A Performance Modeling "Blending" Approach for Early Life-cycle Risk Mitigation

Paul Brebner
NICTA/ANU
Canberra Research Laboratory
7A London Circuit, Canberra
Paul.Brebner@nicta.com.au

ABSTRACT
In this paper we describe our first experiences of a performance modeling "blending" approach for early life-cycle risk mitigation in a large enterprise integration project. The goal was to use performance modeling to assist with defining the requirements for the system and to identify areas of architecture and technology risk which could be addressed in future phases of the project. We modified our Service Oriented Performance Modeling approach to enable useful models to be constructed by "blending" data from a variety of imprecise and incomplete information sources prior to the existence of concrete requirements or implementations. The approach iterated over two phases to ensure that deficiencies in method and information identified in the first phase were addressed in the second phase. Activities included scenario and workload modeling in phase 1, and software infrastructure, workload and "blended" modeling in phase 2. The resulting models enabled early exploration of critical assumptions and architectural alternatives. The results were enthusiastically received by the client and used by key decision makers and as inputs for future project phases. The "blending" approach to early life-cycle performance modeling raised the profile of architecture performance risk mitigation in the inception phase, so that performance is more likely be a feature of the subsequent development phases.

Categories and Subject Descriptors
C.4 [Performance of Systems]: Modeling Techniques

General Terms
Performance

Keywords
Blended performance modeling, Early life-cycle risk mitigation

1. INTRODUCTION
This paper describes our first experiences with using performance modeling for risk mitigation early in the life-cycle of a large-scale enterprise integration project. Typically performance modeling is used in later stages of a software development lifecycle when there is sufficient concrete information about the requirements and implementation of a system to build and parameterize models with the required predictive accuracy. However, for systems that have undefined requirements and which have no implementation the value of building a performance model appears to be limited. Over the last 2 years we have been involved in risk mitigation for a large scale enterprise integration project. The goal was to use performance modeling to assist with defining the requirements for the system and to identify areas of architecture and technology risk which could be addressed in future phases of the project.

Our Service Oriented Performance Modeling approach [5-8] has been applied on many projects of different types and technologies. It is a method with tool support for performance modeling Service Oriented Architectures directly using SOA concepts. Service Oriented Performance Models consist of workloads, composite and simple services, and servers. Workloads capture the external use of the system and include aspects such as external consumers of services and the associated business processes (workflows consuming services). Composite services describe applications in terms of the externally visible service interfaces and the business processes (workflows) and dependencies on other services that implement them, simple services are atomic services for which no further decomposition is possible or required, and servers are the physical hardware resources available for the processing of service demands on simple services (e.g. servers and networks). See Figure 1.

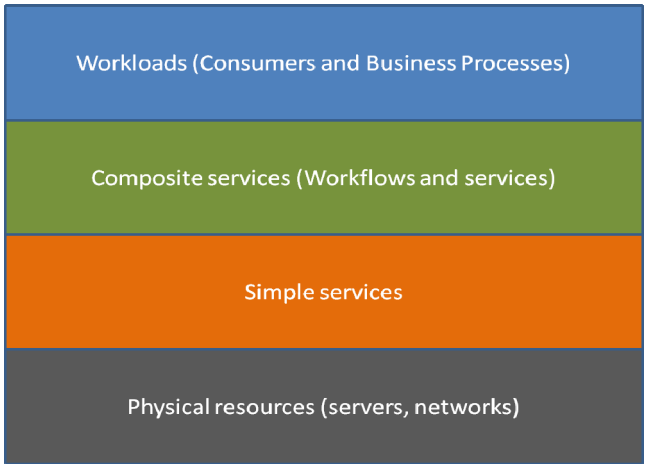

Figure 1 Modelling components

Our methodology for iteratively constructing and calibrating models with these component types from a variety of information and document sources has been adequate for modeling later in the lifecycle [5-8]. Typically we have sufficient concrete information to build a complete model with all component types identified, modeled and parameterized. This produces models of high

ICPE'12, April 22–25, 2012, Boston, Massachusetts, USA.

predictive accuracy but limited benefit due to being built so late in the life-cycle and because they are only models of what actually "is". In other cases some parts of the system exist, and partial models are built of those parts allowing the rest of the system to be modeled in different ways to explore alternatives. For example, the software implementation is modeled "as is", but the unknown workloads and servers are modeled to explore the impact of different load types and sizes on capacity requirements. Or, the known workloads are modeled, but different architectures, designs, and resourcing approaches are modeled for as yet unknown software and hardware.

For this project the challenge was that nothing "concrete" was known at the start. In section 2 we briefly introduce the project context, and in section 3 the details of the phases and modeling activities are covered finishing with the "Blending" approach.

2. PROJECT CONTEXT

The project was an enterprise integration project with the following characteristics. The enterprise was geographically dispersed, with the majority of users located around Australia, but some located overseas. Existing and new applications were to be migrated and integrated using SOA, specifically using Enterprise Service Buses (ESBs). ESBs would be located around Australia and overseas where applications and users were collected. ESBs would be connected together in a federation to allow them to be managed independently, but accessed according to security policies from other ESBs. A variety of common services (e.g. registry, security, XML, etc) would be available on each ESB to allow the construction of new business processes and composite applications which would find, utilize, and extend the functionality of the existing and future applications. The applications would typically be made accessible via the ESBs, but would not be hosted by the ESBs directly. The project was time-bound with delivery milestones spaced regularly across approximately a two year period.

3. MODELING ACTIVITIES

We broke the modeling problem down into the following sub-activities split over two phases. See Figure 2.

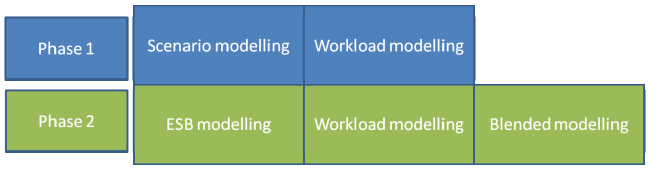

Figure 2 Phases and Modeling Activities

3.1 Scenario Modeling (Business process and composite service modeling)

The first phase of the project was dedicated to understanding how the system would be used. To facilitate this, the project client invested significant effort to develop a set of scenarios to capture the possible uses of the system from a variety of stakeholders. Scenarios included information about the different user roles, the common and application services used, data flows, and the business processes involved in achieving a single and complete end user goal. A single performance critical scenario was selected to focus on, and a partial implementation and initial performance model constructed to provide confidence that the modeling approach was suitable for the task. The implementation and modeling of the scenario was complicated by the fact that a fair amount of interpretation was required in order to build it, that it was complicated and involved many different user roles, 10s of composite and simple services, 100s of business process steps, 5-10 ESB locations, and all of the application services had to be defined and implemented.

Given the project constraints (budget, staff, resources, and finances) only a representative subset of the scenario was implemented. However, we were able to model the entire scenario, and using performance data obtained from the partial implementation we were able to parameterize the complete model. The model was successfully validated against the partial implementation and could be used to predict the performance (e.g. response time) and capacity (e.g. servers and networks) of the complete scenario. We also learnt useful lessons about how best to obtain performance data from the type of ESB stacks and services involved. We also concluded that more information was needed about how the ESBs would be used in practice (the implementation of the scenario was simplistic (the implementation and target architectures were different, and the implementation only used a very small subset of ESB functionality), how ESBs would interact among themselves (e.g. if a target service was not available at a local ESB how would it be accessed remotely), about the workloads (number, location, and user loads).

3.2 Workload Modeling

Workload modeling was done in two phases. In the initial phase (concurrent with 3.1) the data from an organization wide survey of the user community was analyzed. We discovered that there were (a) a large number of communities, (b) that a few communities interacted with a large number of other communities, (c) the majority interacted with only a few other communities, (d) a large number of interactions were within the same user community, (e) no business process information was available (purely high-level data flow information), and (f) the survey was lacking critical information such as locations, load frequency and data volumes.

In the first phase we built a performance model based on the initial workload and ESB implementation information only. This model was designed to show what was assumed about the workloads mapped onto very simple ESB architecture alternatives. We had to make many assumptions to fill in the missing workload information (which just became variables that could be explored with the aid of the model). The main assumptions were that the each user community was located in a single unique location, and that the workload demands (frequency and data sizes) were proportional to the number of data types connected to other communities. The alternative ESB architectures reflected two extreme cases: (1) a single logical ESB backbone (a single logical ESB that all communications went through, but made up of multiple physical ESB resources), and (2) multiple ESBs, one per workload community. In practice, the actual architecture will be something in between (i.e. some user communities will be split over multiple locations; some user communities will be co-located with others at the same location). Being on a partial model of the system and with so many assumptions this type of model was only intended to explore some extreme architectural variants and the impact of changing workload demands, and to reveal gaps in the information available.

In phase 2 we planned to fill in the missing gaps in the second phase with a survey designed to elicit more specific performance

information from a representative subset of the user communities. Part of the overall modeling strategy was to "blend" multiple sources of information together for the final model in phase 2. At the end of phase 1 we had partial models of the business processes/composite services and workloads. The focus of phase 2 was to better understand the ESB implementation and architecture issues and build a more complete model.

We eventually used two approaches to improve our knowledge of the workloads in phase 2. The first was the survey which due to changes in the project and budgets was far more limited than planned and therefore did not produce the desired results. We were able to obtain the type of information we needed (including number and location of user communities, amount of data sent and received within and to and from other user communities, and frequency of data transfers), but the sample size was too small to extrapolate in any sensible way. We therefore relied on a second approach which was to consult with several experts whose knowledge was both broader and deeper across the whole set of user communities. This approach resulted in capturing at least some knowledge about the majority of the user communities and gave us the following critical information: (1) number of user communities and their locations, and therefore the total number of ESB locations as well, (2) basic information about the number and type of workloads each community was involved with including relative frequency, data sizes, and the network of communities and locations involved in each. Several hundred distinct workloads were discovered. Some preprocessing in the way of aggregation of users in the same location was done before this data was used in the blended model in phase 2 (3.4).

3.3 ESB Modeling

The initial implementation of the "scenario" in 3.1 was done by the client on an in-house test-bed on a variety of vendor ESB stacks. For phase 2 we decided that we needed to understand ESB performance and architectural dimensions in a more controlled environment and we set up our own test-bed across a number of servers on an isolated network segment in our own laboratory. In order to do this quickly and to have the maximum control and understanding of the ESB technology and performance results we used an open source high-performance ESB technology. This was deployed and tuned on 4 servers so we could experiment with single ESB node performance, and multiple ESB node performance. One server was used to host the client and load generator, two servers hosted the ESB stack and ESB composite service implementation, and the fourth server hosted the target business service (accessible via the ESBs, but not hosted on them so as to simplify modeling and performance data capture but also to reflect the anticipated architecture of the eventual system).

Further investigation found that there were a number of critical core ESB services that were relevant to the performance of the system. These were: Authentication and Authorization, Encryption, XML Transformation, Registry Lookup, Composite Services & Business Process, and support for multiple transports (e.g. HTTP, JMS).

We designed, implemented and deployed a representative composite application that utilized these ESB services in the way expected, and conducted extensive performance tests on two main architectures. The first architecture was a single node ESB. This example represents the case that a service consumer accesses an ESB and requests a specific service. All of the ESB services are involved in determining if the user can access the service, the service is found to be located on the local ESB, and is then invoked and the response routed back to the consumer. The second architecture involved two ESBs with identical code. See Figure 3.

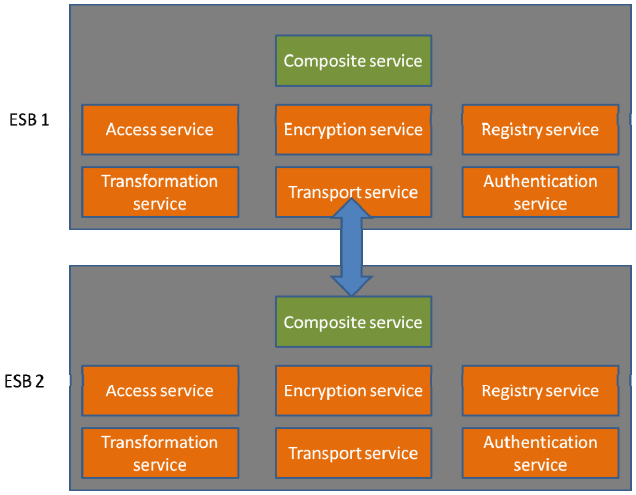

Figure 3 Two ESB node architecture

This example represents the case where the service is not available on the first ESB, so the request is redirected to the second ESB which handles it in the same way that the first one did, eventually finding the service locally and managing the call/response back to the first ESB. This process could be recursive across more than two ESB nodes and represents the case where multiple ESB "domains" have to be navigated in order to use a target service that is located at a specific ESB. We also explored the impact of the message size on the times spent in each ESB service. Not surprisingly the times increase rapidly for some services with increasing data size, specifically encryption and transformation.

Upon completion of the performance testing we had sufficient information to be able to build a detailed model of single and multiple ESB node performance for representation composite applications with different data sizes and ESB services involved. Some of the findings from this work were that: (1) the resource demands of encryption and transformation are substantial and increase with message data size, (2) the resource demands of encryption are asymmetrical (the consumer consumes more resources than the provider).

3.4 Blended Model

Near the end of phase 2 we therefore had good understanding of the types and locations of workloads, the number and location of ESB nodes, and the performance of one representative ESB technology (implementation and architecture). Combined with the work previously done in phase 1 modeling the business process/composite service scenario, and with some assumptions, we had sufficient information to build "blended" models of possible systems.

The first critical concrete piece of information we used from the workload information was the number and location of ESB nodes. This gave us the ability to populate the model with a fixed number of ESB node instances (approximately 20). The location information was also used to determine network connectivity (number and speed of networks between each ESB node) as this was information that was available from another source. The workload information was preprocessed so that model

parameterization was simplified. We ended up with 3 workload types for each ESB node location representing the aggregate workloads of each type for all the user communities at that location. The workload types represented a simple publication of data from one node to other nodes, a more complex business process involving the transfer of data of medium size and the involvement of a reasonable number of services, and a very complicated business process involving the transfer of large amounts of data and the involvement of many services.

The tricky bit of the modeling at this point was determining how to model the dependency on services at different nodes given the originating node. For simplicity we assumed that the probability of any given node being involved was proportional to the connectedness of the nodes from an analysis of the workload data. This resulted in the same probability distribution overall as in the workload data, but may not be true for any given interaction. The service demands on each ESB node were parameterized based on the ESB modeling results from 3.3, and varied depending on the number of ESB nodes involved in calling a target service.

Given that the resulting model was not intended to, and could never in practice, produce a single definitive and validated set of performance metrics for the target system, how was it used in practice? Firstly we used the model to determine which information was likely to be more critical to the performance of the system. We did this by conducting a single variable sensitivity analysis on the parameters related to the workloads, and the parameters related to the ESB resource usage. This clearly showed that the frequency and complexity of one type of workload was critical. Changing it by even a small amount resulted in large changes in performance characteristics of individual ESB nodes and across the whole system (number of CPU cores). It also showed that the ESB security overhead was critical. If either or both of these parameters are significantly higher than the "default" values assumed then the system rapidly becomes unusable and unbuildable, taking too long to process requests and requiring resources orders of magnitude higher than realistically available.

The model was also able to be used to explore some architectural alternatives. These included whether to transfer data via the ESB or use some out-of-band mechanism, and the impact of forcing all services to be available locally on each ESB compared with services having specific (possibly multiple) locations (the most extreme case being centralized only). Again the model was able to show the likely tradeoffs in terms of both performance (response times) and capacity requirements. It was obvious that even with optimistic assumptions about workload demands that transferring data larger than small messages over the ESBs would significantly impact performance and resource requirements. We also discovered that there were huge variations in resource requirements for individual ESB nodes depending on the assumptions made about service locations. Some nodes needed only several CPU cores, while others needed hundreds of cores. In practice this would be exacerbated with dynamic changes in workloads.

Apart from server capacity modeling, we also explored network capacity issues. We discovered that even optimistic assumptions resulted in some of the slower networks completely saturating, and even some of the faster networks were approaching maximum capacity.

4. SUMMARY

In conclusion, this experience has demonstrated that there is potential for earlier and more informative performance modeling of large scale enterprise transformation projects (e.g. migration, integration, etc) for the purpose of reducing risk [3] (rather than purely to give highly accurate performance predictions). The approach evolved from our earlier experiences modeling "concrete" SOA implementations [5], changes in SOAs [6], ESBs [7], and complex real-world modeling [8]. It is different in that the final performance models constructed represent an "abstract" future system (c.f. [1]).

The work has not been validated in the sense that model predictions can be compared to a final working system (which is only in the planning phases, as in [2]). It is a work-in-progress in that we have not yet turned this early life-cycle modeling experience into a complete method or proved it on similar projects. Both the process of modeling and the results have been enthusiastically received by our client and used by key decision makers and as inputs for a number of gateway reviews and future project phases. For this project, the "Blending" approach to early life-cycle performance modeling raised the profile of architecture performance risk mitigation in the inception phase [3], so that performance is more likely be a feature of the subsequent development phases [4].

5. ACKNOWLEDGMENTS

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

6. REFERENCES

[1] Lloyd G. Williams and Connie U. Smith. 2002. PASA^{SM}: a method for the performance assessment of software architectures. In *Proceedings of the 3rd international workshop on Software and performance* (WOSP '02). ACM, New York, NY, USA, 179-189. DOI=10.1145/584369.584397 http://doi.acm.org/10.1145/584369.584397

[2] Woodside, M.; Franks, G.; Petriu, D.C.; The Future of Software Performance Engineering. In *Future of Software Engineering* (FOSE '07). May 2007, 171-187. 10.1109/FOSE.2007.32

[3] Wolfgang Emmerich. 2002. Distributed component technologies and their software engineering implications. In *Proceedings of the 24th International Conference on Software Engineering* (ICSE '02). ACM, New York, NY, USA, 537-546. DOI=10.1145/581339.581405 http://doi.acm.org/10.1145/581339.581405

[4] J.D. Meier, Srinath Vasireddy, Ashish Babbar, and Alex Mackman. Improving .NET Application Performance and Scalability. May 2004. Chapter 2 – Performance Modeling. http://msdn.microsoft.com/en-us/library/ff647767.aspx

[5] Paul Brebner: Performance modeling for service oriented architectures. ICSE Companion 2008: 953-954

[6] Paul Brebner, Liam O'Brien, Jon Gray: Performance modeling evolving Enterprise Service Oriented Architectures. WICSA/ECSA 2009: 71-80

[7] Paul Brebner: Service-Oriented Performance Modeling the MULE Enterprise Service Bus (ESB) Loan Broker Application. EUROMICRO-SEAA 2009: 404-411

[8] Paul Brebner: Real-world performance modeling of enterprise service oriented architectures: delivering business value with complexity and constraints. ICPE 2011: 85-96