning step towards more complete and powerful application frameworks, like Java Enterprise Edition[8] itself, Spring[9], etc.

The STEP Framework is also novel in the way it combines the features of a Web Application framework with a distributed application model, using different domains and providing means to cross physical and trust boundaries.[10]

B. Application layers

The STEP Framework layers are the following (illustrated in Fig. 5): Domain; Service; View; Presentation.

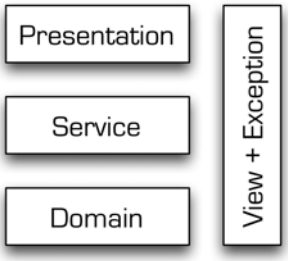

Fig. 5. STEP Framework application layers.

There are also, Web Service (server and client) layers.

Fig. 6 shows the layer's package diagram and its dependencies.

The main application layers are Domain and Service.

Fig. 7 shows a sequence diagram for a STEP Framework application when a request is processed.

The client request is received at the presentation layer, and then it is directed towards a service. The presentation doesn't access the domain directly, only through services and views. Each service has a unit-of-work [2] associated with it. The current implementation of the unit-of-work relies on the underlying database transaction.

[2] The STEP name stands for “Simple, Extensible for Teaching Purposes” (you have to read from left to right and step down and up to form STEP and not SEPT).

[3] `http://www.hibernate.org`

[4] `http://www.stripesframework.org/`

[5] `http://jax-ws.dev.java.net/`

[6] `http://struts.apache.org/`

[7] `http://db.apache.org/ojb/`

[8] `http://java.sun.com/javaee/`

[9] `http://www.springframework.org/`

[10] Organizations (or organization units) have trust boundaries in the sense that they don't fully trust other for all purposes but just for a limited set of interactions and with defined and previously agreed-upon purposes.

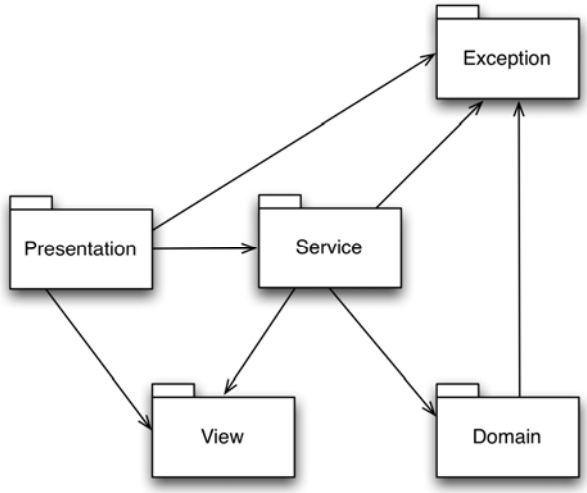

Fig. 6. STEP Framework application layers package diagram.

1) Domain layer

The domain layer is where an object-oriented solution for an application's problem is modeled.

Persistence of the domain objects is essential to ensure the state of an application is properly stored (i.e. data must survive the application instance that created it). We are currently leveraging the Java Persistence API[11] to provide the persistence model for object-relational mapping, using Hibernate as the service provider.

An application should have a single root domain object that represents the application itself and acts as the single entry point to the domain objects. This allows for a seamless navigation through the entire object graph without the need to explicitly query the underlying persistence mechanism. The domain root is typically implemented as a *Singleton* [26] object.

2) Service layer

The service layer (not to be confused with Web Service) mediates the access of presentation to the domain layer.

A service is a class that implements certain functionality through the invocation of domain objects. The architecture mandates that all functionality must be provided by services, isolating the domain model from upper layers. Services are where non-functional requirements that aren't relevant to the domain logic itself should be added.

In order to invoke a service, the invoker must first create a new instance of the service and pass all the necessary data to its constructor. When the service is invoked (through its `execute()` method), the service performs its work within a unit-of-work context. If exceptions occur during service execution, the unit-of-work is always aborted.

Services can be *domain-bound*, if they only invoke domain classes from a single domain, or they can be *multi-domain*, if they use one or more services from different domains. Multi-domain services can also be called orchestrator services.

Services can execute locally or remotely, using WS stubs.

3) View layer

The view layer provides a set of Data Transfer Objects (DTO)[12] that are used to provide/return information to/from

[11] `http://java.sun.com/javaee/technologies/persistence.jsp`

[12] `http://java.sun.com/blueprints/patterns/TransferObject.html`

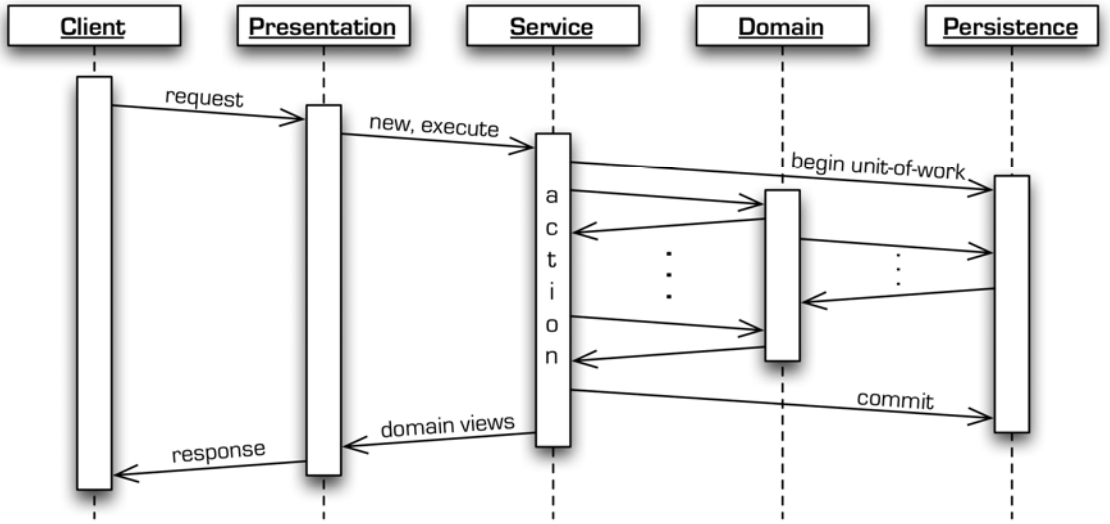

Fig. 7. STEP application layers interaction during a request processing.

services.

The view layer is a read-only set of classes that are mapped to and from the domain. These can be safely returned to a presentation layer, because they don't give direct access to the domain and the underlying database.

Views are described in XSD and the corresponding classes are generated through automated tools. This guarantees the DTO has no logic and simplifies their use in the context of Web Services, as a WSDL can easily import the definitions and reuse them.

4) Presentation layer

The presentation layer is responsible for maintaining user interaction i.e. convert user intents into service executions, report errors in meaningful ways, organize information clearly for humans to understand.

5) Web Service layer

The WS layer is akin to a presentation layer, but its purpose is to allow a domain to be accessed remotely by another application and not by a human user.

For each application service there will be a WS operation declared in the WSDL.

Fig. 8 shows a sequence diagram for a WS, highlighting the interception points used by Extensions. The inbound and outbound SOAP messages are intercepted. The WS operations (i.e., services) are intercepted before and after their execution.

C. Extensions implementation

The Extensions implementation was tested on the platform supported by the current release of the STEP Framework:

- Java 5 (programming language and libraries);
- JWSDP 2.0 (additional XML and Web Services libraries);
- Tomcat 5.5 (Web Application Container);
- Apache Ant 1.7 (build tool);
- ImportAnt 5.5 (build tool library).

The application extension points (including the ones represented in Fig. 8) are the following:

- The *Service super-class* that calls a service interceptor manager before and after the main action for all services[13];
- A *JAX-WS Handler* is used to intercept all SOAP messages and invoke the web service interceptor manager;
- A Web Application `ContextListener` is used to perform the extension engine configuration loading on deploy time[14].

The Extensions Application Programming Interface (API) was designed to use the extension points. Three kinds of objects can be declared in an Extension:

- Service Interceptor;
- Web Service Interceptor;
- Listener.

A *Service Interceptor* is executed before and after the service's main action. The extension itself is executed within the same unit-of-work scope as the service it is extending, and can influence its outcome.

A *Web Service Interceptor* is executed when messages arrive or leave the WS. The extension can influence the message flow.

Finally, a *Listener* is executed when the extension is initialized and destroyed, allowing for resource allocation and release, respectively.

1) Requirements declaration

Policy support was dropped from the current implementation and planned for a later version. Its implementation was considered premature before the underlying configuration mechanism was evaluated and thoroughly tested in practical use.

2) Configuration

The main configuration artifacts are the *properties files*. The main file is called `extensions.properties` and an example is presented in Fig. 9.

The *enabled* option is the main on/off switch. This allows a quick and easy way to disable extensions entirely.

The *list* is where the extension instances are declared. An extension instance is known only after being listed here. The

[13] The Service class hierarchy could be outlined differently if we wanted services that could not be extended, by creating an `ExtensibleService` type, to make the extensibility explicit.

[14] It is useful, but not mandatory, to immediately report if the extensions' configuration was properly initiated, as the extension engine can initialize itself on demand, only when the first interceptor is executed.

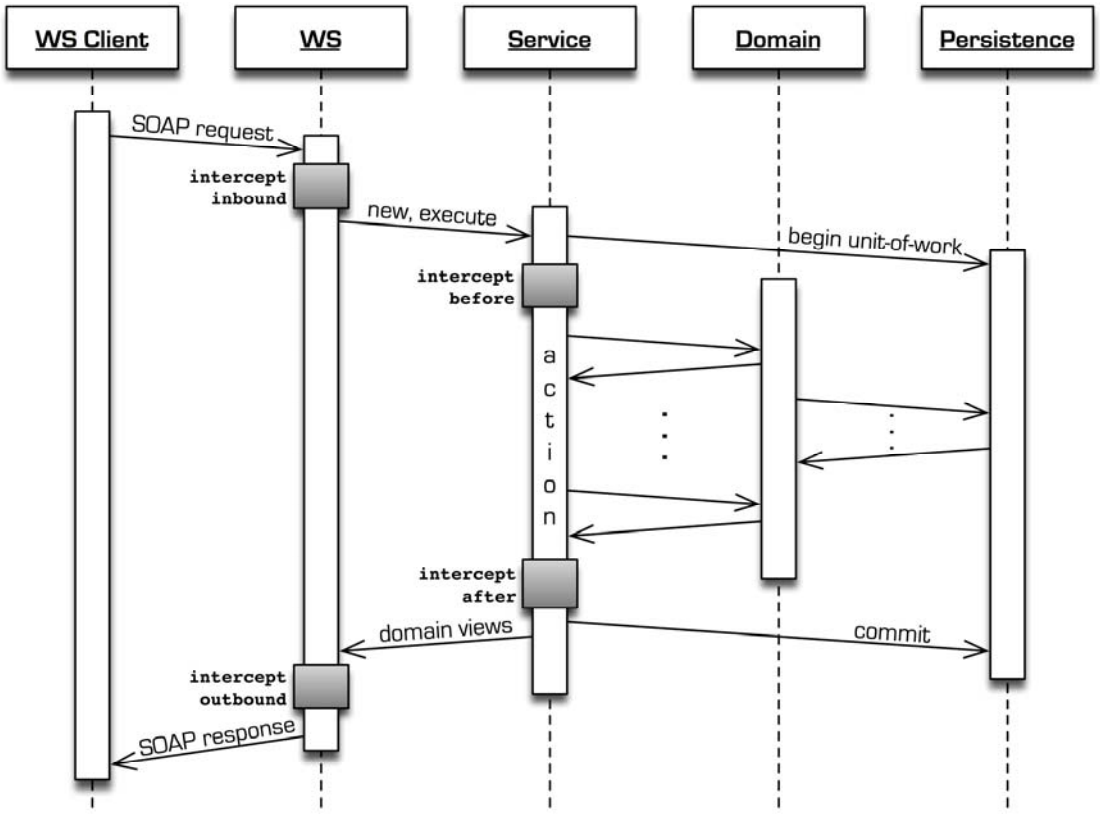

Fig. 8. STEP Framework layers interaction during a web service request processing.

engine initializes the extensions according to the declared

```
extensions.enabled=true
extensions.list=hello,trace,errors
extensions.intercept.service[]=
                hello,errors,trace
extensions.intercept.web-service[]=
                hello,errors,trace
```

Fig. 9. Example `extensions.properties` file.

sequence. In the example, there are three extension instances declared with identifiers: `hello`, `trace`, and `errors`.

After being properly declared, extensions can be used to intercept services and web services.

The *intercept service* properties can have a specifier inside square brackets. It can be:

- **Empty** – all services will be intercepted;
- **Package name** – all service classes inside the package will be intercepted;
- **Class name** – the specific service class will be intercepted.

The property can be specified several times with different specifiers, and the most specific configuration will be selected at run-time. For instance, if a class and package select the same service class, then the class definition is chosen, as it is more specific than the package one.

The intercept list order defines before processing sequence and inverted after processing sequence.

The *web intercept service* properties also can have a specifier inside square brackets. It can be:

- **Empty** – all web services will be intercepted;
- **Namespace** – all web services in the namespace will be intercepted;
- **Service, Port** – the specific web service will be intercepted.

The property can also be specified several times and the most specific configuration will be selected at run-time.

The intercept list order defines outbound processing sequence and inverted inbound processing sequence.

For each declared extension, a `extension-id.properties` file is expected to exist, with additional extension configuration. The `id` is replaced with the declared name of the extension. An example `hello` extension file is shown in Fig. 10.

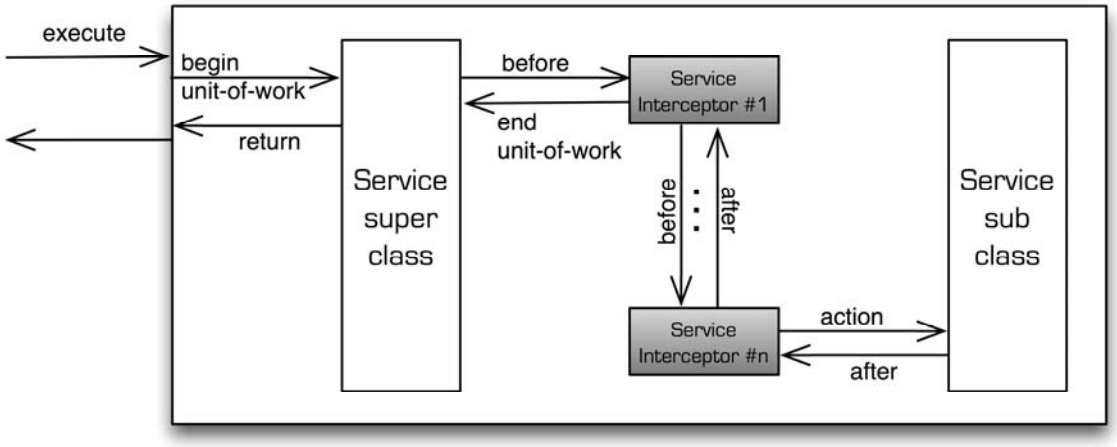

Fig. 11. STEP Framework service interceptor collaboration diagram.

```
extension.enabled=true
extension.service-interceptor=
     extension.hello.HelloServiceInterceptor
extension.web-service-interceptor=
     extension.hello.HelloWebServiceInterceptor
extension.listener=
     extension.hello.HelloExtensionListener
greeting=Hello
```

Fig. 10. Example `extension-hello.properties` file.

The *enabled* option applies only to the current extension.

The *service interceptor* property allows specifying the extension's service interceptor class name. It is optional.

The *web service interceptor* property allows specifying the extension's web service interceptor class name. It is also optional.

The *listener* property allows specifying an extension listener's class name. It is also optional.

Using this configuration flexibility, an extension can combine the following useful capabilities:

- Service interceptor only;
- Web Service interceptor only;
- Service and Web Service interceptor team.

Custom properties can be specified both in `extensions.properties` and in `extension-id.properties`.

3) Contexts management

Execution contexts help to store state variables according to its scope. Extensions have two types of contexts:

- Extension engine context – can be used to share data globally between extensions;
- Extension instance context – can be used to share data inside the same extension team of service interceptor and web service interceptor.

There is no explicit support for contexts with *request scope*. However, a custom solution can be developed using the existing extension contexts by using a request-specific identifier as a map key. The Framework also has generic context management classes, with scopes: Application (global), Session (externally managed identifier) and Thread (thread identifier is used implicitly to distinguish between contexts).

4) Operation interception

The WS operations are mapped to services for their actual execution. A Service Interceptor (SI) intercepts service executions as illustrated in Fig. 11.

A Service Interceptor can access the service instance data by using Java Reflection or by casting the service instance to a known type.

The error behaviors for a service interceptor are the following.

Throw a `DomainException`

The unit-of-work is aborted and the presentation layer receives a domain exception that is indistinguishable from one thrown by the service itself.

Throw a `ServiceInterceptorException`

The unit-of-work is aborted and a `ServiceException` is thrown to report a system condition.

Throw a `RuntimeException` or `Error`

The unit-of-work also aborts.

5) Message interception

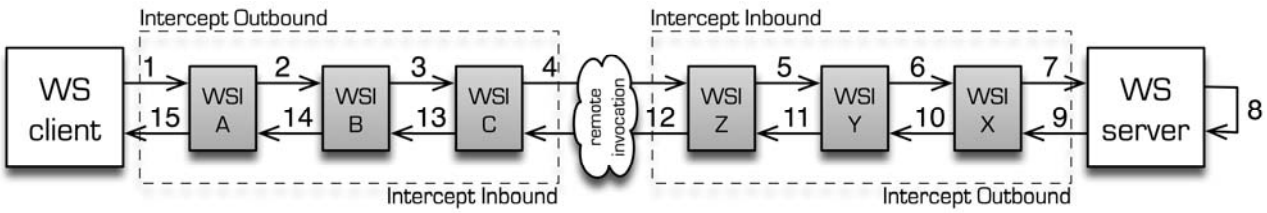

Fig. 12. STEP Framework web service interceptor collaboration diagram.

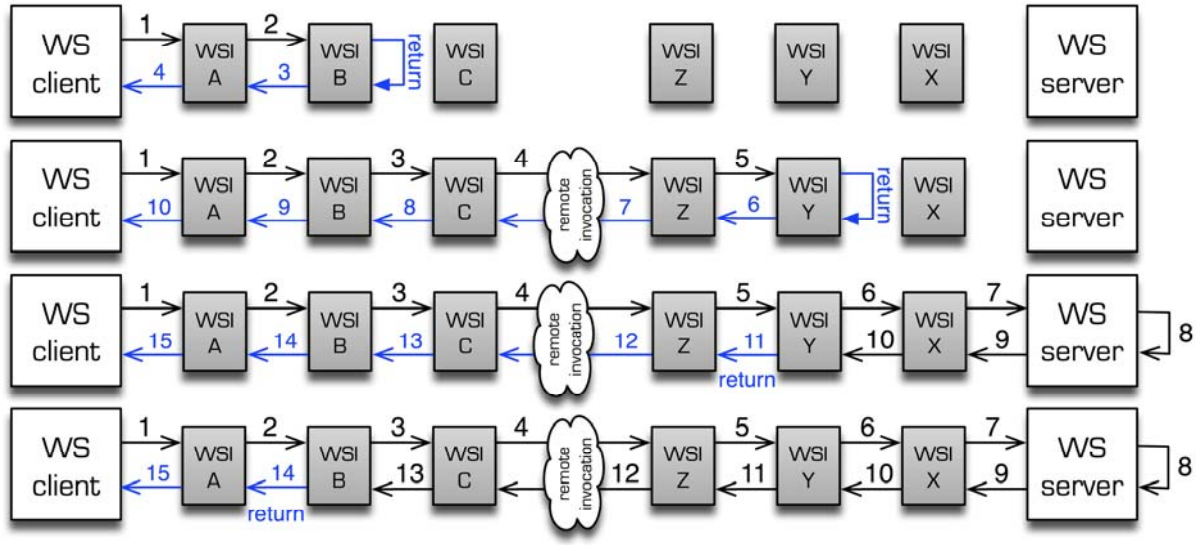

Fig. 13. Web service interceptor returns `false`.

The message interception gives access to the SOAP message's header and body. A Web Service interceptor (WSI) intercepts all web service messages as illustrated in Fig. 12. The SOAP message context enables the interceptor to access the SOAP message.

The message situation can also be queried:

- Is it outbound or inbound?
- Is it a fault message?
- Is it being intercepted on the server-side or on the client-side?

To modify the message flow, a WSI can:

Return `false`

If the message is moving towards the server, it is reversed to go back to the client, as illustrated in Fig. 13.

The WSI is responsible for setting the message contents appropriately.

Throw a `SOAPFaultException`

If the message is moving towards the server, it is reversed to go back to the client, as illustrated in Fig. 14.

The message body is replaced with the SOAP fault provided by the exception.

Throw a `WebServiceInterceptorException`

If the message is moving towards the server, it is reversed to go back to the client, as illustrated also in Fig. 14.

The message body is replaced with a SOAP fault that is created automatically containing the text message specified in the exception.

Throw a `RuntimeException` or `Error`

The message processing is interrupted, as illustrated in Fig. 15. If the error occurs on the server-side, a SOAP fault message will be sent to the client.

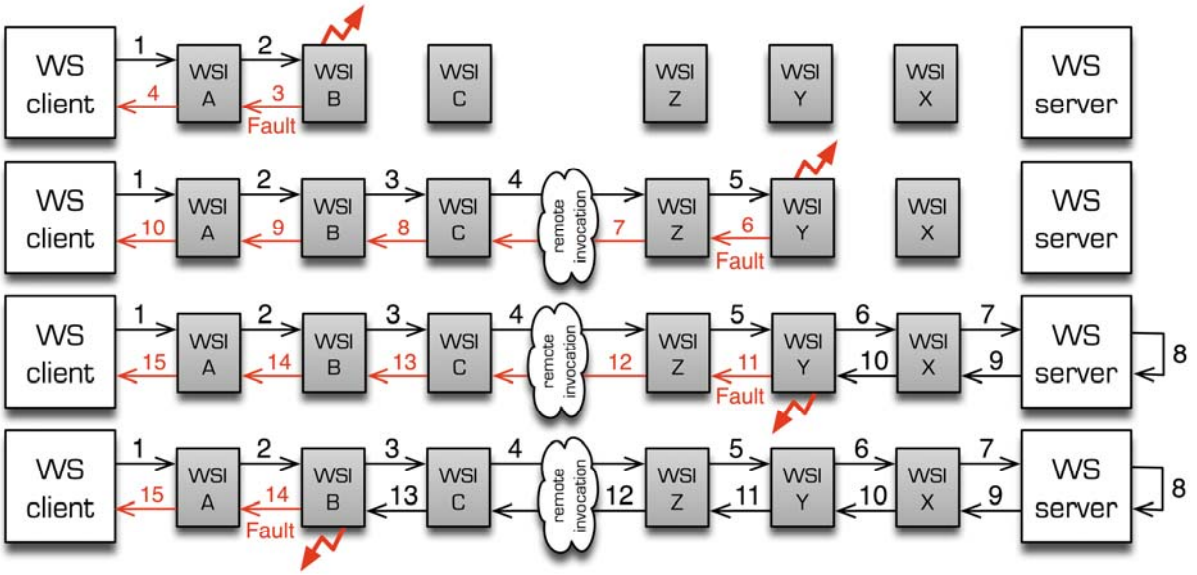

Fig. 14. Web service interceptor throws a `SOAPFaultException` or a `WebServiceInterceptorException`.

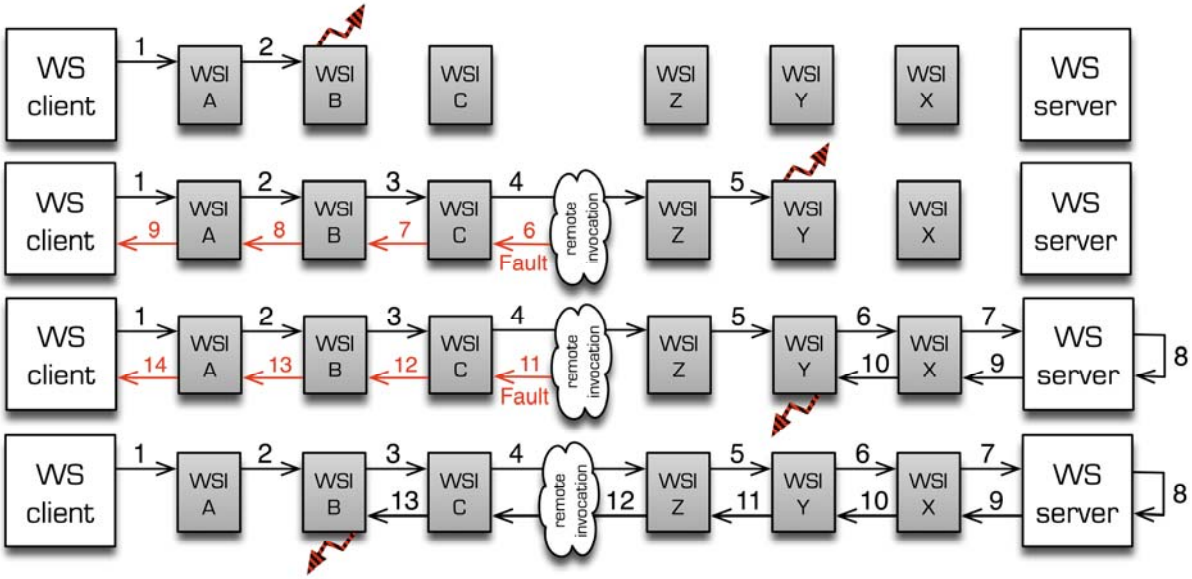

Fig. 15. Web service interceptor throws a RuntimeException or an Error.

VI. Conclusions and Future Work

The main goal of *Web Services & XML tools* should be to empower developers and to simplify programming. The tools should focus on contracts, with specification of data schemas, functions and policy, rather than Java-centric approaches that map other concepts to XML, making the contracts less explicit and therefore more difficult to manage and maintain.

The layered architecture is also very important to the *separation of concerns* that makes practical customization possible.

We presented *Web Services Extensions*, an interception-based approach to customization.

The clear identification of the core mechanisms and its mapping to different WS implementations makes it simpler for developers to focus on the extension's added value and to abstract the platform specifics.

Extensions support both functional and non-functional requirements, and allow flexible configuration. Whenever possible, custom requirements should be implemented by Extensions that share context with applications through meaningful abstractions, to delegate decisions in a simple and effective way.

These features make Extensions a very attractive tool for teaching purposes, because students can focus on the more advanced technological aspects, leveraging an easy-to-use configuration mechanism.

Future work on Extensions in the STEP Framework will be directed towards supporting customization of more capabilities available in the underlying platform as the current implementation uses only the core Web Services stack. Policy support is also a future concern. Interoperability aspects like message transports beyond HTTP and a .NET platform implementation are also planned.

Extensions decrease the "cost-of-entry" into WS customization, broaden the number of developers that can try new ideas and encourage competition and best-of-breed selections that can help practitioners to further advance the state of the art of Web Services technology.

References

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

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

[3] 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.

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

[5] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*, Prentice Hall, 2005.

[6] L. Clement, A. Hately, C. von Riegen, and T. Rogers "UDDI version 3.0.2" OASIS, Systinet, IBM, SAP AG, Computer Associates, 2004. `http://uddi.org/pubs/uddi-v3.0.2-20041019.htm`.

[7] D. C. Fallside and P. Walmsley, "XML Schema part 0: Primer second edition", W3C, October 2004. `http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/`.

[8] D. Booth and C. K. Liu, "Web services description language (WSDL) version 2.0", W3C, Hewlett-Packard, SAP Labs, 2005. `http://www.w3.org/TR/2005/WD-wsdl20-primer-20050803`.

[9] 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. `http://www.w3.org/TR/2003/REC-soap12-part1-20030624/`.

[10] J. Schlimmer, "Web services policy framework (WSPolicy) version 1.2", Microsoft, IBM, VeriSign, Sonic Software, SAP, BEA Systems, March 2006, editor.

[11] 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. `http://www.w3.org/Protocols/rfc2616/rfc2616.txt`.

[12] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible markup language (XML) 1.0 (3rd edition)", W3C, Textuality and Netscape, Microsoft, Sun Microsystems, 2004. `http://www.w3.org/TR/2004/REC-xml-20040204`.

[13] M. Pardal, "WS-Map: Web services standards map", http://web.ist.utl.pt/miguel.pardal/ws-map, November 2006.

[14] F. Curbera and J. Schlimmer, „Web services metadata exchange (WS-MetadataExchange)", MSDN, Microsoft, IBM, Computer Associates, SAP, BEA Systems, Sun Microsystems, webMethods, September 2004. `http://msdn.microsoft.com/ws/2004/09/ws-metadataexchange/`.

[15] A. Nadalin, C. Kaler, P. Hallam-Baker, and R. Monzillo, "Web services security: SOAP message security 1.0 (WS-Security 2004)", OASIS, IBM, Microsoft, Verisign, Sun, March 2004. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss`.

[16] D. Eastlake, J. Reagle, and D. Solo, "XML-Signature syntax and processing", W3C, February 2002. http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/.
[17] D. Eastlake and J. Reagle, "XML encryption syntax and processing", W3C, December 2002. http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/.
[18] K. Iwasa, "Web services reliable messaging TC WS-Reliability 1.1", OASIS, Fujitsu Limited, Novell, Inc., Oracle Corporation, Sun Microsystems, November 2004. http://docs.oasis-open.org/wsrm/ws-reliability/v1.1.
[19] C. Ferris and D. Langworthy, "Web services reliable messaging protocol (WS-ReliableMessaging)", Microsoft, IBM, BEA, TIBCO Software, February 2005, editors.
[20] M. Feingold, "Web services coordination (WS-Coordination) version 1.0", IBM, Microsoft, Hitachi, Arjuna Technologies, IONA, August 2005, editor.
[21] M. Little, "Web services composite application framework (WS-CAF) version 1.0", Sun, Oracle, IONA, Arjuna, Fujitsu, July 2003, editor.
[22] S. Thatte, "Business process execution language for web services version 1.1", Microsoft, IBM, Siebel Systems, BEA, SAP, May 2003, editor.
[23] WSI, "Interoperability: Ensuring the success of web services – an overview of WS-I", WS-I Web Site, 2005. http://www.ws-i.org/about/Default.aspx.
[24] C. Ferris, C. K. Liu, M. Nottingham, P. Yendluri, M. Gudgin, K. Ballinger, and D. Ehnebuske, "WS-I basic profile version 1.1", WS-I, Microsoft, IBM, SAP, BEA Systems, webMethods, August 2004. http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html.
[25] A. Barbir, M. Gudgin, M. McIntosh, and K. S. Morrison, "WS-I basic security profile version 1.0", WS-I, Nortel Networks, Microsoft, IBM, Layer 7, August 2005. http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html.
[26] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.
[27] 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. http://www2.parc.com/csl/groups/sda/publications/papers/Kiczales-ECOOP97/for-web.pdf.
[28] 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 643–652, Springer-Verlag, 2005.
[29] M. Pardal, *Security of enterprise applications in service architectures*, Master's thesis, Instituto Superior Técnico, September 2006. http://mflpar.googlepages.com/MScMflp20060908.pdf.
[30] Microsoft, "Microsoft web services enhancements (WSE) 3.0 documentation", 2005. http://msdn.microsoft.com/webservices/webservices/building/wse/default.aspx.
[31] Apache, "Securing SOAP messages with WSS4J", 2006. http://ws.apache.org/axis2/modules/rampart/1_0/security-module.html.
[32] Sun, "Java web services developer pack", Sun Microsystems Web Site, 2006. http://java.sun.com/webservices/.
[33] M. Guerra, M. Pardal, and M. M. da Silva, "An integration methodology based on the enterprise architecture", in *Proceedings of the 2004 Conference of the UK Academy for Information Systems (UKAIS 2004)*, May 2004. http://mflpar.googlepages.com/GuerraPardalUkais2004.pdf.
[34] M. Fowler and K. Scott, *UML Distilled*, Addison-Wesley, 1999.
[35] R. E. Smith, *Internet Cryptography*, Addison Wesley, 1997.
[36] A. S. Tanenbaum and M. van Steen, *Distributed Systems – principles and paradigms*, Prentice Hall, 2003.

Miguel Pardal (M'2007) became a Member (M) of IEEE in 2007. He was born in Lisbon, Portugal in 1977. He graduated (2000) and mastered (2006) in Computer Science and Engineering from Instituto Superior Técnico (IST), Technical University of Lisbon, Portugal. His MSc topic was "Security of Service-Oriented Enterprise Applications". In 2008, he started his PhD at IST, on the subject "The Internet of Things".
He has been a researcher and lecturer at IST since 2002, teaching distributed systems, enterprise applications integration and operating systems courses. Before his return to academia, Miguel worked from 2000 to 2002 as a consultant for Unisys, in major Banking and Insurance projects. His work was recognized in 2004, when he was awarded the best computer science admission work to the Portuguese Engineers Association (Ordem dos Engenheiros). He has authored 6 publications. His main research interests include web services, service-oriented architectures, enterprise integration technologies, RFID and sensor networks.
Eng. Miguel Pardal is a member of ACM.

Sérgio M. Fernandes was born in Lisbon, Portugal in 1978. He graduated (2001) and mastered (2006) in Information Systems and Computer Engineering from Instituto Superior Técnico (IST), Technical University of Lisbon, Portugal. His MSc thesis focused on "A Workflow Virtual Machine". He began his PhD studies in 2007 on "A Programming Model for Long Transactions in Web Applications".
Since 2002, he has been a Researcher for INESC-ID's Software Engineering Group. He his also a Lecturer at IST teaching Compilers, Algorithms and Data Structures, Software Engineering, and Advanced Programming. He has participated in the COMBINE and ACE-GIS European Research Projects.

Jorge Martins (M'2001) became a Member (M) of IEEE in 2001. He was born in Lisbon, Portugal in 1978. He graduated (2001) and mastered (2006) in Information Systems and Computer Engineering from Instituto Superior Técnico (IST), Technical University of Lisbon, Portugal. His MSc thesis focused on "Monitoring Support in WorkSCo". He began his PhD studies in 2007 on "Composition traceability within Software Product Lines".
Since 2002, he has been a Researcher for INESC-ID's Software Engineering Group. He his also a Lecturer at IST teaching Computer Architecture, Algorithms and Data Structures, Software Engineering, and Software Quality. He has participated in the ACE-GIS European Research Project. His main research interests include software architectures, software composition and software product lines.

Joana Paulo Pardal was born in Lisbon, Portugal back in 1978. She got a "Licenciatura" in informatics and computer science engineering (a 5 year full-time degree), artificial intelligence in 2001 from Instituto Superior Técnico (IST), Technical University of Lisbon, Portugal. She received an MSc degree in 2004 also from IST. In 2006 she started her PhD work also at IST on "Dynamic Integration of Ontologies in Practical Spoken Dialogue Systems", with Nuno J. Mamede, H. Sofia Pinto and James F. Allen as advisors.
She is a researcher at the Spoken Language Systems Laboratory (L²F) of INESC-ID since 2001. She participates on «Dialogs on Dialogs» Reading Group at CMU. She is a Lecturer at IST since 2002, teaching object-oriented programming and design patterns, knowledge representation, artificial intelligence, autonomous agents and multi-agent systems, and distributed systems. She participated in several national projects. In 2004 she was invited for a training period at GRIL, at Université Blaise Pascal, Clermont Ferrand, France. In 2006 she spent the Fall term at the Computer Science Department, University of Rochester, NY, USA as part of the Continuous Understanding project. In 2001 she received an MSc fellowship from FCT (Portuguese National Science Foundation). In 2007 she received a fellowship from ACL to added a Doctoral Consortium at Rochester, NY, USA. She holds a PhD fellowship from FCT. She authored 1 book chapter and 16 papers. Her main research interests include spoken dialogue systems, natural language processing, machine translation, ontologies, semantic web services, distributed systems and software engineering.
Eng. Joana Paulo Pardal is a student member of ISCA (International Speech Communication Association), ACL (The Association for Computational Linguistics) and AAAI (Association for the Advancement of Artificial Intelligence) since 2007.