Testing the Effectiveness of Dynamic Binding in Web Services

Anthony Sargeant, David Webster, Karim Djemame and Jie Xu
School of Computing
University of Leeds
LS2 9JT
United Kingdom
{scs5ajs, d.e.webster, k.djemame, j.xu}@leeds.ac.uk

Abstract

In recent years, Service Oriented Architectures (SOA) have risen in use as an architectural style for distributed systems. They have many desirable features such as flexibility, software reuse and cost benefits. In addition to this, SOA enables and indeed encourages the binding of services at runtime in the form of dynamic binding. Here, services are bound to service requests at runtime and the choice of service is determined with minimal user intervention. Presently, Web Services have risen as the de facto implementation of SOA and existing research for the testing of Web Services have assumed the choice of service at design-time. However, dynamic binding of services raises several additional challenges, such as managing complexity in service compositions using dynamic binding and non-deterministic behaviour in service selection. Few research exists that involve dynamic binding but with limitations as they do not consider the dynamic binding system itself. This paper focusses on the importance of the dynamic binding system and proposes a technique that can be used to test dynamic binding systems such that the behaviour of the algorithm can be determined.

1 Introduction

Service Oriented Computing (SOC) represents a move from traditional computer architectures such as mainframe computing, to more distributed architecture styles such as Service Oriented Architectures. Service Oriented Architecture (SOA) offers an architecture where software functionality is encapsulated in discrete services, each of which can be discovered and bound to at either design time, or runtime [1].

Services within SOAs are considered to be software that is used, but not owned and traditionally offer functionality on a per-use basis [2]. Each service can be either a single or *atomic* service, or a composite service, itself made up from one or more atomic/composite services which are aggregated as a workflow. Service consumers are loosely coupled to services via well defined interfaces, and use standard methods for intercommunication. Over recent years, Web Services technology has emerged as the de facto implementation of SOA [3] and is experiencing widespread adoption, along with Web Services Business Process Execution Language (WS-BPEL) — an XML-based workflow composition language — as a popular orchestration language [4].

With the introduction of this new paradigm, comes new challenges. In particular as services are potentially developed independently, consumers can have a variety of functionally-equivalent services to choose from, each having a different Quality of Service. Moreover, traditional software testing techniques have been shown to be inappropriate with respect to SOAs due to the black-box nature of services [2]. One of the promises of SOA is *Dynamic Binding* of services, where service requests are bound to concrete services at runtime [3, 5, 6, 7].

Existing research considers the challenges of testing in SOA through a variety of test frameworks. In each of these frameworks, the focus is on SOA through static binding of services, where requests are bound to services that are chosen at design time. This paper examines these approaches and considers their suitability with respect to implementing SOA through dynamic binding of services.

Current research involving dynamic binding assumes that the binding mechanism itself is a 'black-box' from the perspective of the 'user' and thus concentrates on the output of a system, such as in the case of [7]. Where that work has its limitations, is that the binding algorithm itself could be subject to faults, and the behaviour of the dynamic binding algorithms could be non-deterministic [8]. The black-box approach makes third party testing of the dynamic binding algorithm challenging. This paper seeks to address this problem by proposing a framework whereby the binding algorithm itself is subjected to testing to allow its behaviour

to be deterministic and compared against expected output.

The structure of this paper is as follows: Section 2 discusses the characteristics of SOA both in terms of static and dynamic binding. Section 3 gives a brief overview of software testing and existing approaches to the testing of Web Services. Section 4 discusses the limitations of current work and suggests approaches that can be used to test the behaviour of dynamic binding algorithms in Web Services. Section 5 proposes a prototype implementation of a system that will be a platform for the testing of dynamic binding algorithms in Web Services. Finally we present our conclusions and proposed future work in Section 6.

2 Service Oriented Architecture (SOA)

SOA uses a 'service-centric' approach which enables the composition of applications by binding together discovered services that are located across a network to achieve a particular goal. Services themselves reside in a heterogenous environment and are linked together using standard, message-based protocols and thus the composed service is ultimately platform independent [9]. As a result of these service compositions, one service may be the client of another.

The Web Services model has three clear roles [10]: Service Requestor, Service Provider and Discovery Agency. The Service Requestor (also referred to as a *consumer*) is the requestor of a particular service. In order to find the required service that meets the needs of the consumer, they can consult a Directory Agency (also referred to as a *registry*), such as a Universal Description Discovery and Integration (UDDI) registry, which acts like a 'yellow pages' of services offered by various providers. Finally, services are delivered by Service Providers (referred to as *providers*) and registered with the directory agency so that consumers can find them. [10]

2.1 Dynamic Binding of Web Services

The Web Services Architecture as illustrated in Figure 1, is an example of services that are discovered at design time and which are then statically bound at runtime. This is the form of binding that features in existing research. In this instance, the consumer binds to exactly one provider for all service requests of a given type at runtime. This method of very early binding results in a tight-coupling of consumers to services as re-binding requires the selection of a new service and connection to that new service by the user [11].

SOA enables and promotes the late-binding of service requests to services at runtime in the form of dynamic binding. In contrast to static binding, dynamic binding provides the ability to bind to services dynamically at runtime

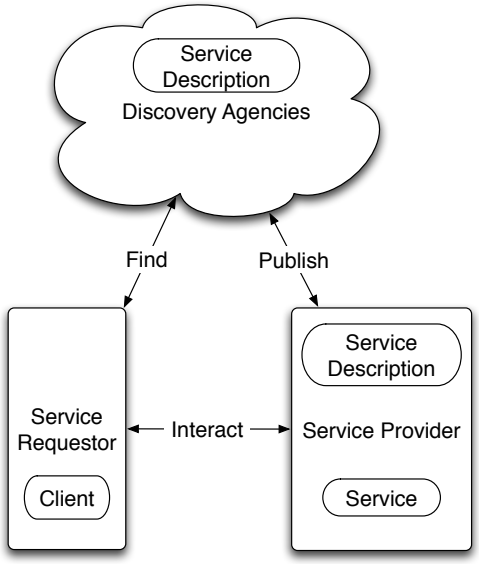

Figure 1. The Web Services Architecture as defined by the W3C in [10].

whereby a request for a service is bound to a suitable concrete service [3, 5]. One of the key aspects of dynamic binding that most literary sources agree on, is the need to have a broker to manage the binding of a service request, to a provider. In this instance, a consumer forms an abstract request, that is to say that they need to use a particular service of a given type, and at runtime, they delegate the decision of which concrete service to use to the broker. The broker, via a discovery mechanism, will discover candidate services that best meet the request and choose the most appropriate service to bind to. The response from the concrete service is then able to be passed back to the consumer via the broker.

The challenge here is because the service provider is not known in advance, the choice of service to bind to must be delegated at runtime. This requires the use of a service broker that can connect service requests to the concrete service implementation. To achieve this, the broker chooses the most appropriate service using an algorithm. An example of a simple algorithm would be: 1. Select candidate services; 2. Rank services in a particular order; 3. Select the top service to bind to.

Choosing the most appropriate service is not as straightforward when we consider multiple services in a workflow such as those defined using BPEL. An example of this is in the work of Mabrouk et al. in [7]. The authors state that decisions made at runtime are subject to time-constraints and that choosing an optimal service composition from a pool of services is NP-hard.

Given that the key aspect to any dynamic binding algo-

rithm is the choice of service to bind to, we must consider the factors that affect the decision. In the literature, factors such as functional and nonfunctional requirements of the user, the scenario context and fluctuations in the Quality of Service (QoS) attributes of the candidate services all contribute to the behaviour of the dynamic binding algorithm [12].

3 Software Testing and Web Services

Software Testing is a relatively new discipline, with a short history beginning in the 1950s. Most sources agree the purpose of software testing is to prove that a piece of software is fit for the purpose to which it is intended [13].

3.1 Testing Strategies

In the present literature, there are three key strategies for the testing of software systems; *Unit Testing* aims to test small programs or, in the case of object-oriented programming, individual objects of a program. Unit tests are in no way a guarantee that when two objects are combined to form a program, the subsequent combination is free of faults. It is for this reason that *Integration Testing* is the next stage of a traditional test plan and aims to find emergent faults that arise from interactions between objects. Finally *System Testing* is described by [14] to be the most complex phase of testing and can be considered to be a higher level of integration testing. However, it also involves many other types of testing such as user-acceptance testing which is beyond the scope of our research.

3.2 Web Services Test Frameworks

Web Services can be distributed across networks and developed by various vendors. Consequently, when vendors implement Web Services, service consumers may not have access to the source code or the execution environment [2]. Furthermore, if the service composition is particularly complex, the number of services and possible combinations of services could be large [7]. It is for these reasons that traditional testing techniques are not applicable as they require access to the system source code [15]. To overcome some of these issues, several authors have suggested new testing techniques and/or frameworks that help consumers to assert a level of trust upon a service or composition of services. Despite this, there are common patterns of test framework: Service-based, Middleware-based, Model-based, and Development-based.

With a service-based framework, the focus is aimed at testing the services themselves. Due to the challenges associated with testing remote services, many research efforts consider Web Services as black-box systems. Test cases are generated by deriving information about the services interface [16].

In middleware-based frameworks, the emphasis shifts away from testing the services themselves, to testing aspects of the middleware such as the workflow composition, or using a broker-architecture as launchpad for testing of services [3, 1].

Model-based testing, for instance the work of [1], aims to test services by using a modelling language to generate test cases that are then applied to the services themselves.

Development-based test frameworks such as proposed by Canfora and Di Penta in [2] look to create test services based around existing services. When there is a need to test the service, the test interface is called.

It is worth noting the increasing use of ontologies to give semantic meaning to services. Work by [3] suggests that applying semantic meaning to services, will aid the automatic selection and testing of services and hence dynamic binding. Despite ongoing research efforts, semantic approaches are still not widely used in practice [17] therefore, implementing a semantic framework for the selection of services is beyond the scope of this research.

4 Dynamic Binding in Web Services — Testing Challenges and Opportunities

As we have seen thus far, there are many approaches to testing a service-oriented system. What is evident from the literature is that current research efforts have not approached the testing of Web Services from a dynamic binding perspective. Indeed, only Karam et al. in [1] considers creating a methodology for testing dynamically composed Web Services as part of their future work. Due to the benefits that a dynamic binding brings however, it is necessary to incorporate the testing of the dynamic binding system in addition to using existing techniques.

Many of the existing test frameworks for Web Services assume static binding of services. That is to say that the services requested are known at design time and as such these testing approaches can be employed. These approaches can still be applied to the dynamic binding of Web Services but with the drawback that none of these approaches consider the dynamic binding system.

As in a static context, service-based testing focuses on the testing of the services themselves. Although testing an individual service does not require the use of a dynamic binding mechanism, the use of dynamic binding can be used to select a service either randomly or based on certain criteria such as nonfunctional requirements, i.e. Service Level Agreement (SLA) or functional requirements, such as a particular operation to be tested.

In middleware-based testing, dynamic binding can be incorporated into a middleware system such as a broker. This

dynamic binding system might be in the form of an Enterprise Service Bus (ESB) or intelligent UDDI registry. By incorporating dynamic binding into the testing strategy, it is possible to test the system's ability to rebind services or the ability to execute the workflow correctly. Alternatively, it can be used to assert a level of confidence on the workflow composition itself. In model-based testing approaches, the test cases are generated based on modelling the behaviour of the service under test, or the composition of services under test. In this instance, dynamic binding can be incorporated by using a dynamic modelling language such as Petri nets which help to model nondeterministic systems [3]. Using a dynamic binding system, the selection of a service to be bound, is determined by certain factors (response time, cost, etc) if a truly random choice is not employed. Even so, it is still possible to assign probabilistic values to services based on those factors.

Development-based frameworks seek to incorporate elements of testing and/or create separate test interfaces for the purposes of testing services. In this instance, dynamic binding would allow the selection of a test interface based on similar criteria as with selecting a normal service.

In each of the above examples the common factor is how to incorporate dynamic binding into these frameworks. In each instance, the way the binding system works changes according to the desired outcome of each test framework. Given the outcome and the number of factors that can affect the behaviour of the binding system, it is clear that the implementation of the system itself, is just as important as determining the validity of the resulting bound services. Yet, previous work in this area does not consider the behaviour of the system beyond the output of the system — the system is treated as a 'black-box' with the assumption that if the output is what is expected, then the system is functioning correctly.

To illustrate this, let us consider the dynamic binding system itself. Consequently the system potentially may be subject to faults like any other system. For example, consider two candidate services that have functionally and semantically equivalent operations. In this instance, the dynamic binding system can choose to use either services. We acknowledge that were one service to change its interface, then there is a risk of an interface mismatch that can lead to an *"interaction fault"* as per the taxonomy by Avizienis et. al in [18]. In this current work, we are not addressing this problem, but is scheduled to be the subject of future work.

5 Prototype Solution

It is the focus of current and future research to assess the dependability of dynamic binding in Web Services as it provides us with a convenient real-world implementation of SOA in which to conduct our experimentation. Our proposed methodology is to use a simulation-based approach in order to implement and assess binding systems. Much like the work of [19], this research will focus on the middleware layer, such as an intelligent broker or ESB-based solution as it is here where the dynamic binding system resides.

5.1 System Model

The system model for our proposed solution, as illustrated in Figure 2, is based on the models by [8] and consists of three entities: A consumer, who is requesting a service; A broker who manages consumer requests and binds those requests to concrete service instances; and service providers such that $2 \leq p \leq n$ where p is a concrete service instance and n is the number of service instances available to the broker.

In this model, the system works as follows: 1. The consumer sends a request for a service to the broker. 2. The broker analyses the request and then finds the most appropriate services from the service repository. 3. The broker ranks the services according to some criteria. This criteria can be specified by the consumer in the form of QoS requirements. 4. The broker selects a service from the list of ranked services and consumes the service on behalf of the consumer. 5. The result is sent back to the consumer. Our model makes the following assumptions: Firstly all services have been discovered by some mechanism - dynamic service discovery is beyond the scope of this research. Secondly there are at least two concrete service instances for every service request in the repository. Where QoS attributes are not advertised, the service selection may looked to historical data via a monitoring mechanism such as used by [11].

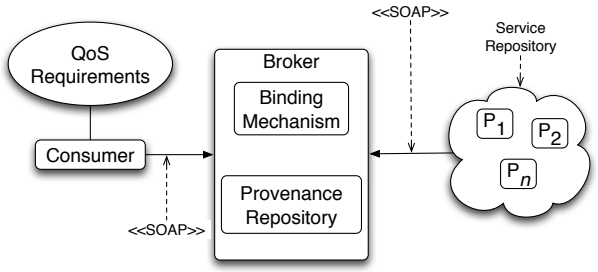

Figure 2. System Model showing one consumer, an intelligent broker and n providers in a service repository.

To test the dynamic binding system, we intend to manipulate the operating environment through the introduction of faults at the middleware level. In order to simulate these faults, a fault-injection mechanism based on the work of Looker [16] will be employed. This mechanism will involve creating a system which will intercept messages and

assess whether or not the message needs to be processed for faults.

In order to force the dynamic binding mechanism to rebind, two possible techniques can be used. The first involves simulating faults with the services such as that employed by Looker [16]. For example, by introducing faults such as service availability faults, we can force the binding mechanism to choose an alternative service. The second approach would be to vary the individual QoS parameters of the SLA such that the binding mechanism has to be either more or less selective when choosing a service to bind to in the presence of faults.

5.2 Fault Model for Dynamic Binding Systems

In order to inject the correct types of faults, Looker in [16] considered the types of fault that can affect Web Services. The high-level fault model included Physical Faults, Software Faults, Resource Management Faults, Communication Faults and Life-cycle faults. However, Looker's work considered binding from a static perspective thus the fault model does not consider dynamic binding. When we consider a dynamic binding system, then we must extend the fault model to include additional classes of faults such as *Timing Faults* as in Figure 3. For example, when using a dynamic binding system, timing is crucial if we are to bind to the best possible service. If an ideal service is selected, but unavailable at the time of binding then a fault will occur and we must rebind to another service. A detailed taxonomy of faults associated with a Dynamic Binding System will be the subject of future work.

5.3 Evaluation Measures and Metrics

In order to evaluate this work, it is important to have a series of measures and metrics by which to judge the behaviour of a dynamic binding system. In the work of [20], measures are defined as being derived from interpretations of one or more metrics with metrics being indicators of system, user, and group performance that can be observed, singly or collectively, while executing scenarios. Consequently, future experimentation for this work will focus on key dependability attributes as defined in [18] for instance, availability, reliability and integrity, etc. as metrics and can be captured as measures of QoS. It should be noted at this point, that the evaluation strategy for services will differ from the evaluation strategy for a dynamic binding system.

In the case of testing services, the focus can be either upon the functional requirements — i.e. the output of the service, or the nonfunctional requirements — i.e. the performance of the service, or both. For example, in [16], the authors focus on two metrics; correctness and timeliness. Correctness is described as verifying whether or not a service returns a correct result. Timeliness is described as verifying whether or not the service returns a result within a certain timeframe.

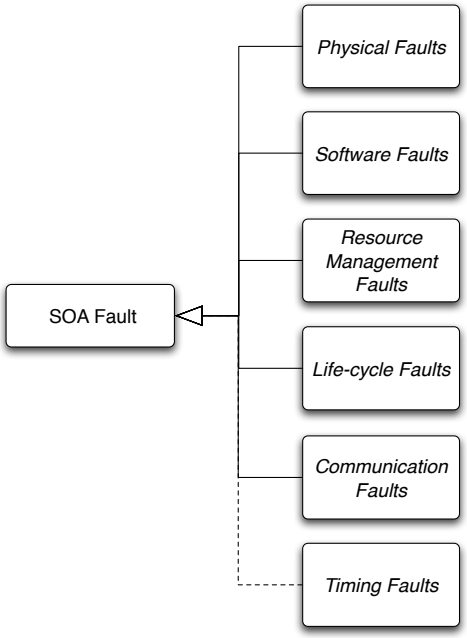

Figure 3. High-level fault model.

When testing dynamic binding systems, it should be noted that although the system involves services, the measures and metrics associated with services will still apply to the dynamic binding system itself, but the context is different. For example, in a dynamic binding system, timeliness refers to whether or not a service was bound within a given timeframe — specified as a nonfunctional requirement. and correctness refers to whether or not the correct type of service was bound at runtime — specified as a functional requirement.

As discussed, the intention of future work is to create a simulated network of services in an environment where faults can be inserted to test the dynamic binding system. By repeated runs of these experiments we will be able to statistically analyse the results to ascertain the behaviour of the dynamic binding system under test.

6 Conclusions

In this paper we have considered the challenges and research opportunities with respect to testing Web Services that take advantage of dynamic binding. We have demonstrated that existing research covers a wide range of testing approaches with respect to Web Services, but none consider the dynamic binding mechanisms that can be employed.

We believe that by manipulating the operating environment through the simulation of faults, we will be able to assess and monitor the behaviour of a dynamic binding system which will result in benefits with respect to the testing of dynamically bound Web Services. A prototype system is presently in development that will enable the testing of dynamic binding systems via a middleware solution that incorporates a fault injection framework.

References

[1] M. Karam, H. Safa, and H. Artail, "An abstract workflow-based framework for testing composed web services," in *Computer Systems and Applications, 2007. AICCSA '07. IEEE/ACS International Conference on*, pp. 901–908, 2007.

[2] G. Canfora and M. Di Penta, "Testing services and service-centric systems: challenges and opportunities," *IT Professional*, vol. 8, no. 2, pp. 10–17, 2006.

[3] A. Bertolino, G. De Angelis, and A. Polini, "A QoS test-bed generator for web services," *Lecture Notes in Computer Science*, vol. 4607, p. 17, 2007.

[4] P. Wohed, W. van der Aalstand Marlon Dumas, and A. H.M. ter Hofstede, *Analysis of Web Services Composition Languages: The Case of BPEL4WS*, pp. 200–215. Springer Berlin, 2003.

[5] X. Gu and K. Nahrstedt, "Dynamic qos-aware multimedia service configuration in ubiquitous computing environments," *Distributed Computing Systems, International Conference on*, vol. 0, p. 311, 2002.

[6] M. Di Penta, R. Esposito, M. L. Villani, R. Codato, M. Colombo, and E. Di Nitto, "Ws binder: a framework to enable dynamic binding of composite web services," in *SOSE '06: Proceedings of the 2006 international workshop on Service-oriented software engineering*, (New York, NY, USA), pp. 74–80, ACM, 2006.

[7] N. B. Mabrouk, S. Beauche, E. Kuznetsova, N. Georgantas, and V. Issarny, "Qos-aware service composition in dynamic service oriented environments," in *Middleware '09: Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, (New York, NY, USA), pp. 1–20, Springer-Verlag New York, Inc., 2009.

[8] N. Looker, *Dependability Analysis of Web Services*. PhD thesis, Durham University, 2006.

[9] G. Hohpe and B. Woolfe, *Enterprise Integration Patterns: Designing, Building and Deploying Messaging Solutions*. Boston: Addison-Wesley, 2004.

[10] The World Wide Web Consortium W3C, "Web Services Architecture." `http://www.w3.org/TR/2002/WD-ws-arch-20021114/`, November 2009.

[11] A. Erradi and P. Maheshwari, "Dynamic binding framework for adaptive web services," in *Proceedings of the 2008 Third International Conference on Internet and Web Applications and Services*, pp. 162–167, IEEE Computer Society, 2008.

[12] L. Baresi, E. Di Nitto, and C. Ghezzi, "Toward open-world software: Issue and challenges," *Computer*, vol. 39, no. 10, pp. 36–43, 2006.

[13] G. J. Myers, *The Art of Software Testing*. New York: John Wiley & Sons, 1979.

[14] J. Abbott, *Software Testing Techniques*. Manchester: NCC Publications, 1986.

[15] N. Looker, M. Munro, and J. Xu, "A comparison of network level fault injection with code insertion," in *Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International*, vol. 1, 2005.

[16] N. Looker, J. Xu, and M. Munro, "Determining the dependability of service-oriented architectures," *International Journal of Simulation and Process Modelling*, vol. 3, no. 1, pp. 88–97, 2007.

[17] L. Cavallaro and E. Di Nitto, "An approach to adapt service requests to actual service interfaces," in *SEAMS '08: Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*, (New York, NY, USA), pp. 129–136, ACM, 2008.

[18] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *Dependable and Secure Computing, IEEE Transactions on*, vol. 1, no. 1, pp. 11–33, 2004.

[19] P. Mayer and D. Lübke, "Towards a bpel unit testing framework," in *TAV-WEB '06: Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications*, (New York, NY, USA), pp. 33–42, ACM, 2006.

[20] D. Russell, N. Looker, and J. Xu, "SOA, Dependability, and Measures and Metrics for Network Enabled Capability," in *IET Forum on Capability Engineering: At Home and Abroad*, 2006.