Real-world Performance Modelling of Enterprise Service Oriented Architectures: Delivering Business Value with Complexity and Constraints

Paul Brebner
NICTA/ANU
Canberra
Australia

Paul.Brebner@nica.com.au

ABSTRACT

Performance and Scalability Modelling of real-world enterprise systems is challenging due to both the complexity and size of the system being modelled, and constraints imposed by real projects such as the need to provide business value, deadlines, and the accessibility, relevance, quality and quantity of available documentation and performance data. Our hypothesis is that enterprise Service Oriented Architectures (SOAs) are more amenable to performance modelling as services are more granular, visible, and measurable. Since 2007 we have developed, trialled and refined a method with model-driven tool support for directly modelling the performance and scalability of increasingly complex Service Oriented Architectures. This paper reports an illustrative experience modelling a large-scale production SOA Enterprise Service Bus (ESB) upgrade, focussing on lessons learnt related to the complexity and constraints of modelling in the real-world. The key observations are that model construction is a type of *theory formation* and therefore: (1) Models (functioning as theories) can be simple but powerful enough to model large complex SOAs within the boundaries of real project constraints; (2) Model formation can be incremental, starting with a simple model (as simple theories are easier to refute) and refining as required; (3) Building multiple competing models can be a useful approach if information is inadequate or ambiguous, as the rival models can be tested with the aim of discarding incorrect ones; (4) If insufficient information is available to build a single "über" model to answer all the performance questions, it is often possible to build multiple specialised models for different purposes.

Categories and Subject Descriptors

C.4 Performance of Systems

General Terms: Performance

Keywords

ESB SOA Performance and Scalability, Real-world Performance Modelling, Modleling approaches for Enterprises.

ICPE'11, March 14–16, 2011, Karlsruhe, Germany.

1. INTRODUCTION

Performance and Scalability Modelling of real-world enterprise systems is challenging due to both the complexity and size of the system being modelled, and constraints imposed by real projects such as the need to provide business value, deadlines, and the accessibility, relevance, quality and quantity of available documentation and performance data. Our hypothesis is that enterprise Service Oriented Architectures (SOAs) are more amenable to performance modelling as services are more granular, visible, and measurable. Since 2007 we have developed, trialled and refined a method with model-driven tool support for directly modelling the performance and scalability of increasingly complex Service Oriented Architectures. This paper reports a recent illustrative experience modelling a large-scale production SOA Enterprise Service Bus (ESB) upgrade, focussing on lessons learnt related to the complexity and constraints of modelling in the real-world.

A key observation is that model construction is a type of *theory formation* and is related to concepts from the Philosophy of Science including Falsification as a basis for encouraging speculative testable theories [1], and Paradigms as the basis for the concurrent development of competition between multiple incommensurate (conflicting) theories [2]. We propose that Model construction for real projects can be best achieved using the following modeling lifecycle approaches (See Figure 1).

(1) Models (functioning as theories) can be simple but powerful enough to model large complex SOAs within the boundaries of real project constraints;
(2) Model formation can be incremental, starting with a simple model (as simple theories are easier to refute) and refining as required;
(3) Building multiple competing models can be a useful approach if information is inadequate or ambiguous, as the rival models can be tested with the aim of discarding incorrect ones;
(4) If insufficient information is available to build a single "über" model to answer all the performance questions, it is often possible to build multiple specialised models for different purposes.

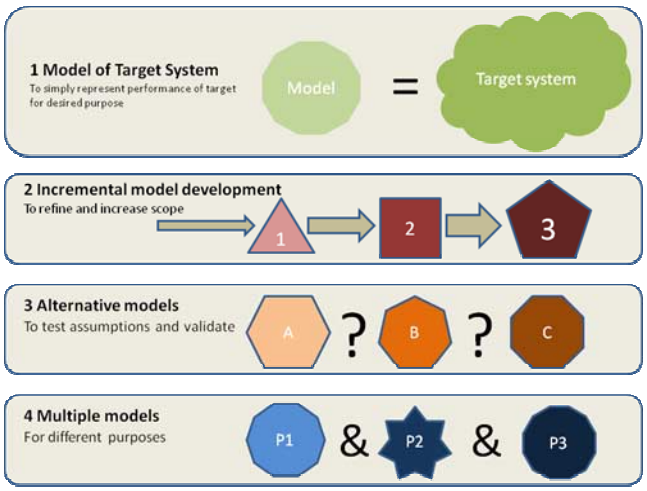

Figure 1 Modelling lifecycle approaches

Performance and scalability modelling of enterprise systems has been previously reported in the literature [3-19]. But papers typically report on modeling enterprise-like systems that are smaller than actual enterprise production systems, and are perfectly understood in controlled laboratory settings; or occasionally full-sized production systems but where modelers have direct access and control over them. In contrast, this paper explores approaches for modeling large complex production enterprise systems to which modelers have only limited and indirect access and no control, and where many other project constraints are imposed on the modeling activity.

Much academic modelling focuses on model accuracy. In practice we have found that modeling in real-world situations is more concerned with the build-ability, utility and usability of models. That is, given the constraints of a real-world project, it is more important to be able to rapidly construct a model that answers the critical performance issues, but only with sufficient accuracy to answer those questions (utility). For example, our models typically predict maximum "theoretical" capacity, or reveal potential bottlenecks or SLAs that may be exceeded, or general architectural flaws preventing adequate scalability. In the case of a prediction of "theoretical" maximum capacity, the actual capacity may be less in practice. However, knowing that the maximum capacity of the system is likely to be significantly less than the required capacity with sufficient time to address this problem is more important than knowing the precise capacity later. Moreover, usability is critical when building a model that represents the target system in a way that makes it easy to build, verify, use and communicate with the client team members. This raises the importance of a methodology and tool support that directly and easily maps to the software and hardware artifacts in the target system, and also enables realistic dynamic visualization and animation of the running system to enable interactive demonstrations and experiments.

Over the last few years we have developed a method with tool support for performance modeling of large complex SOAs [3-8]. We have trialled and refined the method in conjunction with approximately twelve collaborators and clients in both government and non-government (e.g. financial, telecommunications) sectors in Australia. The method and tool support uses a model-driven tool GUI for model development and visualization, and automatically transforms the model into a run-time form to be solved by a discrete event simulation engine to compute performance metrics for each component (including workloads, services, servers) and aggregation of components (including response times, throughput, concurrency, utilization, wait times, service demand, etc). The modeling methodology and approaches taken to model specific types of problems and technology stacks is flexible, and has been evolving to cope with different and more complex and constrained business problems.

The rest of the paper illustrates some of the problems and approaches we have taken to address them by looking in more detail at a single illustrative case study. In section 2 we introduce the modeling project/problem and planned approach, and in subsequent sections we show how the plan worked out in practice, and how the various modeling lifecycle approaches were applied.

2. Modelling Problem and Approach

2.1 Context

The business client was a large international enterprise operating in the Telecommunication sector. When we were contracted to assist with performance modeling they were in the process of a major middleware software and hardware upgrade, as some software versions were no longer supported, and they needed extra capacity to cope with expected growth and peak loads. They wanted to ensure the upgrade resulted in sufficient capacity and headroom for three years.

The software upgrade involved a number of concurrent changes, which made it risky and the results unpredictable. A new workflow engine was introduced for increased flexibility and reduced development costs. The integration architecture was modified to use Web Services and XML data interchange which was better aligned with the Enterprise architecture strategic directions. The ESB software was upgraded and different design patterns introduced ESB features used in the expectation that there would be a performance improvement (as the new pattern was optimized for back-end workflows compared to the previous pattern which was optimized for user interaction only). Transaction and messages were persistent, and a database was used to guarantee message delivery and workflow state persistence. Changes were introduced to the way logging, security, and exception handling was performed. Virtualisation was also introduced, although the middleware vendor did not recommend virtualization of their product stack. Unused software and infrastructure was planned to be decommissioned. Finally, for the hardware, the number and type of CPUs were changed for the ESB middleware and database servers.

Multiple types of users were supported by the system, which was mission critical. User types included customer service agents, dealers, customers, business partners and providers, etc. The upgrade had to be performed with no impact on production (performance, availability, reliability, etc), and still guaranteeing the SLAs of 90% of orders being processed correctly in under 7 minutes.

2.2 Constraints

Compared with much academic modeling, or modeling systems in-house, most of our modeling engagements have been done for external clients which imposes severe constraints on the modeling activities including time constraints, lack of visibility into system, availability (including timeliness, quality and quantity) of

information, ongoing changes in system, size and complexity of system, and business problem to be solved by modelling.

Time constraints

Modelling engagements are typically of fixed duration, and tightly bound to client project milestones. In this case we had 12 weeks to complete the modeling before the production version of the upgraded system went live. There are often time constraints imposed on interactions with the client team (when and for how long they are available). In this case we were engaged by the performance testing team, and we they had limited bandwidth available to interact with us due to their schedule and finite resources. We also had only limited access to other client teams (e.g. development, business) which restricted our access to other types of documentation.

Visibility

Not having direct access to the system or documentation of the system being modeled means that all information has to come via the client. This impact timeliness, quality and quantity of information, so unlike academic modeling, the modeler will never have perfect visibility into the system. In general we find that this means we have access to only a subset of documentation for the system, some documentation either does not exist or is unobtainable, documentation may not be electronic (or in a format that is unusable), and documentation may be incomplete, out of data, and inconsistent. It is also often difficult to get sufficient quality and quantity of measured performance data from running systems, and test data to validate the complete models. In this case, the available documentation was highly focused on the system upgrade, and had been developed by the test team solely for this purpose.

It proved impossible to obtain adequate business process or development documentation, or technical documentation for the middleware. We had to infer the workflow steps from examination of run-time traces of the system. Run-time data was highly detailed and difficult to interpret. Multiple sources of run-time data were available, but more detailed traces were impossible to interpret or correlate with each other. The system was technologically complex, with many heterogeneous components involved with unknown performance and scalability characteristics, including front and back end applications, application services, middleware calls (internal and external), adaptors, database, workflow engine, etc.

Moving Target

The system itself was being run on multiple environments (e.g. development, test, production), and as the system was being upgraded there were ongoing changes in both hardware and software. Performance fixes were also being applied during the modelling period.

Size & Scope

In theory the scope of modeling may be the entire Enterprise system, however in practice modeling is often limited to those parts of the system and level of detail that are essential for answering the performance questions. For this engagement, the modeling was limited in scope to only those transactions that used the ESB. Nevertheless, the potential size and complexity of the system was substantial. There were a large number of transaction types, a large number of steps and interactions with a large number of other systems and services.

Initial size scoping of the system for modeling suggested in the order of 15 (out of a possible 100) transactions, 15 (out of 30) applications, and 20 (out of 200) services, and 10-20 service calls per transaction. The approximate model complexity (in terms of number of model components, including workloads, workflow steps and timing parameters and service calls, services and deployment relationships, and servers, and assuming services are not decomposed further) using our methodology and tool can be estimated with the formula: Complexity = Transactions + (3 * Transactions * Steps) + (2 x Services) + Applications. Assuming the entire system was to be modeled the upper complexity bound of the model is approximately 6530 (100 + (3 x 100 x 20) + (2 x 200) + 30). This is well beyond our capability to manually model in the time available, even if sufficient information was available. However, the upper complexity bound for the scoped model is lower but still non-trivial at 970 (15 + (3 * 15 * 20) + (2 x 20) + 15).

Business Problem

Given the constraints of real world projects, and the complexity of the modeling problem, modeling may appear to be a lost cause. However, the nature of the business problem to be addressed, which is the final constraint, typically makes modeling both tractable and attractive from a business value perspective. Depending on the phase of development, and the type of project (e.g. proof of concept, pilot, development, test, production, evolution, etc) modeling can offer many potential benefits for addressing performance and scalability risk and exploring architecture tradeoffs. For this engagement, modeling was intended to complement, not replace, performance and stress testing. The focus was on the performance and scalability of the enterprise system related to the upgrade of the middleware hardware and software only.

Modelling could enable both more flexible and responsive testing of some scenarios such as different transaction ratios, and testing of higher loads than was physically possible given certain limitations of the test platform (e.g. sizing of systems, capacity of test driver system). Modelling could also be used to determine the load on back-end systems more accurately than in the test environment (as many back-end system were “stubs”, with no real implementation behind them). Modelling was also seen as valuable for assisting with scaling the results given the constraints of some of the test systems, and the use of shared resources for some systems that would have dedicated resources in production. It seemed likely that we would be able to address some of these areas of business and architectural risk through modeling only a smaller subset of target system. The client also wanted to evaluate the suitability of the modeling approach as a strategic tool to increase the use of performance engineering throughput the software lifecycle and across all enterprise systems.

2.3 Approach

Typical of our modelling methodology developed from multiple modelling engagements, the modelling is done incrementally with a number of refinements to increase both the scope and accuracy

of the model. Premature modelling can often result in significant rework. In this engagement we planned three phases. Phase one was a scoping phase to rapidly produce a very high level "straw" model (within a few weeks) to demonstrate the potential uses of a model, and therefore elicit the information required to build a more detailed model which could be used predictively. The second phase was for refinement and initial parameterisation of the model, and the final phase was for increase in scope, final parameterisation and validation, and use of the model to answer performance issues. In practice another phase was introduced in order to explore the ability to build models with different purposes.

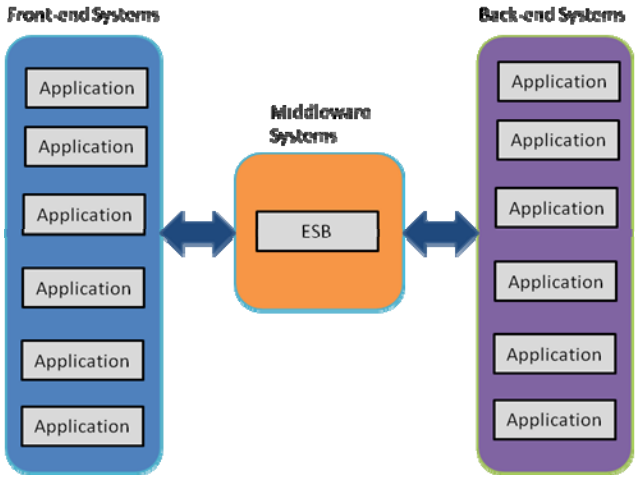

Figure 2 Target Enterprise Architecture

3. Phase 1: Initial Model

The goal of phase 1 was to discover the available information (e.g. architecture, system documentation, performance data, performance requirements, etc), to scope the modelling by selecting a subset of the system to model (based on need, and boundaries of systems, and visibility into systems), and to produce an initial high-level model to communicate the approach, benefits, and information requirements, to the client management and technical teams in terms that were tailored for their environment.

Because we were working in conjunction with the performance test group, a subset of transaction types had already been selected as a focus for testing. This gave us an initial subset of the system to scope the modelling on. There was also good historical data available revealing loads over time, including peak loads and the main contributing transactions.

Our standard modelling method starts with identifying workloads and the services they consume. We model workloads as workflows with steps calling services. Services can be either simple (if there are no known further dependencies, or if the service and all dependencies are deployed to same physical resource) or composite (if further dependencies are known and are on other resources).

During a workshop with the test team we were able to obtain high-level visibility into the broad architecture of the system in terms of input and output applications (and services deployed to them), and the middleware being upgraded (See Figure 2). We determined that further visibility into the external applications would be unlikely (either in terms of implementation details or resourcing), and as the focus of the modelling was the upgrade of the middleware, we decided that further modelling would focus on understanding the transaction workflows and interactions with the middleware, with calls to external applications modelled only as simple services (with no further decomposition into sub-services). Based on previous ESB modelling [4, 5] and knowledge of the network characteristics of this system, we made the added assumptions that LAN and ESB messaging infrastructure could safely be out of scope.

We therefore planned to refine the model through adding more transactions, and more interactions with back-end services (with no more detail), and replacing the simple middleware service with a composite service as more details about the middleware were discovered.

Based on the initial limited information available (which did not include transaction workflow details), we built an initial sample high level model (2 transactions interacting with a front-end application, the middleware, and 3 back-end applications), and presented this model to the management team enabling us to get the go ahead to proceed to Phase 2 (Figure 3). We requested more detailed documentation of transaction workflows and interactions with the middleware for Phase 2.

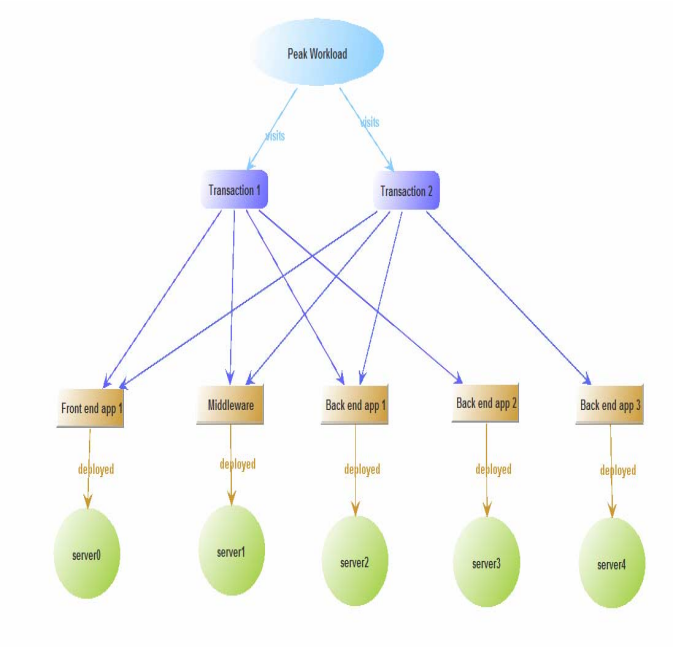

Figure 3 Initial High-level Model

4. Phase 2: Incremental and Alternative Modelling

Phase 2 involves the construction and validation of the first detailed model. This typically involves collection and detailed examination of business, architectural and operational documentation and possible run-time traces/logs (if available), verification with client technical team members, and validation of some parts of the systems as possible. In this phase the initial model is refined into a real model by including more transactions and details. A sample of transaction/workloads is identified as representative or critical (from Use Cases or other business

process documents), workflows are discovered (typically from architectural documents such as UML sequence diagrams), services are discovered (external, internal, and 3rd party, typically from architectural documentation or "service maps" etc), and the a more detailed model is constructed that relates the workloads and services as workflows. Deployment of services to servers, and server resources may also be available and modelled in this phase, and finally initial parameterisation of the model is carried out from available performance data (requirements or SLAs, pilots, development, test, production systems etc). Typically any data that is not available or decisions made about how to model are documented as assumptions which can be verified, and modified as required.

The main extra input required for phase 2 was detailed documentation of the transaction workflows and performance timing data to calibrate the model. We discovered a single example transaction that was both representative of the subset of performance testing transaction types ("typical" in terms of the back-end systems used and the number of calls), but was also performance critical as it was the most common transaction during times of highest peak load on the system.

We managed to obtain audit traces for the example transaction, but little other documentation, making it difficult to interpret the traces. The traces consisted of the name of system/service involved, and timestamps at entry and exit of each system. Each transaction trace had approximately 500 steps (names and timestamps). Initially we obtained only a small number of sample traces as timing data for model parameterisation needs to be obtained from an unloaded system, and it was time consuming for the client team to setup the performance test environment for their exclusive use and run the tests (exacerbated by longer than expected transaction completion times due to a performance issue, see below).

There are typically a number of issues with understanding and using timing data from logs for modelling. These include understanding where the times are being measured from, allocating times to services, and automating the processing of times for use in model parameterisation. In this case we determined that all the times were measured from the perspective of the middleware, thus giving good visibility into systems which directly interacted with the middleware, but no visibility into indirectly connected systems.

With assistance from the client we were able to identify where calls to external systems were made in the traces, and therefore which applications were used, how many service calls there were, and how long the external services were taking. The example transaction used 7 external applications, with 16 service calls (i.e. some applications have multiple services, and some services are called multiple times in the same transaction). We also determined that most service calls were synchronous (request-response), so response times were just the difference between two consecutive time stamps. However, one service call was asynchronous and required more complex processing to obtain the time (i.e. looking for a synchronisation event later in the trace and taking the difference between the time stamp then and the time at which the call was initiated).

Examination of the resulting external service times revealed that they were taking significantly longer than expected (10s of minutes), and that the maximum SLA would be exceeded. This required the client and vendor technical teams to determine the cause of the underlying problem and remediate it before we could proceed further with measurement and modelling.

Given that the focus of the modelling was on the middleware upgrade performance and capacity it was critical to allocate the correct times to the middleware. Close inspection of the trace with the client team enabled us to determine that a number of other systems were closely involved with the ESB, including a database, adaptors, and workflow engine (Figure 4). The problem was then to allocate times to either the middleware or one of these other related systems (which we correctly assumed were independently hosted).

In attempting to understand the traces it was vital to determine what the different steps were that were being logged, and why there were so many steps (500) for only 16 external service calls. It was apparent that many of the steps logged were for low level ESB components and actions, including transaction boundaries, orchestration events, events related to external service calls (e.g. getting and transforming data, preparing calls, doing calls, waiting for or obtaining the response, processing returned data, and finalising call). In fact we discovered 66 calls in the transaction, which were made up of multiple trace steps. Only 16 of these calls were external, and the remainder were to "related" middleware services. The high-level view of the base model is shown in Figure 5, and Figure 6 shows an expansion of the service calls. The complexity of the base model was 79 (1 + (3 * 1 * 18) + (2 x 8) + 8).

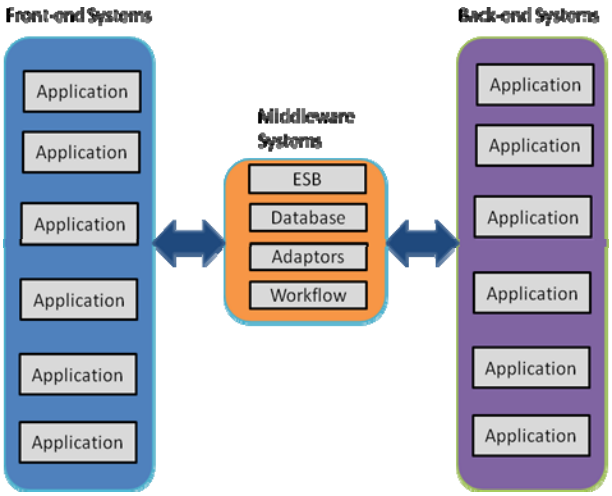

Figure 4 Target Architecture Middleware Systems

More detailed traces were available from some of these related systems, but given the lack of other supporting documentation (e.g. business process documentation), or sufficient direct access to vendor technical experts, and difficulties relating the timing information collected from multiple systems (e.g. attempting to correlate events from the same transaction, and synchronise clocks), the best solution was to develop *alternative* competing models based on different assumptions about how to allocate times to systems. The alternative models would then be tested to determine which could be rejected, hopefully leaving us with one correct model. Figure 7 shows the basic model assuming that all the middleware time is spent in the ESB.

We therefore built 3 models as follows. Model 2a assumed that *all* times were spent in the middleware *except* times that were "obviously" spent in other systems, specifically external application services, and the middleware database (Figure 8).

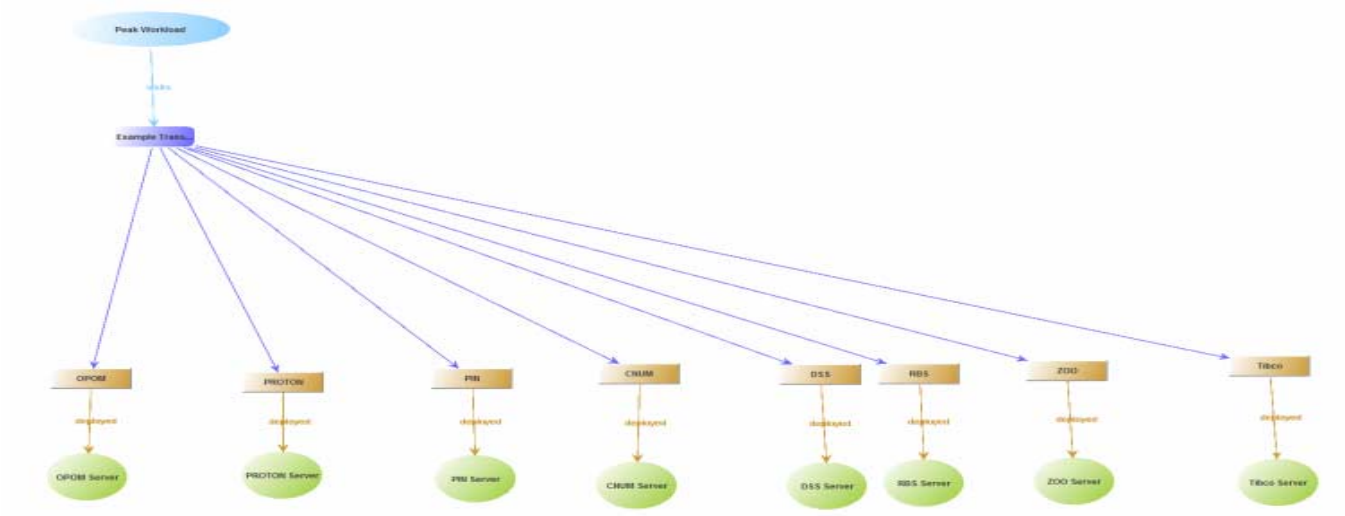

Figure 5 Base Model

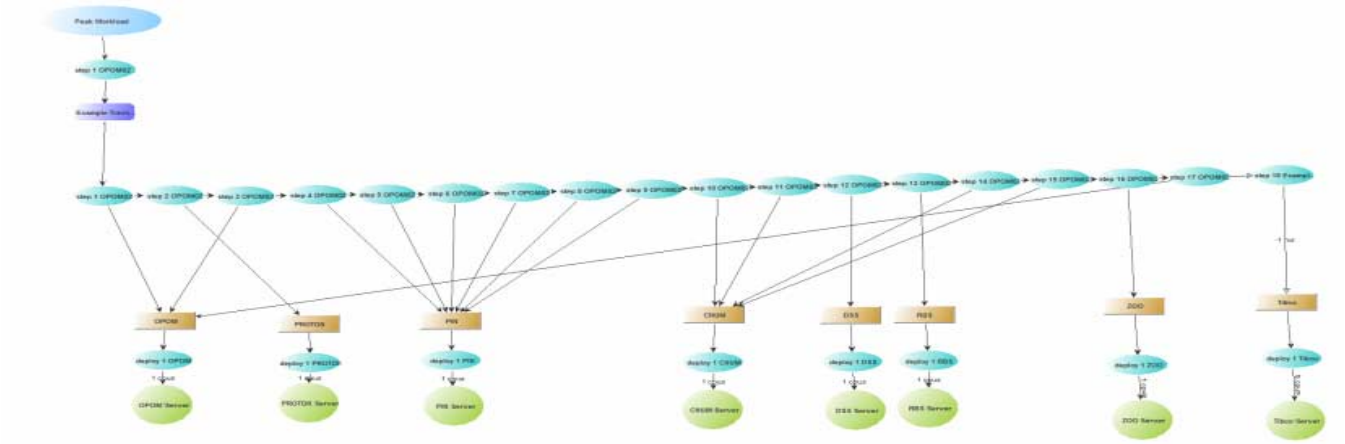

Figure 6 Base Model Detail

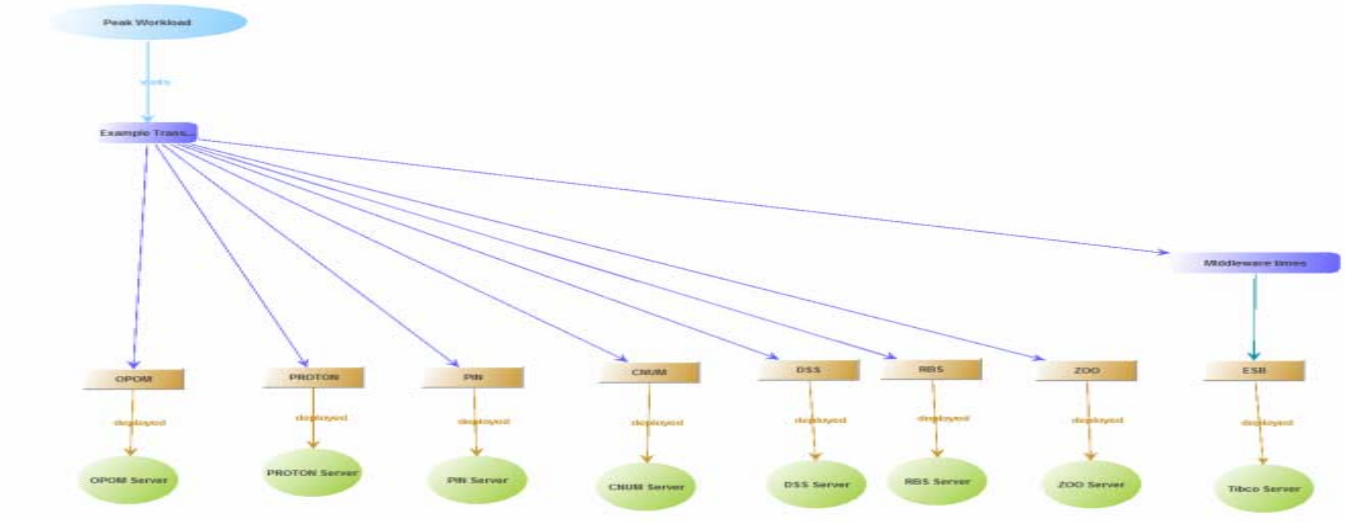

Figure 7 Base ESB Modelling

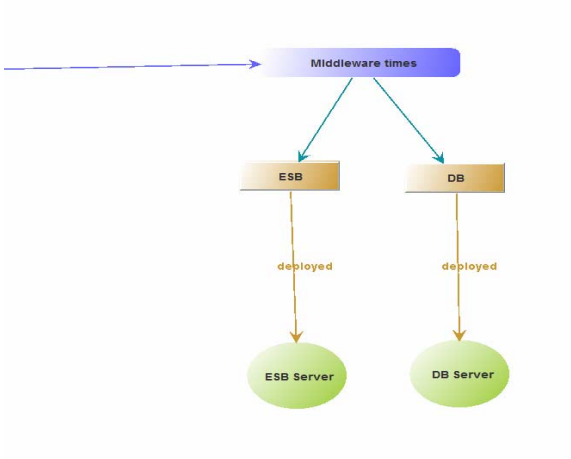

Figure 8 ESB Detail 2A "Pessimistic"

We viewed this as the most conservative (as it was an interpretation of the traces closest to that suggested by the client team) and pessimistic model (as it would predict the least capacity). Model 2b attempted to allocate more time to other systems based on inference (Figure 9). For example, some times were allocated to other systems for transactions, adaptor call outs, and the workflow engine. We viewed this as a reasonably likely model, as it was based on more intuitive rather than purely mechanical interpretation of the traces. Finally, Model 2c (Figure 10) was constructed with the assumption that the *only* times spent in the middleware were directly related to orchestration (steps related to case/sub-case start and termination), and the rest of the times could be allocated to an aggregate of other "related" systems (i.e. an aggregation of all the systems in model 2b and any others undiscovered). This assumption was justified by the observation that the main activity of the ESB was orchestration, processing data for use in calls or obtained from calls, and calling services. This model was the most optimistic in terms of capacity prediction, but also the most risky as it was based on the loosest interpretation of the traces.

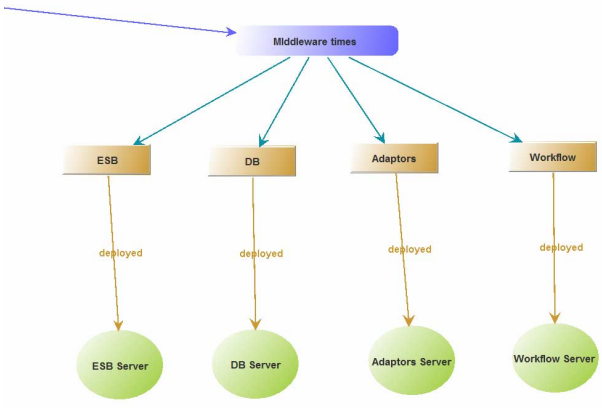

Figure 9 ESB Detail 2B "Likely"

Critically, model 2c also indicated a possible unexplored architectural scalability risk as in order for the ESB to achieve this capacity a larger portion of middleware time is assumed to be spent in the other "related systems", which need to be sufficiently well resourced to ensure they aren't a bottleneck.

Initial parameterisation

Having decided on the alternative models and the alternative allocations of times to systems, we then turned out attention to the quantity and quality of performance timing data. We received 10 sample transactions once the initial performance issues had been rectified, but only two of them were "clean" runs. The other eight had problems with insufficient precision (i.e. second precision compared with millisecond precision for each timestamp) or were duplicated, incomplete, or contained exceptions. We therefore requested at least another 8 samples. The number of samples required to parameterise a model with the desired accuracy is difficult to determine in advance.

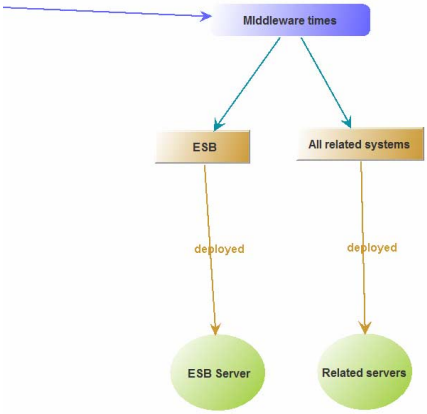

Figure 10 ESB Detail 2C "Optimistic"

From past experience, 10's to 100's of samples are needed to ensure sufficient confidence in the model predictions (and to compute the min/max and confidence intervals etc). Two samples did not give us sufficient confidence in the variability of the data to be able to predict how many samples were needed, the range of values, the confidence intervals or error bounds. However, in order to illustrate to the client the value of more data we produced initial indicative predictions of the capacity of the middleware for each of the alternative models in terms of minimum ("Worst") and maximum ("Best") values (with no error bounds). Results (Figure 11) are reported in Transactions Per Hour (TPH) *per CPU*, as the total available CPUs was unknown. Predictions across the models are highly variable, ranging from a low of 400TPH to a high of 2700TPH (approximately 700% difference), and the variation within models increases from 150% in 2a to over 300% in 2c.

One of the goals of the ESB upgrade was to increase the capacity to cope with forecast and forecast+30% loads (the load forecasted to be reached in 3 years). We therefore increased the number of CPUs for the ESB server until at least one of the model predictions reached the forecast+30 capacity. Figure 12 shows the results with 5 CPUs, compared with the target capacities (Current Production with 3 CPUs, Forecast, and Forecast+30%). Even given the poor data quality model variant 2a is unlikely to scale to the peak demand, but 2c may if the actual values are nearer to the best case values.

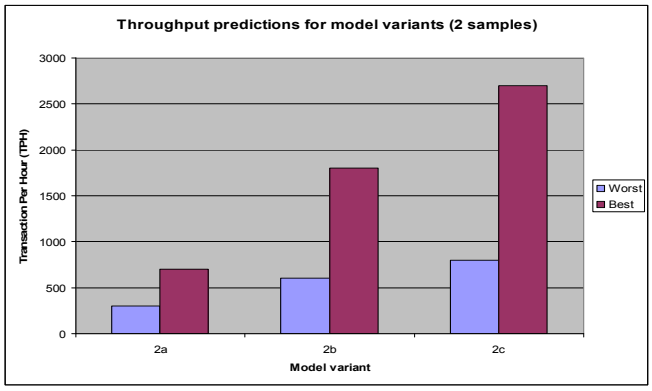

Figure 11 Throughput predictions, 2 samples

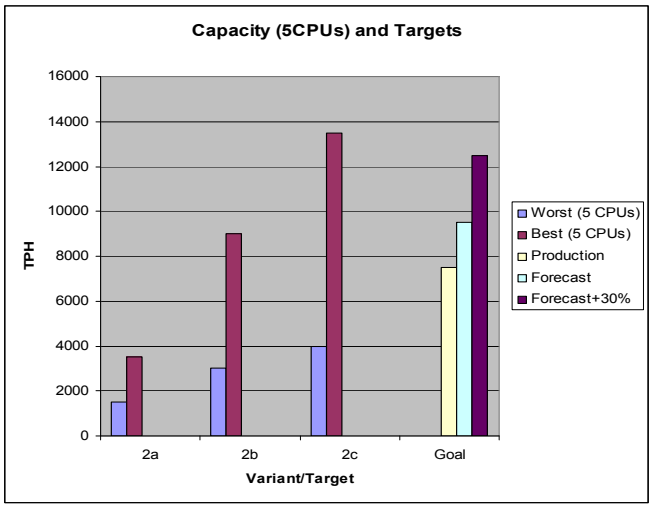

Figure 12 Throughput Predictions 5 CPUs, and Targets

5. Phase 3: Final Paramaterisation, Validation, Scaling

We received eight more trace samples for the example transaction, and initial performance test results for the next phase. However, it was evident that the eight new samples were obtained from a newer configuration of the upgraded system compared with the previous samples, so we were unable to use the previous two clean samples in conjunction with the new ones.

Using only the eight new samples to parameterise the model gave a relative sampling error of 10% and therefore surprisingly good confidence in the model accuracy. Parameterising each of the model variants with the updated and more extensive performance data resulted in the following predictions (Figure 13), which are in terms of median throughput (for 5 CPUs on the ESB server) and include upper and lower 95% confidence limits. With more samples the model predictions are less variable and more conservative compared with only 2 samples. The most optimistic model variant (2c) now predicted a throughput of between 6,000 and 7,5000 TPH which is comparable to current production (3 CPU server), but less than the forecast goals. In order to achieve the forecast+30% target at least 9 CPUs is predicted to be required.

Given the constraints on availability of the test system and test team we were unable to extend the model by including more transaction types as we had planned for this phase.

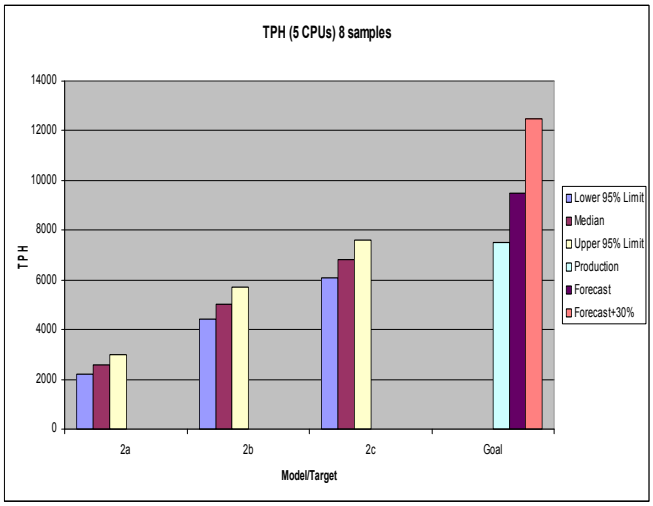

Figure 13 Throughput Predictions, 5 CPUS, 8 samples

5.1 Model validation

There are two main approaches that can be taken to ensure that a model is likely to produce correct results. The first is to verify the model by enumerating all assumptions, and then testing them against available documentation and expert knowledge. In the absence of further documentation or access to middleware vendor technical experts, we were unable to directly verify which (if any) of the model variants were most likely to be correct. The second approach is to validate the model by comparing model predictions to available performance test results. By this phase of the project some preliminary performance test results had become available using a test subset of transactions, for 4, 5 and 6 CPUs. First, the results confirmed an assumption that scalability was linear with increasing number of CPUs. Second, the measured test results for 5 CPUs (7,000 TPH) were comparable with the predictions of only one of the alternative models: The optimistic model variant (2c). See Figure 14. We therefore concluded that the other models (2a and 2b) had been refuted and model 2c was the only remaining candidate model for further use.

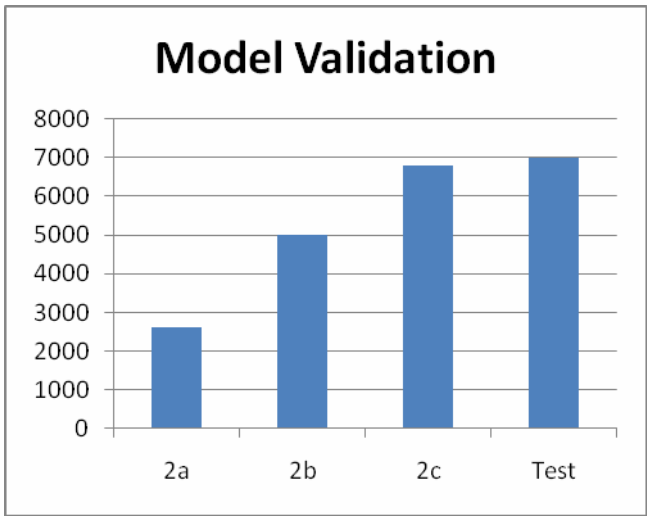

Figure 14 Predictions compared with Test Results

5.2 Model and test results scaling

A fundamental problem is that both the model predictions and test results are lower than the target goals. The upgrade does not appear to have resulted in any substantive improvement in capacity, and the scalability may actually be worse as the production throughput was achieved with 3 CPUs compared with 5 CPUs for the upgraded results. The current model (2c) was highly reliant on the representativeness of a single transaction. We had planned to add at least another 2 transactions to the model but were unable to obtain workflow documentation or timing data for any more. More transactions would have made the model more robust and more likely to have a representative transaction mix. An alternative solution to adding more transactions was to further investigate the representativeness of the sample transaction.

Upon closer analysis the client determined that the subset of transactions used for testing was in fact *more demanding* than the production transaction mix. For testing, the average number of calls to external services per transaction was 22, compared with the 13 for production. Assuming that the ESB & associated middleware times are simply linearly proportional to the number of calls per transaction they had then scaled the results by a factor of 22/13 = 1.69, giving a revised maximum capacity of 12,000 TPH for 5 CPUs, which is comparable to the forecast+30% target.

We therefore also modified the model using similar assumptions (essentially by producing a different special purpose model, 3a, which explicitly modelled the constant ESB overhead per transaction, and then a variable number of external service calls and ESB overheads per call) for 13 calls per transaction (reduced from 16). The revised model (3a) gave an improved predicted capacity approximately 23% higher, between 7,500 and 9,200 TPH, although still approximately 4,000 TPH or 30% less than target. See Figure 15.

5.3 Revised scaling

The scaling used by the client assumed that the middleware time could simply be scaled in proportion to the number of service calls per transaction. However, re-modelling the sample transaction for 13 calls per transaction and examining the traces again revealed that there is a constant ESB/middleware time associated with the transaction as a whole, independent of the actual number of service calls. This suggests that the initial scaling approach may give results that are too optimistic. We estimated this constant overhead for both the test transaction mix and the model production transaction, which were slightly different due to a dependency on the ratio of the external to internal service calls. The revised test capacity was therefore reduced to 10,000 TPH, but the revised model (3b) prediction was only slightly lower at 8,000 TPH (both for 5 CPUs). Based on this analysis the client reconsidered their scaling approach and also planned to redo performance testing with transaction mixes closer to production ratios. See Figure 15.

This model also had the flexibility to easily explore the impact of different transaction mixes (in terms of the average number of external calls per transaction), or the relative benefits of optimising the constant or per call ESB overheads.

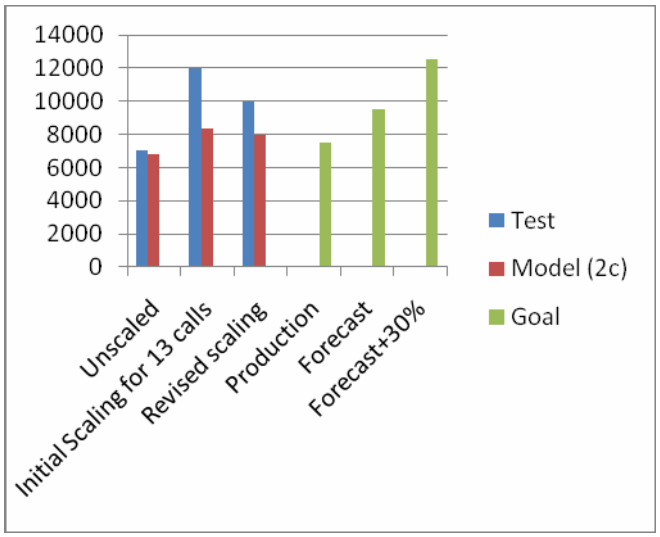

Figure 15 Model and Test Results for Production Transactions

5.4 Observations

The model results suggest that the upgraded system with 5 CPUs may not be able to meet the forecast targets. Moreover, the revised model suggested that the performance test results scaling may be too optimistic, reinforcing the view that the more resources may be required to meet the targets. The revised model predicts that 8 CPUs are needed to meet the Forecast+30% target.

However, we also noted that there was still some uncertainty with the interpretation of trace times. Even if the most optimistic assumptions are correct (2c), it is possible that time measurement isn't as accurate as the precision implies. For example, the measured times may include wait times due to queuing or asynchronous messaging protocols, logging overheads, and times assumed to be spent in one system only may in practice need to be split between multiple systems. Further investigations and experiments with the ESB middleware and monitoring infrastructure, and discussions with the vendor technical experts, would hopefully have resolved some of these ambiguities.

It is also possible that the measured times needed to be scaled for the modelling in order to take into account variable speed processors. If variable speed processors are used in the performance test environment, then the processors may not be running at full speed for the example trace times, but will be running at full speed for the performance test results. For example, the CPUs used in the production system appeared to have dynamic power saving which enables them to run a variable clock speed of between 3.5 (low utilisation) and 5 GHz (high utilisation) depending on utilisation. Thus, assuming trace times were obtained at the slowest speed of 3.5GHz then the revised scaled model (3c) predicts a significantly higher capacity of 11,441 TPH at the top speed of 5GHz (with 5 CPUs), which is close to the Target+30% goal. See Figure 16.

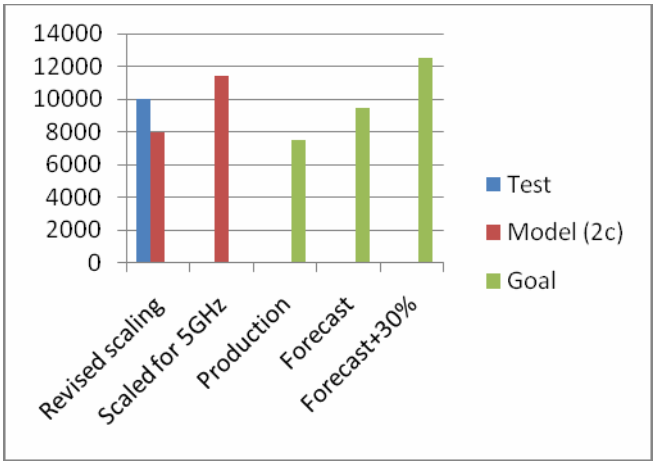

Figure 16 Model Scaled for variable speed CPUs

6. Phase 4: Multiple Models

The model was developed to explore a number of performance issues including the capacity of the upgraded ESB middleware, the customer experience (in terms of end-to-end response times for different numbers and types of users, different transaction mixes, and workload distributions), and the impact on the external back-end systems (many of which are legacy systems and have very limited capacity and scalability). Model building in phases 1 to 3 focused on the first question (the capacity of the upgraded ESB middleware). Due to the limited documentation and sample transaction data, it turned out to be impractical to construct a single model which answered all three issues at the same time. However, it was feasible to construct model variations with increased scope for the second and third issues.

The second model (4a) took into account a bigger subset of transaction types (in terms of the number of service calls per transaction, based on other documentation available) but was parameterised only with response times from the sample transaction, and did not attempt to model external service resourcing – i.e. resources were assumed to be infinite for back-end services). Results for customer experience modelling were good, with 100% of end-to-end times under 5 minutes (the SLA was 7 minutes) and median times under 3 minutes. However, as this prediction depends on the capacity of the external services, which was the third issue, we also built a variant model to address this.

For model 4b we increased the scope to include all the calls to the external services for all transaction types. However, we couldn't do this by measuring and modelling every single transaction type given the constraints (time, documentation, measurements). Instead we developed a probabilistic model of the number and distribution of calls to external services per transaction. The model was not parameterised with response times or resource constraints, and had the sole purpose of predicting the demand on external services (in terms of throughput, concurrency, or service demand) for comparison against documented service SLAs. The result was that several services appeared to be at risk of overload for the forecast load and further testing, mitigation and modelling strategies would need to be applied (e.g. increased use of asynchronous calls to those services and throttling the service capacity, modelling of impact on customer perceived response times, and extra resourcing of services or resource constrained dependent systems where possible).

7. Conclusions

In this paper we have illustrated a variety of modeling lifecyle approaches that can be used to mitigate some of the problems encountered in performance modeling real-world enterprise systems given system complexity and project constraints: modelling only a subset of the scope and detail of the target system, incremental modelling, modelling alternatives, and multiple models.

This project also illustrated one of the reoccurring problems we encounter modelling systems late in their development lifecycle, which visibility into the system (documentation) and the quality and quantity of performance data available. In many cases we can successfully work around these constrains. In other earlier lifecycle modelling projects the lack of data can actually be used as a feature of modelling however.

For example, we have used modelling to assist a government tender process to refine the performance and scalability requirements for the tender process, and also as a gateway process to rank the tender responses in terms of their ability to understand and supply available system information, performance and scalability characteristics, and indicative performance data in their tender response. In this example, we were able to easily determine which systems existed and had been used in production, and which had better performance or scalability characteristics, and would be more likely to meet the performance and scalability requirements, and which probably only existed on paper and would be more risky options.

Finally, for systems in very early requirements and architecting stages, modelling can be used to assist with developing the performance and scalability requirements, and characteristics of different architectures and technologies. This can be achieved using a "blending" modelling approach that utilizes information from a variety of sources to build and parameterise the models (including test-beds, vendor input, benchmark results, stakeholders' requirements, knowledge of legacy systems, etc). These predictions then can be used to further refine options and test solutions during subsequent architectural proof of concept phases. We plan to publish more on this type of modelling project in the future.

8. 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.

9. REFERENCES

[1] Karl Popper, *Conjectures and Refutations*, London: Routledge and Keagan Paul, 1963

[2] T. S. Kuhn, *The Structure of Scientific Revolutions*, 1st. ed., Chicago: Univ. of Chicago Pr., 1962

[3] Brebner, P. C. 2008. Performance modeling for service oriented architectures. In *Companion of the 30th international Conference on Software Engineering* (Leipzig, Germany, May 10 - 18, 2008). ICSE Companion '08. ACM, New York, NY, 953-954. DOI= http://doi.acm.org/10.1145/1370175.1370204

[4] Paul Brebner, Liam O'Brien, Jon Gray. Performance Modeling for e-Government Service Oriented Architectures (SOAs). ASWEC Conference Proceedings (Perth, March, 2008), 130-138.

[5] Brebner, P. 2009. Service-Oriented Performance Modeling the MULE Enterprise Service Bus (ESB) Loan Broker Application. In *Proceedings of the 2009 35th Euromicro Conference on Software Engineering and Advanced Applications* (August 27 - 29, 2009). SEAA. IEEE Computer Society, Washington, DC, 404-411. DOI= http://dx.doi.org/10.1109/SEAA.2009.57

[6] Brebner, P., O'Brien, L, Gray, J., Performance modeling power consumption and carbon emissions for Server Virtualization of Service Oriented Architectures (SOAs). Enterprise Distributed Object Computing Conference Workshops, 2009. EDOCW 2009. 13th. 1-4 September 2009. 92-99. DOI=10.1109/EDOCW.2009.5332010

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

[8] Paul Brebner, Anna Liu. Modeling Cloud Cost and Performance. Proceedings of Cloud Computing and Virtualization Conference (CCV 2010), Singapore, 2010.

[9] A multi-level simulation model of MVS/JES2 Batch Workloads, Charles Hackett, Guardian Life Insurance Company, CMG Annual Conference, 1988.

[10] Incremental Performance Analysis of Client-Server Systems, Thomas Bell, Anne Falk, CMG Annual Conference, 1995.

[11] Using Analytical Modelling to Ensure Client/Server Application Performance, Gene Leganza, CMG Annual Conference 1996.

[12] Singleton, Paul. Performance Modelling — What, Why, When and How. BT Technology Journal, Volume 20, Number 3, 133-143, Springer Netherlands, 1358-3948 DOI: 10.1023/A:1020860029447

[13] J. Skene and W. Emmerich. Model Driven Performance Analysis of Enterprise Information Systems. In Proc. of International Workshop on Test and Analysis of Component Based Systems (ETAPS/TACoS), Warsaw, Poland. Electronic Notes in Theoretical Computer Science (ENTCS) No. 82(6). 2003. Elsevier Science B. V. April 2003

[14] A Comprehensive Toolset for Workload Characterization, Performance Modeling and On-line Control. Li Zhang, Zhen Liu, Anton Riabov, Monty Schulman, Cathy Xia and Fan Zhang. In Performance TOOLS Conference 2003.

[15] Guinness, D. M. and Murphy, L. 2005. A simulation model of a multi-server EJB system. In *Proceedings of the 1st international Workshop on Advances in Model-Based Testing* (St. Louis, Missouri, May 15 - 21, 2005). A-MOST '05. ACM, New York, NY, 1-7. DOI= http://doi.acm.org/10.1145/1083274.1083278

[16] Vittorio Cortellessa, Pierluigi Pierini, Daniele Rossi, "Integrating Software Models and Platform Models for Performance Analysis," IEEE Transactions on Software Engineering, vol. 33, no. 6, pp. 385-401, June 2007, DOI=10.1109/TSE.2007.1014

[17] Marco, Bernardo, Jane, Hillston, Connie Smith. Introduction to Software Performance Engineering: Origins and Outstanding Problems. Formal Methods for Performance Evaluation, Lecture Notes in Computer Science, 2007, Volume 4486/2007, 395-428, DOI=10.1007/978-3-540-72522-0_10

[18] Dubey, A., Mehrotra, R., Abdelwahed, S., and Tantawi, A. 2009. Performance modeling of distributed multi-tier enterprise systems. *SIGMETRICS Perform. Eval. Rev.* 37, 2 (Oct. 2009), 9-11. DOI= http://doi.acm.org/10.1145/1639562.1639566

[19] Cherkasova, L., Ozonat, K., Mi, N., Symons, J., and Smirni, E. 2009. Automated anomaly detection and performance modeling of enterprise applications. *ACM Trans. Comput. Syst.* 27, 3 (Nov. 2009), 1-32. DOI= http://doi.acm.org/10.1145/1629087.1629089

[20] Wells Fargo Performance Modeling - Techniques for Integrating into Development Life-Cycle Processes, Todd Nichols, White paper at HyPerformix User Conference. http://www.anser-e.com/performance/modeling/WellsFargoModeling.htm

[21] Performance Modeling for Web based J2EE and .NET Applications, Shankar Kambhampaty, Venkata Modali, World Academy of Science, Engineering and Technology 8, 2005. http://www.waset.org/journals/waset/v8/v8-63.pdf

[22] Rolia, J., Casale, G., Krishnamurthy, D., Dawson, S., and Kraft, S. 2009. Predictive modelling of SAP ERP applications: challenges and solutions. In *Proceedings of the Fourth international ICST Conference on Performance Evaluation Methodologies and Tools* (Pisa, Italy, October 20 - 22, 2009). International Conference On Performance Evaluation Methodologies And Tools & Workshops. ICST (Institute for Computer Sciences Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, 1-9. DOI= http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7988

[23] Mos, A. and Murphy, J. 2002. A framework for performance monitoring, modelling and prediction of component oriented distributed systems. In *Proceedings of the 3rd international Workshop on Software and Performance* (Rome, Italy, July 24 - 26, 2002). WOSP '02. ACM, New York, NY, 235-236. DOI= http://doi.acm.org/10.1145/584369.584403

[24] Kounev, S. 2006. Performance Modeling and Evaluation of Distributed Component-Based Systems Using Queueing Petri Nets. *IEEE Trans. Softw. Eng.* 32, 7 (Jul. 2006), 486-502. DOI= http://dx.doi.org/10.1109/TSE.2006.69

[25] Brosig, F., Kounev, S., and Krogmann, K. 2009. Automated extraction of palladio component models from running enterprise Java applications. In *Proceedings of the Fourth international ICST Conference on Performance Evaluation Methodologies and Tools* (Pisa, Italy, October 20 - 22, 2009). International Conference On Performance Evaluation Methodologies And Tools & Workshops. ICST (Institute for Computer Sciences Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, 1-10. DOI= http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7981