The Audition Framework for Testing Web Services Interoperability[*]

Antonia Bertolino and Andrea Polini
Istituto di Scienza e Tecnologie della Informazione “Alessandro Faedo” – CNR
Via Moruzzi, 1 – 56124 Pisa (Italy)
{antonia.bertolino, andrea.polini}@isti.cnr.it

Abstract

Service Oriented Architectures and Web Services are emerging technologies, which have overall inherited problems and advantages from the component-based approach, but exacerbated the aspects of loose coupling, distribution and dynamism of "components", here elements furnishing published services on external client requests. In this paper we highlight the urgent need for methodologies supporting Web Services reliable interaction, and in particular deal with testing concerns. We then propose a framework that extends UDDI registry role from the current one of a "passive" service directory, to also sort of an accredited testing organism, which validates service behaviour before actually registering it. This testing stage (called audition) mainly focuses on interoperability issues, so to facilitate the coordination among services registered at the same UDDI. The audition needs to rely on a Web Service specification augmented with information on how the service has to be invoked. We propose that this information is given in the form of a Protocol State Machine, which is a newly introduced behaviour diagram of the UML 2.0.

1. Introduction

Enterprise Application Integration (EAI) is today one of the hottest topics within both the academic and industrial communities. In general terms the objective is to develop methodologies for integrating applications running on different platforms in a distributed setting and possibly hosted by different organisations. Research in EAI continuously evolves as a consequence of the restless introduction of new communication technologies (such as the mobile paradigm) and consequently of the increasing opportunities of linking together more and more information systems.

In the 90's a strong effort has been sustained to provide powerful and easy ways to develop and integrate distributed software applications. The most remarkable example is probably the CORBA initiative [2], which produced an open standard and widely accepted middleware specification on top of which building a distributed objects application almost "easy" as a local one. This technology in fact provides the software developer with quite powerful means to address problems originated by distribution and heterogeneity.

In the last years the evolution and pervasive presence of the Web made it plausible to imagine a further level of integration, in which not only could the applications involved be distributed on different hosts and written in different languages, but also be owned and run by different organisations. The above scenario has been the main motivation behind the emergence of the Service Oriented Architecture (SOA) paradigm, of which Web Services technologies constitute at the same time the most relevant instantiation, and the best sponsors. To characterise SOAs, we summarise below the main aspects from the W3C list of properties [6]:

Logical view: The service typically carries out a business-level operation, described in terms of what it does, and at an abstract, logical level.

Message orientation: The service only defines the messages exchanged between provider agents and requester agents. Any detail about how an agent implementing a service is constructed is deliberately abstracted away. A key benefit of this concerns legacy systems, as an existing software component or application could be "wrapped" in message handling code so to adhere to the formal service definition.

Description orientation: To support the public nature of the SOA, a service must describe by means of machine-processable metadata all and only those details that are important for the public use of the service. The semantics of a service should also be documented, either directly or indirectly, by its description.

Granularity: Services generally provide a small number of operations with relatively large and complex messages.

Network orientation: Services tend to be oriented towards use over a network.

[*]This work has been supported by the European project TELCERT (FP6 STREP 507128).

Platform neutral: Messages are sent through the interfaces in a platform-neutral, standardised format, most often XML.

Different related technologies and standards are used to achieve the properties listed above in the case of Web Services (WS)s. In particular the WS architecture is currently based on three main elements:

Web Service Description Language (WSDL) [7]: this is a XML based language used to describe a Web Service. Information in a WSDL file specify offered services, access points, format of in/out parameters and accepted mechanisms used to exchange messages;

Universal Description and Discovery Integration (UDDI) [8]: this open technology, which specification is developed by the OASIS consortium [1], defines a common mechanism to publish and retrieve information about available Web Services. It acts as a registry and can catalogue information following the yellow, green or white pages paradigms;

Simple Object Access Protocol (SOAP) [12]: this is a simple protocol that can be used to exchange XML based messages, and it is the natural choice to establish communications among Web Services. The protocol is generally implemented over *http* but other implementations are possible (e.g. *smtp*).

Given the technologies listed above, a WS has been characterised by the W3C consortium using the following definition [6]:

> *A Web Service is a software system designed to support* ***interoperable*** *machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web Service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialisation in conjunction with other Web-related standards*.

This definition stresses the importance of WSs to support interoperable communications between software applications. The relevance of interoperability is also highlighted by the emergence of the WS-I initiative [2], a consortium grouping the leading organisations in the WS community. WS-I aims at introducing specific rules and profiles that should reduce interoperability problems, at least at the message format level. In particular WS-I has defined a basic profile that specifies several different rules, also on formatting aspects, and relations that must hold between the pieces of information contained in related WSDL files, SOAP messages and UDDI entries, so to facilitate the communication between different services.

[1]details at: http://www.oasis-open.org/home/index.php

[2]details at: http://www.ws-i.org

However it is generally recognised that current technologies are not adequate to permit the reliable integration of services developed by different stakeholders. In particular, the technology relies on the restrictive assumption that a developer of the service's clients knows in advance the semantic of the operations provided by a service or other properties of it [3]. Such information, in fact, does not appear in any standardised form in the current WS architecture. Trying to address this evident weakness in the current technology, different approaches are being proposed to improve the format of WSs specifications, so to also include the integration level between interacting WSs. We are currently at the stage in which these proposals need to aggregate consensus within the community. Beyond the different technical solutions, we can recognise some common basic ideas, that of increasing the information that should be provided with a WS, such as input/output relations and the ordering of operation invocations, and of providing a language to specify the choreography depicting coordination scenarios between different WSs. Major examples of such technologies are the Business Process and Execution Language for Web Services (BPEL4WS) [4] and the Web Service – Choreography Description Languages (WS-CDL) [14].

In our research we also ask the WS developer to augment the WSDL definition with additional information. Our objective is to increase the testability of such artifacts and in particular make it possible the application of rigorous model based testing methodologies to the dynamic assessment of WS interoperability.

We believe in fact that while the community is moving towards identifying a common standard model for WSs architecture, allowing for WSs smooth combination and interoperation, it is important to raise awareness within the same community that also common standard methods for verification and validation of functional and non-functional properties of WS must be contextually sought and agreed upon. In fact, as we also develop later in the paper, the properties of WS-based application make their testing and analysis very difficult, therefore we can afford such tasks only if we plan ahead for them, and in particular only if we proactively impose some discipline in WS specification so to later facilitate testing from the interested stakeholders (see Section 4). As this augmented specifications of course require extra-effort on the developer's site, it is important that WS testability aspects are sustained and agreed upon by the community.

Hence the goal of this paper is twofold: on one side, we would like to push awareness in the WS community at large of testing and QoS relevance for WSs and sustain the claim that these activities need to rely on a rigorous augmented WS specification; on the other side, we overview a general framework we are developing to this purpose, which embraces and extends currently existing proposals into a comprehensive organisation, inclusive of both technical aspects and procedural ones.

In our idea the WS information can be used to create a sort of "improved UDDI registry" that tests the WSs as they require to be registered. This assessment is aimed at verifying that the WS under registration can correctly interact with the WSs already registered that will be possibly invoked. This process requires some overhead on the side of the UDDI registry, but should reasonably guarantee that all the services registered within the same registry can correctly inter-operate.

After overviewing related work in Section 2, in Section 3 we discuss in more detail the information that needs to be included with the WS definition. In Section 4 we speculate on different objectives and scenarios for WS testing. In Section 5 we then present our Audition testing framework. Finally we draw up conclusions in Section 6.

2. Related Work in Testing of Web Services

WSs testing is an immature discipline in intense need of further research by academy and industry. Indeed, while on the practitioner's side WSs are evidently considered a key technology, research in the area seems not to draw an adequate attention from the testing community, probably due to the contiguity/overlap with other emerging paradigms, especially with component based software engineering (CBSE), or perhaps to the quite technical details that this discipline entails. In this section we give a brief overview of those papers that share some similar views with our work.

The possibility of enhancing the functionality of a UDDI service broker with logic that permits to perform a testing step before registering a service has been firstly proposed in [18] and [19], and subsequently in [13]. This idea is also at the basis of the framework introduced in this paper. However the information we use and the tests we derive are very different from those proposed in the cited papers. In particular while in the cited works testing is used as a means to evaluate the input/output behaviour of the WS that is under registration, in our work we mainly monitor the interactions between the WS under registration with providers of services already registered. In this sense, we are not interested in assessing if a WS provided is bug-free in its logic, but we focus on verifying that a WS can correctly cooperate with other services, by checking that a correct sequence of invocations to the service leads in turn to a correct interaction of the latter with other services providers, (and that vice versa an incorrect invocation sequence receives an adequate treatment).

With reference to the information that must be provided with the WS description, the authors of [18] foresee that the WS developer provides precise test suites that can be run by the enhanced UDDI. In [13] instead the authors propose to include Graph Transformation Rules that will enable the automatic derivation of meaningful test cases that can be used to assess the behaviour of the WS when running in the "real world". To apply the approach they require that a WS specifically implements interfaces that increase the testability of the WS and that permit to bring the WS in a specific state from which it is possible to apply a specified sequence of tests.

The idea of providing information concerning the right order of the invocations can be found in different way also in specifications such as BPEL4WS and WSCI. The use of such information as main input for analysis activities has been also proposed in [11]. However the objective of the authors in this case is to formally verify that some undesired situations are not allowable given the collaboration rules. To do this the authors, after having translated the BPEL specifications into Promela (a language that can be accepted by the SPIN model checker), apply model checking techniques to verify if specific properties are satisfied. Another approach to model based analysis of WS composition is proposed in [10]. From the integration and cooperation of WSs the authors synthesise Finite State Machines and then they compare if the obtained result and allowable traces in the model are compatible with that defined by BPEL4WS-based choreography specification.

3. An Enriched Information Model for WS Interoperability Testing

Speaking in general, the capability of testing a software artifact is strongly influenced by the information available. In fact different kinds of testing techniques can be applied depending on the extent and formalisation degree of the information available. The technique applied will also be different depending on which is the aspect that testing aims at evaluating, such as functionality, QoS, interoperability.

The notion of the SOA, as we have seen, establishes rigid limitations on the kind of documentation that can be provided and used for integrating services. In particular, a service must not include information on how it has been implemented. This obviously is desirable to enable the decoupling between requesters and providers of services, but obviously makes integration testing more difficult.

Although similar problems have been encountered and tackled in the area of software components, testing of WSs is probably even more difficult since the different elements participating to the interaction could be dispersed among different organisations, so even a simple monitoring strategy or the insertion of probes into the middleware is not generally feasible. In CBSE different proposals have been made to increase the information available with software components generally using a metadata approach [17]. Fortunately, the insertion of additional information documenting a WS is perhaps easier thanks to established mechanisms in the UDDI registry such as tModels [8].

On the other hand, these mechanisms are used by technologies cited above such as BPEL4WS and WS-CDL. These specifications suggest to add, among the other data, information concerning the protocol that a client needs to follow to correctly interact with the service itself. We claim however that it is useful to attach this description in the form of an XML Metadata Interchange (XMI [1]) file, since in this form it can be easily reused by UML based technologies. XMI, in fact, is becoming the de facto standard for enabling interaction between UML tools, and it can be automatically generated from widespread UML editing tools such as IBM Rational Rose XDE or Poseidon. Hence our proposal is that the WS description (including the WSDL file) will report some additional information documented by the WS developer in UML, and in particular, as we explain below, as a Protocol State Machine, that is a UML behaviour diagram newly introduced into the latest version of this language [9]. In particular an XMI file representing the associated PSM could be inserted in the UDDI registry.

3.1. Protocol State Machine Diagrams

A Protocol State Machine (PSM) is a particular kind of state machine that focuses only on the transitions of states and on the rules governing the execution order of operations. This kind of diagram has been introduced in the UML 2.0, in order to support component-based development by providing clear rules to describe the communications interactions between different objects. Therefore in the general case a PSM, when associated to a class, provides rules for implementations of its interfaces or ports; these rules provide the guidelines that other systems must comply with in order to correctly work with the associated class [9].

Hence a PSM does not specify a detailed behaviour for the associated object, although a standard behaviour state machine relative to the object can conform to a PSM accepting, as correct, transitions between states specified by the rules in the PSM. Special features distinguishing a PSM are that PSM states can have a name but cannot show entry actions, exit actions, internal actions, or do activities; PSM transitions show operations but not actions or send events; they can also have pre-conditions and post-conditions, shown in square brackets.

Pre- and post-conditions provide a strong testing capability to the model we are figuring, by introducing verification concepts typical of the Design by Contract (DbC) methodology firstly introduced by Meyer [15]. Following the DbC paradigm a predicate can be associated to each invocation and the former must be satisfied before the invocation can be performed. In such a case after the invocation of the method the post-conditions are guaranteed to be satisfied. Such verification mechanisms have already showed their potential in the area of component based software in which they have been used for many different verification tasks.

As an example, in Figure 1.a we report the definition of a simple interface for the management of a file. The interface only defines the methods to open the file, read a line, write a line and finally close the file. On the base of such interface, in Figure 1.b we report a simple PSM that defines the protocol that a user of the service defined by that interface must follow to correctly interact with an implementation of that same service (note the annotation [pre-condition]operation/[post-condition] on the transition arcs).

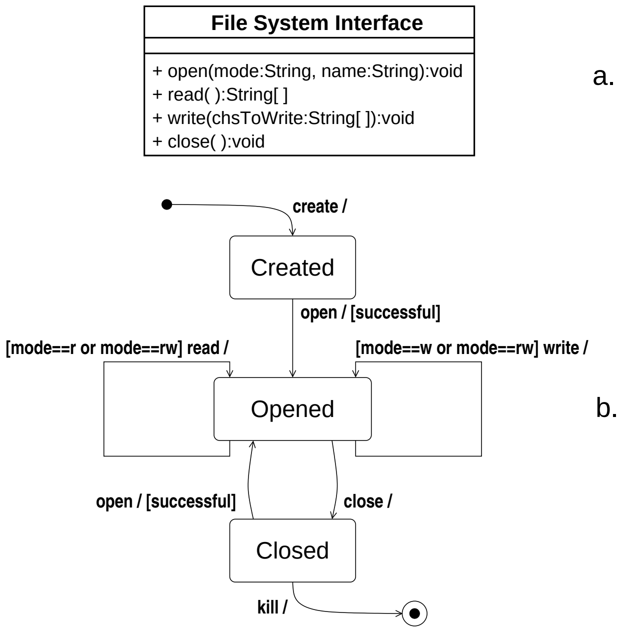

Figure 1. Definition of an interface for file management (a.) and the corresponding PSM (b.)

4. WS Testing Scenarios

As said, WS is a recent technology and as a consequence a discipline for WS testing still needs to be developed. In particular a well defined testing process should clearly state the verification steps and the way through which a WS, or a collection of cooperating services, must undergo so to provide sufficient guarantees on their behaviour.

We leave out of the scope of the present paper the proposal of a comprehensive testing process. As a first step towards the definition of such a process, however, we believe a classification is useful of both the different objectives in the testing of WSs, and the different stakeholders that can be involved in this process.

We start from identifying the stakeholders who could be interested in testing WSs and their coordination. We think there are mainly three of them:

- **WS developer**: as for any other software element, the developer is clearly interested in assessing if the produced results match with the expected one, in terms of functionality, interaction with other components and probably QoS. In the WS architecture, we can distinguish mainly between two different artifacts: WSs that will be registered in a UDDI server to successively provide the service to other services, and stand-alone clients that access to registered services.
- **Service Broker Provider**: these are agencies that act as service brokers and could be interested in testing a WS before granting registration in order to be self guaranteed on the offered services.
- **Standards Body**: these are agencies/consortia that in particular domains define standards or generally accepted specifications. The potential of WS technologies convinced some of these organisations (consider for instance IMS in the e-learning domain[3]) to adopt and promote such technologies and so they are currently defining precise interfaces for specific services in order to increase the chance that independently developed service will be able to correctly inter-operate.

Considering these three stakeholders, several possible different scenarios become available for testing.

Testing on the WS developer's side. The developer will be more interested in testing its clients in terms of functionality trying to find bugs in its implementation. This process will involve different steps and techniques, reasonably in a similar way to the traditional testing of software artifacts. In particular the tester being the same developer, they can take advantage from the knowledge of specifications and of the internal structure, and test to obtain for instance a particular coverage on the branches of the client code. The execution of the tests will also require the development of precise stubs for the accessed services. The difficulty in deriving effective stubs is directly proportional to the quantity of information that is made available. As a subsequent step the developer may want to assess its software product while interacting with the required services. This steps is particularly difficult if we consider that in the SOA context required services generally run and reside on a remote machine on which the WS developer generally has no control. This is probably one of the biggest differences with respect to integration testing in CBSE in which some control on the component can be supposed on the tester's side. Similar to the scenario depicted above is the testing by its developer of a WS that will be made available to other clients.

[3]details at: www.imsglobal.org

Testing on the Service Broker Provider's side. The registration of a service is obviously a critical point for a broker that wants to guarantee the quality of the registered services. Some authors suggest to use test cases provided by the same developer of the WS [19]. But this approach could not provide completely objective test suites (for instance, if a sort of accountancy is performed against WS usage, the test cases could be produced in a "lazy" way to increase the profit gainable from their usage). In our opinion, it would be more practical and effective to enable the derivation of test cases from models provided by the WS developer. Or, it could be equally useful to derive test cases from a specification defining coordination scenarios in which the WS under registration plays a role. Another interesting scenario for the Service Broker Provider emerge when a service already registered change its implementation. Obviously it is equally important for the Service Broker Provider to re-test the service in order to guarantee the possible already registered client of that service.

Testing on the Standards Body's side. Finally the third stakeholder that we consider is an authority (either normative or because recognised by the community) that releases some WS specifications to which service providers should be compliant (in order to be accepted in the market). In this case the test suite must be developed directly by the authority and (successfully) executed on the service that wants to claim conformance. Depending on the model defined for conformance we can have a more or less strict level of conformance with consequently different levels of obtainable interoperability.

Obviously the different scenarios discussed above can be present and variously intermixed in any different testing process. For instance, considering the third case, the WS conformance test suite should be published by the Standard Body together with the WS prescribed specification, and is executed first in-house by the WS developer (in combination with the developer's own test cases) and after by a testing laboratory acting for the certification authority.

The difference in each case stays on the testing focus that will strongly influence the choice of the test cases to run (using a metaphor from fishing or hunting we can say that the kind of fish or quarry that we want to catch strongly influence the choice of the decoy).

5. A Framework for WS Testing

We now finally introduce our framework for testing WSs. The framework relies on an increased information model concerning the WS, as illustrated in the previous sections, and is meant for introducing a test phase when they ask for being published on a UDDI registry. In this sense we call it the “Audition” framework, as if the WS undergoes a monitored trial before being put “on stage”. It is worth noting that from a technical point of view the implementation of the framework does not present major problems and even from the scientific perspective it does not introduce novel methodologies; on the contrary one of its target is just to reuse complex software tools (such as test generators) in a new context. The major difficulties we foresee is that a real implementation based on accepted standards requires that slight modifications/extensions are made to such standard specifications as UDDI. This in turn requires wide acceptance from the WS community and the recognition of conformance testing importance.

Figure 2 shows the main elements of the framework. The figure provides a logical view, i.e., the arrows do not represent invocations on methods provided by one element, but a logical step in the process and point to the element that will take the responsibility of carrying on the associated operation.

The process is activated by the request made by a WS of being enclosed in the entries of a registry and is structured in eight main steps, which are also annotated in Figure 2 (numbers in the list below correspond to the numbers in the figure):

1. a web service WS1 asks a UDDI registry to be published among the services available to accept invocations. Contextually, WS1 provides references to: the WSDL file defining the syntax of the offered service, and to the XMI file codification of a Protocol State Machine (expressing the protocol that a possible client should follow to correctly interact with the service). If the description for the specified service to be provided is already registered only a reference to the WS access point is provided;

2. the UDDI service puts WS1 in the associated database, but marking the registration as a pending one. Successively it starts the creation of a WS tester (see Section 5.1). To do this we foresee in the framework a particular WS that encapsulates a test generator engine. The idea is to reuse test generators already developed or that will be made available in the future. In particular for our purpose we look for generators that take as input UML 2.0 diagrams. To the best of our knowledge at the moment no PSM based tool has appeared yet, nevertheless we think that tool developed in the area of protocol conformance could be profitably adapted, and this will constitute our future research work.

3. the *WS Testing Client* will start to make invocations on WS1, acting as the driver of the test session. Our main objective developing the framework has been to have a means to check the order of invocations made between two different services so to discover interoperability problems. Nevertheless it is natural to imagine a generator engine based on the concepts of Design by Contract (DbC) [15], which on the base of pre- and post-conditions will generate test cases to check that, given an input satisfying to the pre-conditions, the test execution ends up with outputs satisfying to the post-conditions.

4. during the audition, unless it is a basic service, i.e. a service that does not require to cooperate with other services to provide its functionality, WS1 will ask to the UDDI service for references to other services necessaries to complete the provision of the service;

5. UDDI checks if the service asking for references is in the pending state. If not the reference for the WSDL file and relative binding and access point to the service are provided. In the case that the service is in the pending state the UDDI will generate, using a WS factory, a WS Proxy for the required service. This WS proxy will implement the same interface of the required service, and its creation take as input the PSM for the simulated WS (or other information depending on the kind of checks that we want to monitor);

6. For each inquiry request made by WS1 the UDDI service returns a binding reference to a Proxy version of the requested service.

7. WS1 on the base of the reference provided by the UDDI service will start to make invocations on the Proxy versions of the required services. As a consequence the Proxy version can check the content and the order of any invocation made by WS1; In particular in our case the checks that will be executed concern the ordering of the invocations made by the client service. If an error occurs which violates the checks currently monitored by the Proxy the UDDI service will be informed. As a consequence the directory service removes from the pending entries the service currently under test, and denies registration;

8. If the current invocation it is conform the Proxy service invokes the real implementation of the service and return the result obtained to the invoking (WS1) service. Then the process continues driven by the invocations made by the testing client.

5.1. Framework Testing Objective

We discussed above the architecture of the framework mainly in terms of the steps that constitutes the approach, the different parts and corresponding necessaries interactions. We discuss here briefly on the kind of tests that we

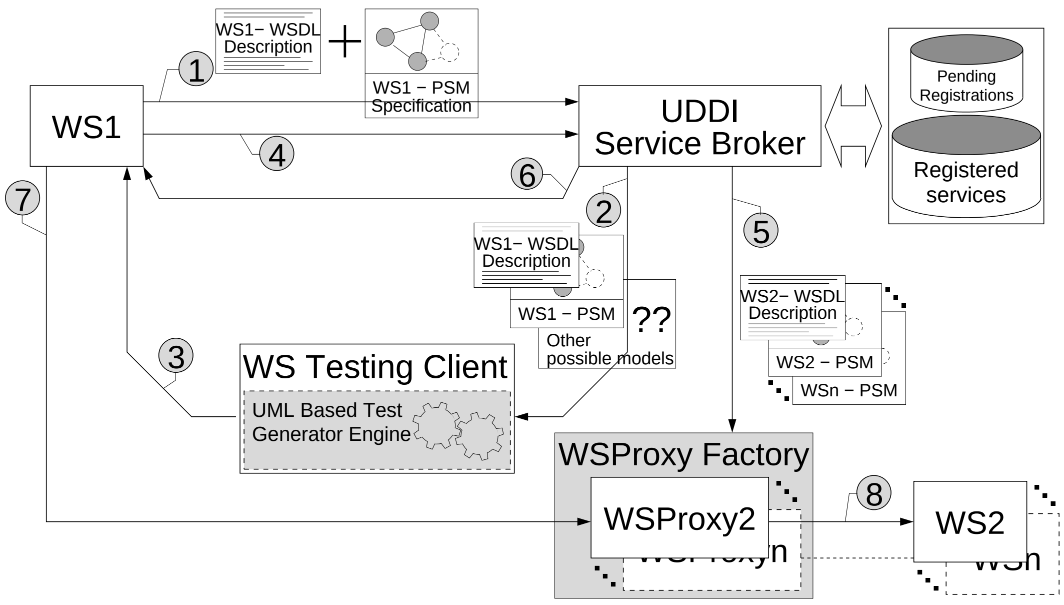

Figure 2. The Audition Framework

think to use, in particular with reference to the scenarios discussed in Section 4.

The development of the framework has been initially pushed by the necessity from a "standards body" of having a technique for testing conformance to interaction specification among different services. In this context we suggested that a first step could be the introduction of PSM specification that being a UML diagram resulted in a more acceptable tool for such kind of community. For this reason the framework is mainly focused on the use of Protocol State Machine and has as main objective the testing of interoperability issues. As a consequence the test generator engine that acts as the main audition driver needs to be able to derive meaningful test case sequences from the PSM specification. In particular two different strategies, with two different objectives can be followed:

- **disciplined client strategy**: in this case the service under test will be invoked following sequences admitted by the specification of its PSM. The framework will check in this case that all the interactions with other services follow correct scenarios.
- **malicious client strategy**: being WS dynamically invoked by unknown clients, their ability to react in "graceful" way to incorrect invocations is of basic importance as well as the ability to serve correct ones. Robustness becomes of major importance and the malicious strategy intends to evaluate such property by intentionally generating invocation sequences that do not correspond to those admitted by the PSM for the WS under test.

Therefore the Audition framework fits in test scenarios in which the test cases are defined by an external authority that intends to check the conformance of a WS to a particular context in which the available specifications define interactions rules. However the framework can also be easily modified to consider the derivation of different scenarios. The element that must be changed in each case is the test generator engine that must be obviously chosen on the base of the available models defining the specification.

On the other hand, the same framework can be used by a Service Request Broker agency (UDDI) to evaluate how a service interacts with the other provided services. In this case the PSM can be either directly provided by the developer of the service itself with the associated WSDL description or specified as the reference standard for the service that the WS under test claims to provide.

5.2. Considerations on the Framework

After introducing the main steps that the proposed framework for WS testing should take, we provide here some specific reflections, following the step ordering:

1. in the first steps we propose to add to the service specification the definition of the PSM for the provided service. As discussed in Section 3 this is certainly in line with what proposed by specifications such as BPEL4WS or WSCI [5], and moreover we discussed the benefits of adopting an approach based on XMI format. It is interesting to note that more than one PSM could be associated to the same service. However this

is not a difficulty for the framework, since an incorrect behaviour results from an execution that is not following any of the paths specified by the different state machines for the same service;

2. the second steps requires that the UDDI maintains a record of the services for which the registration is still pending. This is not a major problem since a flag in the table of the registered services is enough to introduce this feature. However to enable the testing process it is necessary that the UDDI service requires authenticated inquiry in order to recognise if the service is still in a pending state or not. If so it will activate the generation of Proxy service, if not it will return to the service the requested reference. Another important point that needs to be carefully evaluated is the performance burden that will mainly affect the inquiry operation. In fact as a first step of an inquiry the UDDI needs to check the status of the invoking service. For this reason it is probably better to have a different database of the pending requests that likely will be quite small;

3. the proxy services that we included in our version only check, at this stage, for constraints that can be expressed within a PSM. However we think that with other information model for the WS description, the framework can still be applied. At the same time the WS Testing client is limited at this stage to the generation of meaningful inputs and does not make any kind of verification on the output produced by the service under test. On the other hand the automatic generation of an oracle is not easy and certainly requires more accurate information on the service, as discussed in Section 4;

4. as shown in Figure 2, the proxy service makes an invocation on a real and running WS during a test session. In this manner the testing process does not need to generate stub versions of the required services. As the reader can imagine this behaviour does not raise major issues if the service do not modify persistent data, as for instance it is the case for web services provided by the Google search engine. However if the service manages data in a persistent storage the invocations resulting from a test session certainly will lead to major business problems (simply think to airline services for booking seats). In this case two possible solutions can be figured out: 1) the WS can be invoked within a test session and in this case it returns pre-ordered consistent information without accessing to the associated data storage; 2) the WS proxy must encapsulate also a stubbed version of the service.

6. Conclusions

This paper presented our ongoing research on testing and analysis of WSs. In overviewing current topical issues in development of SOAs and in WS interoperability, we have discussed the generally agreed requirement that the open specification of a WS be augmented with additional information than the currently provided public interfaces, e.g. WSDL. This is for instance pursued by the recent BPEL4WS and WS-CDL technologies. In particular, to permit reliable coordination among interacting WSs, the specification should include the correct protocol to access the WS, which could then be used for model based testing. To maximise interchangeability of specifications, we proposed to adopt the de facto standard of UML 2.0, and in particular the newly introduced Protocol State Machine, which provides the right level of abstraction for describing the prescribed ordering of operation invocations. We have discussed how quite different testing concerns may regard WS-based applications, and we have attempted a preliminary classification based on the involved stakeholders. Then, we have proposed a general framework in which the UDDI registering role is extended to also play the role of an external testing organism which validates the WS conformity to the published (augmented) interface, and its ability to interact with other available services. The underlying idea is that before being registered at a UDDI, a WS must pass an “audition”, in which its dynamic behaviour is tested, especially regarding coordination aspects in the interaction with other services.

This is of course ongoing work at a preliminary stage. Future work will include a more comprehensive classification of testing issues regarding WSs and possibly the inclusion, with each identified testing stage, of techniques for test generation. In particular we are thinking of a technique combining Category-Partition with state-based over the PSM. We also plan to investigate the evident analogy of testing from the PSM with the ALTS-based test generation, which has been earlier proposed in the domain of Software Architecture [16]. In fact, we believe that the two diagrams (PSM and ALTS) rely on the same philosophy of identifying the relevant operations for testing purposes, and “abstract away” irrelevant ones for the purpose of test case generation. Then, of course, we plan to implement a prototype environment to support and validate the proposed Audition framework.

As a conclusive remark, we are aware of the fact that in this quickly evolving field of WSs, it can be easy to come with a new perhaps appealing idea (such as our audition framework) and just launch it. The difficulty is in making these ideas realisable and effective, also compatible with what is already established practice. Above all, to win the challenge of a pervasive service-oriented information society, it is highly important that open standards and rigorous disciplines are agreed upon and enforced in the development of publicly available services. There is neither room nor need in this landscape for improvised solutions or personalised interests. Our proposed Audition framework is intended as a small, humble contribution to raise awareness of WS interoperability issues and to gather efforts towards a rigorous accreditation of behaviour adequacy before a service is made publicly available, and it is only in this spirit that we proposed it.

References

[1] XML Metadata Interchange (XMI) Specification ver. 2.0. http://www.omg.org/docs/formal/03-05-02.pdf, May 2003.

[2] AA.VV. Common Object Request Broker Architecture (CORBA), ver. 3.0.3. downloadable from: http://www.omg.org/cgi-bin/doc?formal/04-03-12, March 2004.

[3] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services – Concepts, Architectures and Applications*. Springer Verlag, 2004.

[4] T. Andrews et al. Business Process Execution Language for Web Services (BPEL4WS) – ver. 1.1. dowloadable from: ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf, May 2003.

[5] A. Arkin et al. Web Service Choreography Interface (WSCI) ver. 1.0. http://www.w3.org/TR/wsci/, August 2002.

[6] D. Booth et al. Web Services Architecture. http://www.w3.org/TR/ws-arch/, February 2004.

[7] E. Christensen et al. Web Service Definition Language (WSDL) ver. 1.1. http://www.w3.org/TR/wsdl/, March 2001.

[8] L. Clement et al. Universal Description Discovery & Integration (UDDI) ver. 3.0. http://uddi.org/pubs/uddi_v3.htm, October 2004.

[9] H.-E. Eriksson et al. *UML 2 Toolkit*. John Wiley and Sons, 2004.

[10] H. Foster et al. Model-based verification of web services compositions. In *Proc. ASE2003*, pages 152–161, Oct., 6-10 2003. Montreal, Canada.

[11] X. Fu, T. Bultan, and J. Su. Analysis of interacting BPEL web services. In *Proc. of WWW2004*, May, 17-22 2004. New York, New York, USA.

[12] M. Gudgin et al. Simple Object Access Protocol (SOAP) ver. 1.2. http://www.w3.org/TR/soap12/, June 2003.

[13] R. Heckel and L. Mariani. Automatic conformance testing of web services. In *Proc. FASE*, Edinburgh, Scotland, Apr., 2-10 2005. to appear.

[14] N. Kavantzas et al. Web Service Choreography Description Language (WS–CDL) ver. 1.0. http://www.w3.org/TR/wsci/, August 2002.

[15] B. Meyer. Applying design by contract. *IEEE Computer*, 25(10):40–51, October 1992.

[16] H. Muccini, A. Bertolino, and P. Inverardi. Using software architecture for code testing. *IEEE Transaction on Software Engineering*, 30(3):160–171, March 2004.

[17] A. Orso, M. J. Harrold, and D. Rosenblum. Component metadata for software engineering tasks. In *Proc. EDO 2000*, LNCS 1999, pages 129–144, 2000.

[18] W. Tsai et al. Verification of web services using an enhanced UDDI server. In *Proc. of WORDS 2003*, pages 131–138, Jan., 15-17 2003. Guadalajara, Mexico.

[19] W. T. Tsai et al. Scenario-based web service testing with distributed agents. *IEICE Transaction on Information and System*, E86-D(10):2130–2144, 2003.