Performance Modeling for Service Oriented Architectures:

Paul Brebner  
NICTA  
Locked Bag 8001  
ACT 2601, Australia  
Paul.Brebner@nicta.com.au

Liam O'Brien  
NICTA  
Locked Bag 8001  
ACT 2601, Australia  
Liam.OBrien@nicta.com.au

Jon Gray  
NICTA  
Locked Bag 8001  
ACT 2601, Australia  
Jon.Gray@nicta.com.au

ABSTRACT

We present a tool for performance modeling of Service Oriented Architectures (SOAs). As mission-critical use of whole-of-government SOAs become pervasive, the capability to model and predict the performance of interdependent composite applications is critical. The tool can be used by architects early in the software engineering lifecycle to predict performance and scalability, to evaluate architectural alternatives, to provide guidance for capacity planning and the negotiation of Service Level Agreements (SLAs). It directly models and produces metrics for SOA applications in terms that are familiar to architects (services, workflows, and compositions of services). The tool enables the performance model to be generated from available architectural artifacts and performance data, making it easy to use. It is highly dynamic to facilitate interactive evaluation of alternative architectural choices. The tool can model complex deployment scenarios, including hosting multiple services on shared fixed or virtual servers. Development and evaluation of the tool was carried out in the context of architectural modeling for large-scale SOA-based Australian e-Government systems. The tool radically simplified the construction and execution of SOA performance models, and contributed critical architectural insights for the software engineering of these systems.

1. BACKGROUND and MOTIVATION

In recent years, many governments around the world, including the Australian Government [11], have launched e-Government initiatives. e-Government refers to the government’s use of information technologies to exchange information and services with citizens, business, and other arms of government. Although there are differences in emphasis from one country to another, these initiatives aim to: provide more responsive, convenient, and easier access to government information and services by citizens; reduce the cost and time for business when interacting with government; and seek efficiency gains across government agencies through rationalisation of systems and increased interoperability. Many government agencies currently have a legacy of IT systems developed over several decades. However, these have typically been developed in isolation and to agency specific requirements, and are not usually designed to integrate with other agencies and external systems. When e-Government solutions are built on top of these legacy systems, and services are delivered to citizens and business through the internet, the performance demands on the legacy systems can often exceed their original design capacity, placing the e-Government solution at serious risk of failure [12].

In Australia, there are examples of large and complex e-Government systems both under development and currently deployed. Some systems provide services directly to citizens [13] and business, whereas others provide mission-critical services to other federal, state or local government agencies [14]. Increasingly, these systems are being designed as SOAs. They are implemented as composite service applications (services of services), consuming both internal services and external services provided by other agencies, and therefore function in the dual roles as service providers and consumers. Because of their critical role in the delivery of services to citizens and business, it is vital to understand the performance and scalability limits of these SOA systems well in advance of switch-on. Demand for the service, particularly a new service not previously offered by government, is often hard to predict, and a mismatch between demand and capacity can lead to cases of system “meltdown” [1].

Load testing is a common strategy used to measure the capacity of traditional software systems, but it is often technically difficult to load test SOA applications end-to-end due to problems including: testing across organizational boundaries; security requirements; lack of tools and skills; high overhead of turning on low-level performance monitoring; the presence of resources that are shared with other organizations and/or production systems; the use of services provided by other organizations. Load testing may be perceived as a denial of service attack, SLAs may impose restrictions on use, or there may be a cost to use a service. By the time that integration testing is conducted on a production ready system it is inevitably too late and too expensive to radically change the software architecture to address performance and scalability deficiencies. It is therefore critical to predict the performance implications of architectural alternatives for SOAs early in the development lifecycle.

National ICT Australia’s (NICTA’s) software engineering group has been developing a range of related technologies to address these issues of performance assessment of large, heterogeneous service architectures. Demonstrations of specific tools and applications of this technology have previously been reported in the research literature [2, 3]. Since 2006 NICTA has undertaken a series of collaborative research engagements with Australian government agencies to validate the effectiveness of this approach to performance assessment. After repeated field trials of this technology on production systems delivering e-Government

services, a body of empirical results has been compiled regarding the accuracy of the methods and tools and the scope of systems to which this approach can be applied. With each field trial, the performance assessment methods are becoming more robust, repeatable, and mature. However, the long-term goal of this research is to develop an enduring technology that is transferable to, and usable by, software architects who provide technical leadership in major e-Government projects. The initial phase of technology development had been conducted with prototype tools intended for sole use by NICTA researchers. To facilitate the transfer of this technology into enduring use within government agencies, more robust and accessible tools were required.

We identified critical requirements for a performance modeling tool: support for composable modeling (models which can be reused and form sub-parts of larger models) of composite SOAs (services implemented as workflows consuming internal and external services, often implemented on an Enterprise Service Bus); support for realistic service deployment models (e.g., services deployed on shared fixed capacity servers, and services deployed on virtual servers); robust, easy to use and interactive when developing and maintaining the models. We evaluated a number of Open Source and commercial tools (e.g. [3, 7-9]), but were unable to find a suitable tool. All the existing modeling tools we investigated were found to be unsatisfactory in one or more ways. Such as needing extensive custom programming, or ad-hoc error-prone pre-processing of architectural and performance data prior to use; not supporting composable modeling of service compositions, or shared or virtual resourcing models; or were not sufficiently interactive. We therefore decided to develop our own tool focusing on SOA performance modeling.

2. TOOL OVERVIEW

2.1 How the tool is used

Our tool supports the steps followed by architects to enable them to easily produce SOA performance models, either from scratch or from existing architectural artifacts (UML sequence and deployment diagrams, service maps, hardware diagrams, etc). Models are visualized as a tree view of the available SOA model components (explained in section 3). Models are parameterized with measured performance data (from an unloaded system), hardware capacity (e.g. number of CPUs per server) and optional configuration information (e.g. number of virtual server CPUs per service). The tool automatically provides an extensive set of performance and scalability parameters and metrics appropriate for each type and combination of model component. The model simulation can be run, paused, restarted and reset. While the model is running, selected metrics are computed continuously and graphed, and selected parameter values are graphed and can be changed (e.g. using slider controls) giving immediate feedback. Multiple models can be run concurrently to compare architectural alternatives. Alternatives within a single model can also be compared (e.g. different workload ratios or workflow implementations).

The tool is intended for a range of architectural modeling tasks related to performance and scalability, including capacity and resource planning, modeling of complex workloads, modeling complex composite applications, tuning of service deployment options and server configurations (e.g. virtual servers), comparing architectural alternatives, and developing SLAs for services consumed and provided.

2.2 The user experience

The tool significantly reduces the cost of developing SOA performance models by reusing existing architectural artifacts, and modeling SOA concepts directly. This reduces effort, and increases correctness and flexibility. Modeling service compositions directly increases reuse of performance data, as times previously measured for existing services can be reused when modeling the performance of new compositions which also use those services.

We have found the interactivity of the tool beneficial in a variety of contexts, including individual use by an architect, in workshops with multiple participants, and for demonstrations and training. We produced an educational demonstration “game” in which a user adjusts multiple parameters in response to an unpredictable workload. The winner must keep the response time below 5 seconds, server utilization under 80%, and allocate spare CPUs to servers. Audible warnings are given when thresholds are exceeded. Figure 1 shows a screen shot of this use of the tool.

![Figure 1: Screen shot of a simple application of the tool](image)

3. MODEL COMPONENTS

We now outline the components of our SOA performance models: services (simple services, and composite services and workflows), servers, workloads, and metrics and parameters (See Figure 2). The tool was primarily designed to model services as first-order entities. Services are either simple or composite.

3.1 Simple Services

A simple service is defined with a service name and an optional time, which is the typical time taken by an unloaded server to process it. The actual time taken (corresponding to the response time metric) depends on the load on the server, and is computed by the model engine (see section 4). The time is optional as it may vary significantly depending on the use of the service, and custom times can be supplied by the invoker of the service (see 3.2).

Simple services are assumed to have no further implementation details available and are thus treated as a black box. This is because: they are implemented as a single component; the implementation details are unavailable (as if often the case with security sensitive or COTS systems); the service is implemented externally by a 3rd party, in which case the time may be described by a more complex function (e.g. a load dependent function), or a
SLA (guaranteeing qualities of service, and imposing obligations on consumers). Model parameters (3.5) can be used to express some SLAs.

### 3.2 Composite Services and Workflows

Composite services are “services of services”, for which implementation details are known, and include services explicitly implemented as workflows, and simple services that are delegated to multiple servers (modeled with a service for each server).

A composite service is defined by a service name, and a workflow, consisting of one or more steps. Each step has a service name, and a service time. If the service time is zero, the default service time is used if it is a simple service. The actual response time for each step is computed by the model. The response time for a composite service is the sum or the response times of all the steps.

A service may be called multiple times (potentially with different times) in a workflow. Modeling each call separately is desirable for two reasons. First, this is how real architectural artifacts represent workflows. For example, composite services can be modeled as sequence diagrams. Second, it enables the use of measured performance data directly for each workflow step (rather than having to pre-process it), and also enables the use of different times in each workflow instance representing the range of values encountered. In effect, replaying raw performance data obtained from repeated invocations of the service (imported into the tool from a file or database). In contrast, simpler Queue Network (QN) modeling approaches only support the use of a single aggregate time per server per workflow.

### 3.3 Servers

In order to execute, services (which are software) must be resourced (by hardware). Resources are modeled as servers, and services are deployed to servers (i.e. servers host services). A server is defined by a name, one or more services that are deployed to it, and a capacity (number of CPUs). Virtual servers (e.g. VMware, [10]) are an increasingly common hosting option used to maximize resource utilization and increase deployment flexibility. To model virtual servers the number of virtual CPUs per service is specified as weights. Virtual CPUs represent a share of the server capacity available, which depends on the total number of virtual CPUs and the load on the server. Weights modify the simulation server resource model to process proportionally more events from services with larger weights.

### 3.4 Workloads

Finally, in order for the model to actually do something it must be stimulated, by modeling external demand on the services. A workload models the consumption of a service from outside the system, and is defined by a name, the service invoked, the average time between invocations per user (arrival Interval), the number of users in the workload, and the arrival distribution (currently constant or exponential). The number of users can be interpreted as a workload ratio (e.g. 80:15:5). Multiple workloads can be defined for the same service, representing different classes of users. As well as being invoked “externally” by workload classes, services can also be invoked “internally” by a service composition.

### 3.5 Metrics and Parameters

The final components are metrics and parameters. Metrics are computed by the model. Parameters control the model, and can be set and changed by the user. Metrics and parameters are defined for specific model component types (e.g. services, servers, workloads and combinations), or at the top-level of the model itself. The default service and workload metrics are response time (ms), arrival rate (transactions per second, TPS) and throughput (outgoing TPS), and wait time (ms), the delay waiting for server resources. Current, minimum, maximum, average (cumulative), and time averaged (specified interval, default 1s), count (number of updates), and total (sum) values are computed for each metric by the model engine. The default server metrics are response time (ms), throughput (TPS), and utilization (percentage of resources that are busy over time). Parameter values can be set at the start of the model, or changed during the running of the model.

### 4. TOOL DESIGN

The tool was designed around models (consisting of the components described in section 3), a model simulation engine, and a Graphical User Interface (GUI). The GUI supports the steps in our SOA performance modeling method, allowing users to create, view and edit models, and to run them to obtain metrics. The tool was developed in Java, and real-time metrics were graphed using the Time Series chart from a free java chart library (JFreeChart). The models and engine have an Object-Oriented design enabling easy reuse, enhancement, and replication of the code.

The performance model is executed with a custom Discrete Event Simulation [6] simulator (also written in Java). The simulator components are FIFO queues and ”servers” (Figure 3). Simple services are implemented as queues with simulator ”servers” (representing model servers) pulling events off them whenever there is spare server capacity. This combination simulates the time spent in each service (waiting in the queue and execution by a server), and increasing server load (resulting in longer wait times). Special purpose simulator “servers” are used to implement model workload and composite service components, and execute workflows. As these do not model real world servers, but are purely used for simulation purposes, they have infinite capacity and take zero time to process events. Workload generators produce events (simple or composite service invocations)

---

**Figure 2: SOA Performance meta-model**

![Figure 2: SOA Performance meta-model](image-url)
according to the arrival interval, number of users, and distribution parameters of the corresponding workload model components. Composite services are implemented as workflow generators. These emit a workflow instance event which is passed to the workflow engine. The workflow engine manages the execution of workflows (steps through the workflow, maintains a workflow call-stack when further composite services are invoked) and computes the majority of the metrics.

Even though we used a simple iterative approach to find the next event in the simulation (c.f. maintaining an ordered Future Event List [6]), the simulation runs fast for models consisting of 10’s of services, workflow steps, servers, and workloads (1 simulated minute per clock-time second). The simulation can run indefinitely, or terminate at a specified stop time. The speed can be controlled, enabling long periods of real time (e.g. days) to be simulated in minutes. This is useful to measure the extremes of the load on the system over extended periods of time. For example, we can measure how long the system is saturated for per day as a consequence of load bursts from an exponential arrival distribution. This information can be used to determine a SLA for the service consumer (e.g. given an average arrival interval of 10s, the response time will exceed 5s for less than 30s in a 24 hour period). The impact of the convergence of multiple workloads with peaks at different times can also be explored, as this type of data is often available from existing legacy systems. Running the simulation slower than real time is also useful to understand the model dynamics.

5. EVALUATION
Planned tool enhancements will allow more explicit modeling of real services including operations, parameters, payloads, protocols, message exchange patterns (MEPs), and non-determinism in workflows. Other possible improvements include: the ability to reason with and generate SLAs expressed in a richer language (e.g. [5]); more flexible mapping of performance data to the performance model; automatic sensitivity analysis to explore the impact of modeling assumptions or variation in data; better support for comparing architectural alternatives through the use of model sub-classing; and modeling other non-functional attributes such as availability.

The tool reported in this paper is only part of a suite of related technology artifacts that also includes modeling techniques, technology assessments, design guidelines, development processes, architecture frameworks, and performance metrics analytics. Collectively these represent the NICTA SOA performance assessment technology. This technology has been refined and empirically validated through a series of field trials with collaborating government agencies in Australia. In depth reports of these field trials covering the empirical techniques used in the conduct of the trials, an assessment of the technology’s effectiveness and the business impact upon the participating organisations are currently being prepared for publication. The creation of the NICTA custom tool has played a significant role in the maturation of this technology. We anticipate further refinement of the technology through additional collaborations with government and industry partners during 2008-2009, with the aim of maturing the technology to a point were it is a candidate for commercial exploitation.

6. ACKNOWLEDGMENTS
Our thanks to our software engineer, Le Ma, for GUI development, and Liz Brebner for proof-reading the paper.

7. REFERENCES