

Dependability of Dynamic Binding in Service-Oriented Computing

Anthony Sargeant
School of Computing
University of Leeds
Leeds, LS2 9JT, United Kingdom
scs5ajs@leeds.ac.uk

Abstract— Dynamic behavior in Service-Oriented Computing (SOC) allows for flexible selection of services when meeting a service request. By choosing the 'best' service at runtime, we can introduce flexibility and dependability into SOC that is missing from traditional, more 'static' approaches. Present research in this area focuses on dynamic or late binding of web services, either through runtime service discovery and selection, or runtime service composition. However, what is missing from existing work is an understanding of how dynamic binding affects the dependability of SOC. The aim of this paper therefore is to develop a fault model that captures the typical faults that can affect such a system. In order to support this, we propose a unified system model captured from existing work that describes an abstract model of Web Services with incorporates Dynamic Binding.

Keywords—Service-Oriented Computing; Service-Oriented Architectures; SOA; SOC; Web Service; Dynamic Binding; Late Binding; Dependability

I. INTRODUCTION

Service-Oriented Computing (SOC) provides a framework in which applications are built up from services, often distributed across a network [1]. By loosely coupling service consumers to well-defined interfaces, and by using standardized intercommunication methods, we are able to create complex applications through the aggregation of one or more services. This offers us a flexible infrastructure in which the choice of service is agile to the needs of the service consumer, and services need not be bound to at design time [2]. In fact, it is desirable to leverage these characteristics by searching out candidate services that are functionally equivalent and selecting from them, the 'best' service at runtime [1, 3-5].

Runtime binding of services is referred to as *Dynamic Binding* or *Late Binding*. Here, abstract consumer requests are bound to concrete service instances at runtime [6, 7]. Existing work involving dynamic binding considers certain aspects such as dynamic service composition [1, 3], how to best match requests to services [4, 6], dynamic service discovery [7], or dynamic reconfiguring of services [8]. Other works propose frameworks for the implementation of context-aware dynamic binding such as in ubiquitous computing environments [9]. What is clear from these works is that there exist several components that help to provide the necessary behavior for dynamic binding.

In order to ascertain what the 'best' service is, we must look to nonfunctional requirements [10]. This is often done using a separate framework for describing nonfunctional requirements - often referred to as Quality of Service (QoS) requirements [1, 5] - as interface definitions do not describe nonfunctional properties [10]. The final aspect to ascertaining the 'best' service is context. In [9], Vuković et al. discuss dynamic behavior in ubiquitous computing environments. They note that one additional factor is context-awareness as any change in context might require the rebinding of appropriate services to reflect the change.

A common feature of existing dynamic binding research is the introduction of new techniques to enable dynamic binding. Here the focus is on the evaluation of such a system as in the case of [1]. Similarly, in existing dependability research in SOC, the focus is on traditional, more 'static' approaches. Consequently, there is a need to understand the dependability of a Dynamic Binding System (DBS) that is not considered in the literature.

The contribution of this paper is to consider the impact of dynamic binding on the existing fault model for SOC, and extend the existing model so that it incorporates dynamic binding behavior. To support this, we collate the contributions of existing work to identify and unify the components that enable dynamic binding, into a single reference model for dynamic binding for SOC and consider the types of fault that can occur at each component.

The remainder of this paper is laid out as follows: Section II discusses the System Model and its various components that enable Dynamic Binding to take place. Section III concerns the Fault Model that applies to the System Model and Section IV discusses future work into the evaluation of Dynamic Binding Systems.

II. SYSTEM MODEL OF DYNAMIC BINDING IN SOC

The System Model for a DBS consists of the following components: Request Processing, Service Discovery, Service Selection, Interface Mediation and Context Monitoring and is illustrated in Figure 1. We will briefly describe each component in turn. However, it is worth noting that dynamic binding in SOA is on a sliding scale, that is to say that some SOA may contain certain dynamic elements, but not others and so on [5]. In this paper we aim to capture the necessary components to make an 'idealized' Dynamic Binding System.

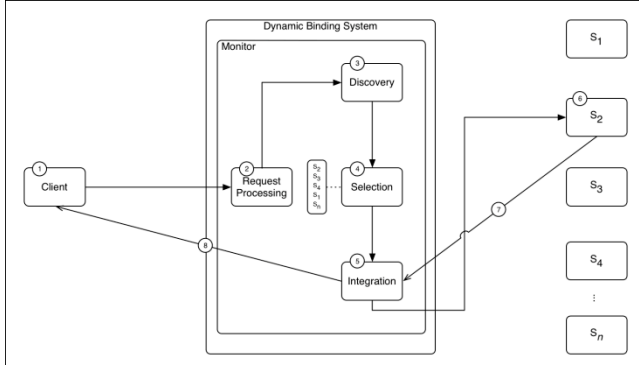


Figure 1. System Model for Dynamic Binding in a Service Oriented Architecture (SOA).

Request Processing: The aim of request processing is to take an abstract request from a consumer and identify that consumer's functional and non-functional requirements. This will enable the discovery mechanism to search for suitable candidate services, and the selection mechanism to determine the 'best' service to bind to [3, 7, 12].

Service Discovery: As the name suggests, the aim of service discovery is to find candidate services that will meet the consumer's *functional* requirements [1, 6]. Most services are published in terms of their interface. In the case of web services, the interface is published as a Web Services Definition Language (WSDL) document. However, WSDL presently does not describe semantic information about the service and attempts to attach semantic information to WSDL is not in widespread use [4].

Service Selection: It is important that not only is a service selected based on its functional requirements, which is the purpose of the discovery mechanism, but to then select the *best* service based on the client's nonfunctional requirements [10]. It is worth noting that the 'best' service in this context is one that meets, or exceeds the client's minimum nonfunctional requirements [1]. To this end, the service selection mechanism aims to select, from a list of candidate services, the best service that meets the consumer's *nonfunctional* requirements.

Service Integration: In the static binding of services, then integration is a simple case of designing a client for a specific service interface. At runtime, assuming there are no changes to the interface by the provider, then the client program will be compatible with the service. However, if we consider heterogeneous interfaces and protocols between services, then it is important that there is some mediation between the potential mismatch of the consumer's abstract request and the interface of the concrete service [4]. To further complicate matters, Nehzad in [11] notes that the real difficulty is identifying not only the mismatches between two service interfaces, but also the protocols that the services use.

Context Monitoring: When a request is sent, it will be in a given context. This is typically a result of the environment in which the original request is sent [9]. Any context awareness needs to be encapsulated as a monitor of

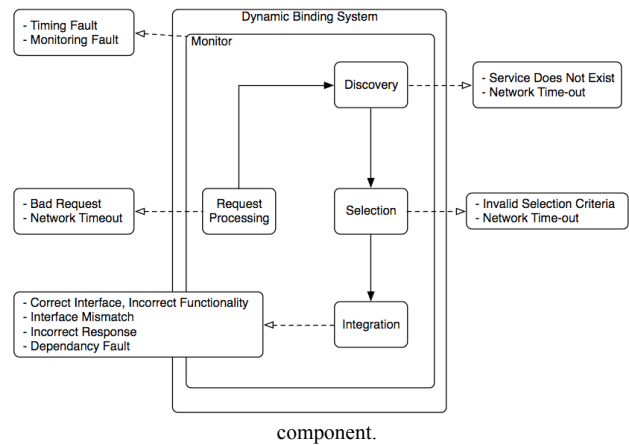
the overall DBS, as context can change at any point before, during and after binding.

III. FAULT MODEL FOR DYNAMIC BINDING

Now that we have described the system model, in order to evaluate a DBS from a functional perspective, the next step is to describe the types of fault that can affect the DBS. In order to do this, we have analyzed each component in turn and identified the faults that might affect that particular component.

Much of the work is derived from previous dependability research by Avizienis et al in [12], Jhumka in [10] and Brüning in [13]. However, that work only concerns itself with the general dependability of Service-Oriented Computing and does not consider faults that might affect a Dynamic Binding System. It is also important to note that this is not an exhaustive model as it is not possible to identify all possible faults that might affect a system.

Figure 2. System Model including high-level faults that affect each



A. Request Processing Fault

Processing a client's request is important as it ascertains the needs of the consumer. However, if the request has been poorly formatted, or the request is outside of the system scope, then this component may fail. In this example the request could be considered to be not complete [14]. Examples of faults that could occur at this stage include:

- **Bad request:** In this instance, the request being sent by the client is rejected. This might be down to a poorly formatted request by the client, or the request is correctly formatted, but is outside of the scope of the system.
- **Network time-out:** Here the request is not received by the system, either because there is no connectivity between the client and the DBS, or the request does not reach the DBS within the specified time.

B. Discovery Faults:

Although how service discovery is implemented is out of scope for this paper, it is still important to understand the faults that could occur. Typical faults include:

- **Service does not exist:** This fault can manifest itself if the search criteria are not recognized. This might be due to the search terms not matching any service available, or that the search terms are simply not recognized i.e. they are outside of the domain of the system. Lastly, the service may exist, but has yet to be published [10].
- **Network time-out:** This fault might occur if there is no connectivity between the DBS and the search facility or the search facility is unable complete the request within the allowed time.

C. Selection Faults:

Selection faults occur when the dynamic binding system selects the wrong service, or there is no suitable candidate service available. Typical faults include:

- **Invalid selection criteria:** Here the selection mechanism fails because the criteria being specified by the client is not recognized by any service. This might be down to the service not using the specified criteria in describing its nonfunctional requirements. Alternatively, the client request has been poorly formatted either by specifying invalid criteria and/or values.
- **No suitable service:** Here the selection mechanism has been unable to find a suitable service from the list of candidate services. This might occur as a result of the client's nonfunctional criteria being too strict such that a service cannot be found that meets these minimum requirements. Alternatively, the service chosen, and any subsequent replacement services, might be unavailable. This would most likely be due to a service being at capacity, or not accessible across the network. Finally, there may be a mismatch in the QoS values such that either the advertised values by the provider is different to that experienced at runtime, or that historical QoS data might be incorrect. In both instances, the QoS requirements would be violated and the system will fail.

D. Integration Faults:

Integration faults occur as a result of one or more services not being interoperable and where the mediation between the interfaces and/or protocols is not possible.

- **Correct interface, incorrect functionality:** Here the service advertises an interface that has the same method signatures, but the service functionality is different.
- **Incompatible interface:** Here the service interface is incompatible with the request.
- **Dependency Fault:** Here a failure occurs because a service's dependency on another service is not satisfied.

E. Context Faults:

Context faults occur when either the current context is not reported correctly, or that the timing of selecting the

'best' service is incorrect. Faults that occur here would include:

- **Timing Fault:** A timing fault occurs as a result of the binding happening too early - i.e. a more suitable service becomes available prior to binding, or too late - i.e. in waiting for a more suitable service, the choice becomes limited as services become unavailable.
- **Monitoring Fault:** A context fault might occur when the request and/or response is no longer appropriate in the current context. This might be because system monitor has failed to identify a change in context, or has reported the context incorrectly.

IV. FUTURE WORK

In order to evaluate a dynamic binding system, our future work will focus on a framework for the testing of a DBS implementation. To achieve this, we enclose the DBS with proxies that manage all messages into, and out from the DBS. These proxies will have the capability to inject faults into the messages similar to the work done by Looker et al. in [15]. By using the fault model to guide the selection of faults, we are able to target a specific component or components that we wish to test. By inducing failures at one or more of these components, we can observe the behavior of the DBS by monitoring the outputs of the system. This 'black-box' approach means that we are not dependant on the implementation details of the DBS, in order to introduce faults and we are able to concentrate more on the behavior of the DBS in the presence of faults. A diagram illustrating our framework is given in figure 3.

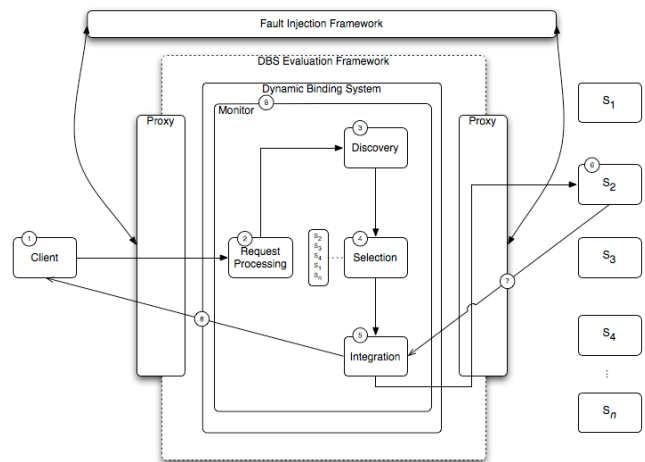


Figure 3. Evaluation Framework Model for testing a Dynamic Binding System (DBS)

V. CONCLUSION

Dynamic Binding in SOC offers consumers a more flexible and agile way of consuming services. In this paper we acknowledge that this flexibility gives us ways of tolerating certain faults present in traditional, more static

approaches. However, this flexibility brings new issues of dependability not considered in the existing work.

To address this, we have proposed a unified system model based on commonalities between different models of dynamic binding in the existing research. Understanding this model allows us to identify the types of fault that can occur at each stage of a Dynamic Binding System. To this end we have proposed a fault model for such a system that will enable the evaluation of future Dynamic Binding Systems.

Finally we propose a framework with which we will implement our vision. This is the subject of continuing work.

ACKNOWLEDGMENT

The author would like to express his gratitude to Prof. Jie Xu and Dr. Karim Djemame who have provided invaluable help and supervision towards this research. This work is funded by an EPSRC grant and a DTA from BAE Systems.

REFERENCES

- [1] N. B. Mabrouk, S. Beauche, E. Kuznetsova, N. Georgantas, and V. e. Issarny, "QoS-aware service composition in dynamic service oriented environments," in *Middleware '09: Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware* Urbana, Illinois: Springer-Verlag New York, Inc., 2009, pp. 1-20.
- [2] R. D. Callaway, M. Devetsikiotis, Y. Viniotis, and A. Rodriguez, "An Autonomic Service Delivery Platform for Service-Oriented Network Environments," *Services Computing, IEEE Transactions on*, vol. 3, pp. 104 -115, april-june 2010.
- [3] 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*, 2008, pp. 162-167.
- [4] 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* Leipzig, Germany: ACM, 2008, pp. 129-136.
- [5] P. Châtel, J. Malenfant, and I. Truck, "QoS-based Late-Binding of Service Invocations in Adaptive Business Processes," in *Web Services (ICWS), 2010 IEEE International Conference on*, 2010, pp. 227 -234.
- [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* Shanghai, China: ACM, 2006, pp. 74-80.
- [7] E. M. Maximilien and M. P. Singh, "A framework and ontology for dynamic web services selection," *IEEE Internet Computing*, vol. 8, pp. 84-93, 2004.
- [8] Z. Zheng and M. R. Lyu, "A Distributed Replication Strategy Evaluation and Selection Framework for Fault Tolerant Web Services," in *Web Services, 2008. ICWS '08. IEEE International Conference on*, 2008, pp. 145 -152.
- [9] M. Vuković, E. Kotsovinos, and P. Robinson, "An architecture for rapid, on-demand service composition," *Service Oriented Computing and Applications*, vol. 1, pp. 197-212, 12 2007.
- [10] A. Jhumka, "Dependability in Service-Oriented Computing," in *Agent-Based Service-Oriented Computing*: Springer London, 2010, pp. 141-160.
- [11] H. R. Motahari Nezhad, B. Benatallah, A. Martens, F. Curbera, and F. Casati, "Semi-automated adaptation of service interactions," in *WWW '07: Proceedings of the 16th international conference on World Wide Web* Banff, Alberta, Canada: ACM, 2007, pp. 993-1002.
- [12] 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, pp. 11-33, 2004.
- [13] S. Brüning, S. Weissleder, and M. Malek, "A Fault Taxonomy for Service-Oriented Architecture," in *High Assurance Systems Engineering Symposium, 2007. HASE '07. 10th IEEE*, 2007, pp. 367-368.
- [14] U. Küster and B. König-Ries, "Supporting Dynamics in Service Descriptions - The Key to Automatic Service Usage," in *Service-Oriented Computing - ICSOC 2007*. vol. 4749, B. Krämer, K.-J. Lin, and P. Narasimhan, Eds.: Springer Berlin / Heidelberg, 2007, pp. 220-232.
- [15] N. Looker, M. Munro, and J. Xu, "WS-FIT: A tool for dependability analysis of web services," in *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*. vol. 2, 2004.