# Performance Engineering of Service Compositions

John Grundy
Dept Electrical and Computer Engineering & Dept Computer Science
University of Auckland, Private Bag 92019
Auckland, New Zealand
+64-9-3737-599
john-g@cs.auckland.ac.nz

John Hosking, Lei Li and Na Liu
Dept Computer Science
University of Auckland
Private Bag 92019
Auckland, New Zealand
+64-9-3737-599
john@cs.auckland.ac.nz

## ABSTRACT
While a service-oriented approach to software engineering has become popular in recent times, the actual performance of systems composed from many distributed parts is still largely unpredictable. We describe our recent research applying performance test-bed generation techniques to service-oriented architectural models as an advance on the state of the art in performance engineering of service-oriented software. We outline our related research on tools for business process composition, performance engineering and dynamic system architectures.

**Categories and Subject Descriptors**
D.2.2 [Design Tools and Techniques]: Computer-aided software engineering; D.2.8 [Metrics]: Performance measures; D.2.11 [Software Architectures]: Domain-specific architectures.

**General Terms:** Measurement, Performance, Design, Experimentation

**Keywords:** Performance engineering, business process composition, domain-specific software tools, dynamic architectures

## 1. INTRODUCTION
The use of a service-oriented approach to software engineering has become popular, with a number of architectural design techniques, implementation technologies and exemplar applications having been researched in recent times [1][12]. One aspect of such service-oriented systems is that their component services can usually be composed and used in a variety of unplanned-for ways. A natural consequence of unpredictable service deployment and composition is unpredictable performance of the eventual composed system [1], [16].

We have been working on a number of related architectural design techniques, realization technologies and software tools applicable to the service-oriented software systems domain. These include architectural models augmented with aspects to better characterize
services, dynamic discovery and composition of these augmented service descriptions, and various realizations of these approaches via both augmented implementation technologies and standard component technologies [10][11][14]. All of these have demonstrated great unpredictability in performance of the resultant service compositions.


*IW-SOSE'06,* May 27–28, 2006, Shanghai, China.


Recently we have been extending and applying a performance test-bed generator, MaramaMTE [6], for use in the domain of service-oriented architecture performance analysis. Our eventual aim is to allow service compositions – both static and dynamic – to be modelled and performance test-beds to be generated – again both statically and dynamically – to assess the performance of both models and resultant service-oriented systems.

We outline a motivating example for our work, the key aspects of our approach, the set of co-operating tools for service composition modelling and performance analysis, and discuss preliminary results from our research to date. We conclude with areas for continuing future research.

## 2. MOTIVATION
A classic example of a service-oriented software system is a travel planning application built from dynamically discovered web services providing travel item search (flights, cars, hotel rooms etc), booking, payment, event scheduling and itinerary management. The application may allow dynamic discovery of appropriate services providing these functions and multiple, alternative service providers may be discovered. Services may provide limited or comprehensive functionality. Some may be free, others require payment. They may be from "trusted" providers or unknown 3rd parties. Some may support business transaction models, respond faster than others to requests, or support security models that others don't. An outline of such a system is shown in Figure 1. The client discovers various services from a web services registry e.g. UDDI (1). Flight searches are performed via various providers (2), which may use different protocols and data representations. Travel item bookings made directly or via agents (3), possibly using a payment system (such as credit card authorization) (4).

Whole rafts of issues present when composing such a system from discrete, distributed services as opposed to a classic monolithic single application:

- How to identify appropriate division of responsibility into services from one or more applications' set of requirements.
- How to implement, describe, deploy and advertise the constituent services.
- How to discover, integrate with and invoke a range of services from multiple providers and potentially using incompatible protocols, data representations etc. This may be done statically (compile time) or dynamically (run time), and the composition may change at run time e.g. the service goes down so another is discovered and integrated to replace it.
- Whether the composed set of services will meet various non-functional requirements of the overall application.

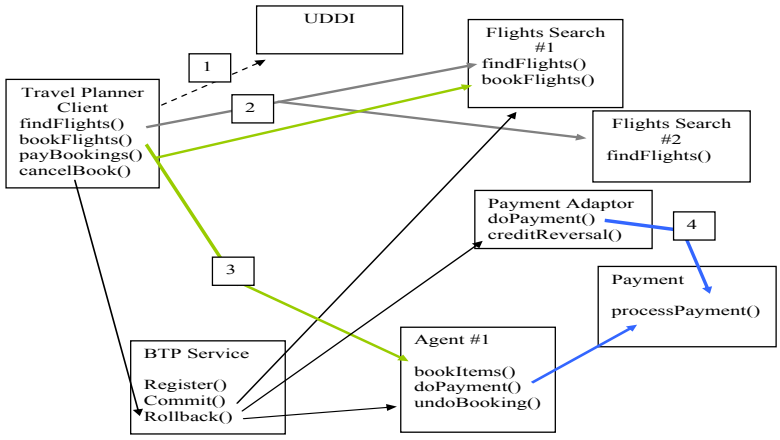


**Figure 1. Travel planner service-oriented system.**

The research we describe here attempts to address the last item, specifically issues related to the performance of the composed service-oriented software system. Performance measures may include transactional throughput of one or multiple composed services and response time of a service, set of services or a client application to the end users. Software engineers are interested in how the compositions will perform, how different compositions may perform, how the system performs under different loading scenarios, and how performance may change if the service compositions change dynamically. Ultimately performance is but one non-functional requirement; others that may need assessment include security, transactional consistency, resource utilization, reliability and fault-tolerance.

Currently there is a lack of suitable approaches and tools to support such performance engineering of service oriented systems. Various methods have been developed for performance engineering of highly distributed systems [4], [15]. However most of these assume static composition of components and often assume a fixed set of client and server components [2], [5]. Some efforts to performance engineer service-oriented architectures have been attempted, though usually via simulation rather than empirically [7], [8], [13]. Due to the complexity in performance engineering of large distributed systems some tools for this task have been developed [5], [7], [9]. However few of these have been applied to service-oriented software systems. There has been a trend to using models of software architecture to assist with performance engineering [3], [6], [12], [15], [16], [19]. Our work is closely related to this trend in its use of high-level architecture models, service compositions and client load models.

## 3. OUR APPROACH

Figure 2 provides an outline of our approach to performance engineering of service-oriented software systems. Formal models of service compositions are described (1) at a high level using one of our tools for specifying business processes and service composition (using either BPMN or Tool Abstraction-based modelling approaches). These high-level formal compositions are extended with a lower-level service composition model at the detailed service interface level (2) is then specified using the architecture design capabilities of our MaramaMTE performance test-bed generation tool. Both existing and proposed services can be incorporated into this model; proposed services have an implementation generated (3) by our service generation component of MaramaMTE. This latter approach can also be used to create a test-bed proxy for existing web services to avoid the need to directly interact with the live web service when estimating service composition performance. Client load models are developed in MaramaMTE (4) using another formal model, the form chart, to provide realistic client user interface load models. In addition, back-end service invocation load models can also be developed if required. From the composition and load models MaramaMTE generates one or more performance test-beds (5). These test-beds are executed (6) to stress-test the service compositions and results are presented to the engineer (7). Partial compositions may be tested or the entire service-oriented system. Compositions, load models and test descriptions can be modified (8) and performance tests re-run to compare different performance profiles.

## 4. EXAMPLE USAGE

Service compositions can be specified in three ways using our toolset: (1) by the use of the Business Process Modelling Notation [21]; (2) by use of our own ViTABaL-WS web service composition notation [14]; and (3) at a more detailed service interface level in MaramaMTE (based on the software architecture notation we developed for our earlier performance engineering tool, ArgoMTE [9]). Using BPMN has the advantage that it is a "standardized" approach to high-level service composition, where services represent business process stages and the notation provides a compositional metaphor.

Figure 3 (left) shows an example BPMN process specification for part of the travel planning application. The diagram shows itinerary creation and booking confirmation process as a set of composed BPMN process stages. Most stages correspond to available services e.g. searching for flights; confirming seats; paying for confirmed seats. These services have been assembled into a composition using BPMN's process stage compositional metaphor, providing a high-level compositional model. In this example BPMN tool, an Eclipse plug-in, we do not specify details of the protocols or message formats. In order to performance test such compositions we need to flesh out these details using MatamaMTE views (see below). Some of the BPMN stages shown here represent "error handling" for the service composition e.g. Notify Error and Find Alternative. These are not services at all but are part of the compositional logic used to resolve problems arising during the "normal" inter-service process flow.

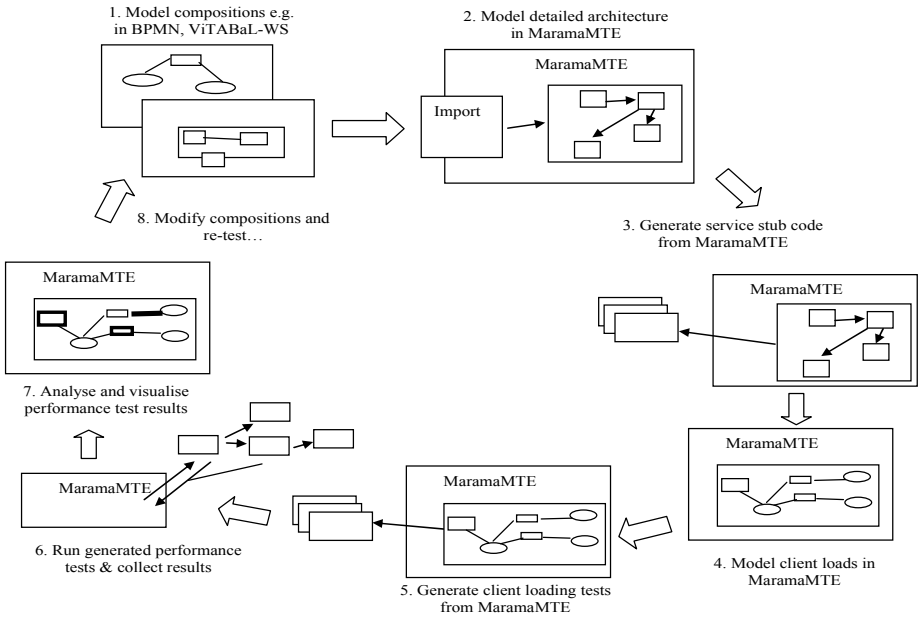


**Figure 2. Performance engineering of SOAs with MaramaMTE.**

The second way to specify service compositions at a high level is using our ViTABaL-WS tool, another Eclipse plug-in illustrated in Figure 3 (right). This uses a different service composition metaphor based on the Tool Abstraction paradigm [14]. In this example two messages (itinerary updates and travel booking) are received by different composite "toolies" (web service compositions), which invoke other services to carry out the composite business processes (updating the travel itinerary data to add/remove planned seats and to confirm/pay for seats respectively). Services are characterized as process-centric (ovals), data-centric (squares, e.g. flights and itinerary at the bottom) and exception-handling (e.g. find alternative at the bottom right). Again some of these "services" may be indeed discrete, remote service components e.g. a web service, while others are themselves composites or error handling logic for the composition. Unlike our BPMN modeling tool, ViTABaL-WS supports specification of message formats and port details for web services, i.e. more detailed service-oriented architecture information. While the services composed and inter-service messages are in common between BPMN and ViTABaL-WS, their compositional metaphors and constructs differ. BPMN is message-flow oriented while ViTABaL-WS supports event and control synchronisation. Their error handling specifications are quite different as are their abstractive approaches (enclosure vs drill-down**.**

We export from these high-level service composition views a formal model of the BPMN and ViTABaL-WS composite specifications. These are complementary and provide our MaramaMTE tool two overlapping architectural models that are used to assemble a partial architectural model for the target service-oriented software system. However, in order to run performance tests on these compositions further information must be supplied, including host/port information for each service; full message structure details; and a loading model for the "client" for each service composition.

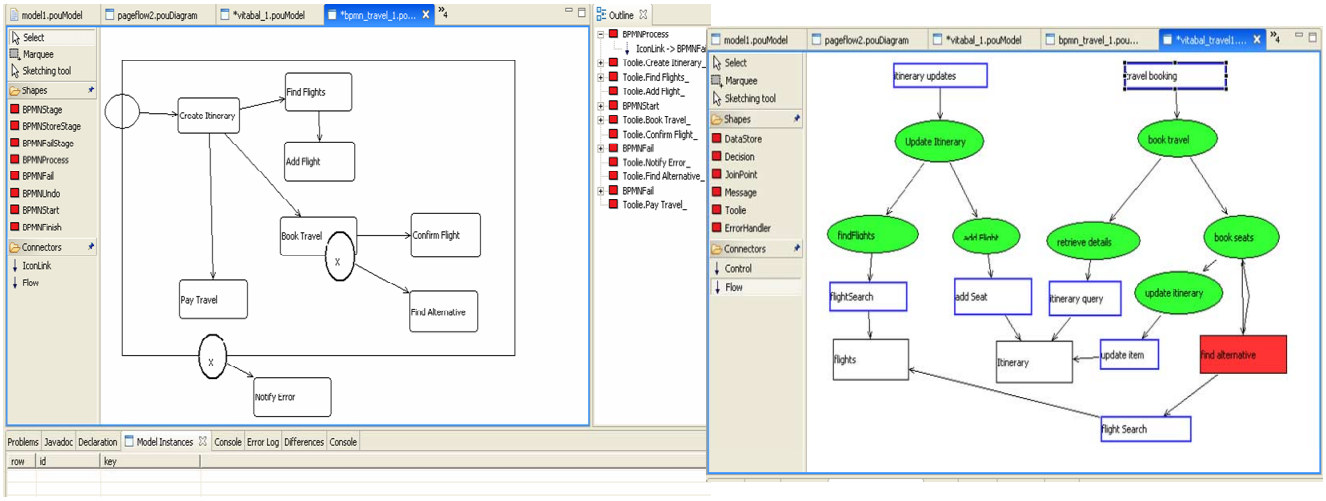


**Figure 3. BPMN and ViTABaL-WS service compositions.**

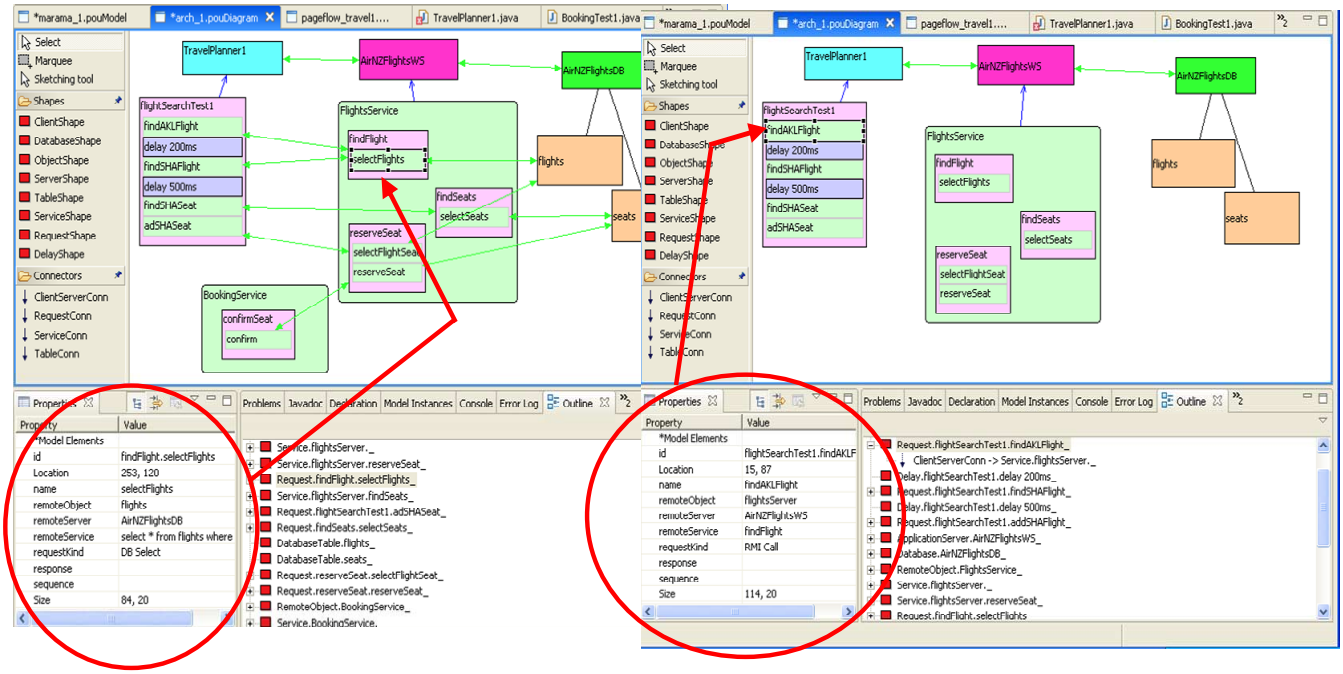


**Figure 4. MaramaMTE Architecture view of service-oriented system.**

Some of this information can be obtained automatically e.g. message format extracted from a Web Service Description Language (WSDL) document for a service. Other information must be supplied via MaramaMTE's architectural modelling view.

An example, part of the service-oriented architecture model for the travel planner is shown in Figure 4. This focuses on the flights server which provides three services: finding flights, finding available seats for a flight and reserving one or more seats. Each service has a host/port, input message format and response message format. A client load test that invokes these flights services (i.e. is a composition of these services) is shown on the left hand side at top, while another service that invokes the seat reservation service is shown left hand side at bottom.

In this example we also specify relationships from the services to a database, shown on the right hand side (AirNZFlightsDB) and database tables (flights and seats). This information is used to synthesise code for each service if the service doesn't yet exist i.e. to generate a performance test-bed for the services. The MaramaMTE user specifies request details for each service e.g. for the reserveSeat service there are two database requests, the first doing database select the second an insert. If a service already exists and we wish to performance test the real service the architect instead specifies information to construct a test input message to send to the service.

The example client load specification, FlightSearchTest1, is shown for one of the composites from Figure 3, the flight search and seat selection service composition. This simple load test describes a flight search/seat selection composite service invocation scenario with example user input data, user delay and user selection of service invocation result. In the property view in the screen dump on the right is shown some of the properties of one of the client load test requests to the findFlight service.

MaramaMTE generates code to implement this test (either a Java application, a Microsoft Application Centre Test load testing tool script or an Apache JMeter load testing tool script, depending on whether the actual client is intended to be an application or web browser). This simple load test when run will call the findFlight, findSeats and reserveSeat services multiple times with example message input data and delays between each mimicking client application delays. Multiple concurrent instances of this example load test can be run by multi-threading or even running single instances on multiple hosts [9].

The simple loading tests as illustrated above have proved to be accurate across a range of applications for stress testing the likely maximum throughput performance and concurrent usage of service compositions [6], [9]. However they are quite limited in terms of representing actual service client behaviour, especially when the client of a service composition is e.g. a web browser with user interaction. To this end we have developed a new client load modelling technique based on the Form Charts formalism [6]. An example of such a model is shown in Figure 5. A form chart specification models how users interact with submit/response type interfaces, like a web browser, and is composed of pages (ovals) and actions (rectangles). Each transition link from a page to an action to the next page has a probability assigned, and each page specifies a stochastic model to use to calculate likely user delay while viewing the page/filling out a web form before selecting an action. In this example the initial starting page is "login", which is denoted by the transition like from this form chart test, BookingTest1.

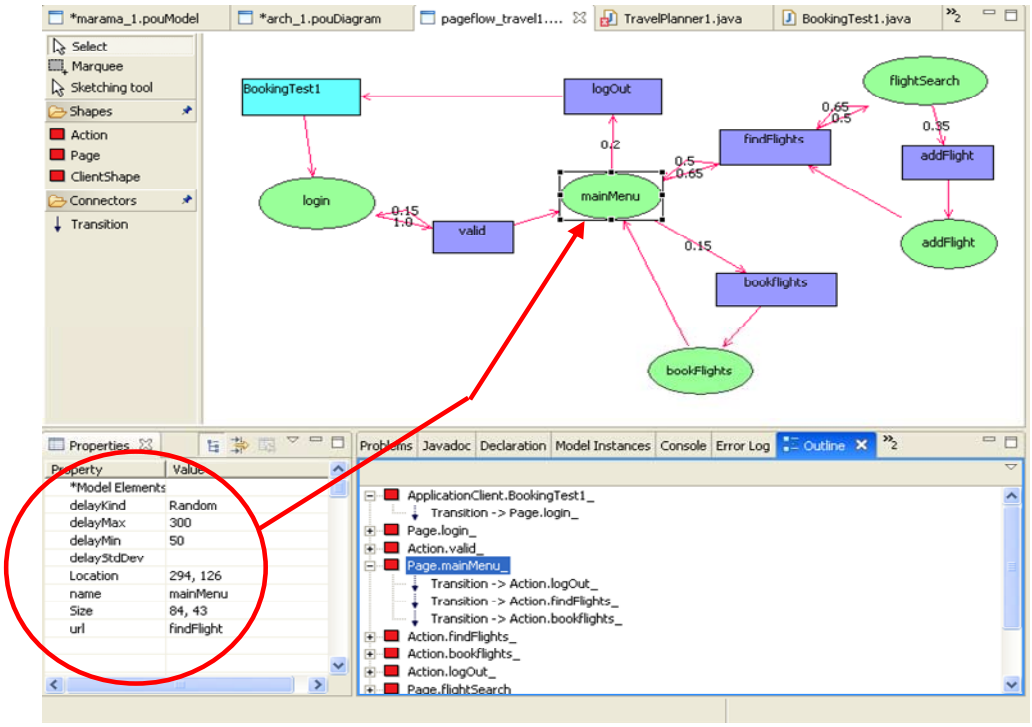


**Figure 5. MaramaMTE example client load model view.**

This example form chart model captures part of the flight search/seat selection and seat booking process a user would see realised in a web browser as a set of pages (login, mainMenu, flightSearch, addFlight, bookFlights). Each page has one or more actions the user can perform with different likelihood (e.g. find flights, book flights and logout for the mainMenu).

The architect specifies the service to invoke for each page (a URL), example form data to send to the service, and a stochastic delay algorithm and parameters (random delay, fixed, normal curve, based on historical data, or no delay).From this form chart model a more complete loading test for the service compositions can be generated – we generate a Java program implementing a state chart for the model – and run to stress test the composition. Again, multiple concurrent state chart programs can be run via threading or deployment of the generated Java client application to multiple hosts.

Results are collected from execution of the generated Java client program, possibly many multiple concurrent runs, and reported to the architect. Figure 6 shows an example of results for 10 concurrent threads of the form chart load model specified in Figure 5 on our exemplar travel planner service-oriented software system. Even though this is a simplistic example and we have generated stub code for the services from MaramaMTE (each service is simply a set of database operations via JDBC to a SQL Server database), some interesting results can be observed. Because this test run was made with client, services and database for services all running in the same host, the average time for all services is very low. However, concurrent service invocation causes bottlenecks and delays for the mainMenu and flightSearch services. The seat selection and booking confirmation for chosen seats use less time on average despite being more complex services as they don't suffer from much contention with only 10 concurrent users.

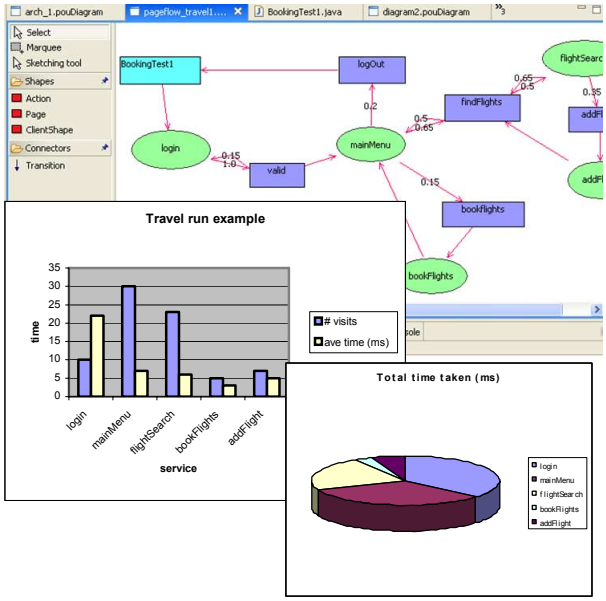


**Figure 6. Results of a performance test run.**

The software architect can use our tool suite to quickly modify a range of architectural, design and test properties and then re-run the load tests and compare results. For example:

- Merging or splitting service compositions in the BPMN and ViTABaL-WS tools will result in merging or splitting services in the MaramaMTE architectural model. This can dramatically impact performance e.g.

putting the flights and seats in different databases and accessed by services on different hosts will reduce performance under light loading (more communication) but potentially increase it under heavy loading (less network, thread pool and database pool contention);

- Using different database commands, service-to-service vs service-to-database and enabling simple cache code generation in the MaramaMTE architecture tool all impact performance e.g. less or more time in database; more realistic test data used cause database page hits to reduce; transaction commits for database in different places reduce/increase overheads; and cache hits reduce read time but distributed cache synchronisation may impact write time severely.
- Using different stochastic models for the form chart pages increase or reduce concurrent service invocations; increasing concurrent Java form chart-generated client load application runs increases concurrent accesses to services; using real historical data to replay service messages from generated client load application instead of stochastic model gives more realistic service loading tests.

Our current tool prototypes are all Eclipse plug-ins implemented with the help of a meta-tool generator. The architecture models in MaramaMTE are represented in an XMI format but currently do not leverage any standardised architectural representation model. The stub service code, client load test code and scripts, and form chart-generated Java code are all written to the Eclipse workspace and are synchronised with the Eclipse Java Development Toolkit, allowing on-the-fly re-generation, compilation and loading test runs.

# 5. DISCUSSION

Our approach, as currently implemented, has a number of strengths to it. Firstly, the suite of visual specification languages makes it very quick and straightforward to specify and generate both service compositions and test-beds for estimating the performance of those compositions. Implementers have a choice of modelling paradigms they can use and can approach modelling at several different levels of abstraction, from very high level process oriented descriptions through to detailed architectural views. The modelling tools are integrated into the commonly used Eclipse development environment.

Secondly test beds can be generated for either complete systems with fully implemented components or partial systems with stub implementations acting as proxies for either existing or yet to be developed services. The simplicity of the modelling approaches and the automatic generation capability means that compositions can be rapidly changed and tests re-run to empirically determine optimal configurations without significant programming overhead. Progressively exchanging proxy stubs for live services means an iterative approach can be taken to developing and testing a complete service composition. Extracting the specifications for newly developed or discovered services via WSDL or similar descriptions allows the MaramaMTE architectural models to be extended as required.

Our current implementation has a number of limitations with respect to addressing the issues identified in Section 2. Firstly, our modelling languages currently only support static compositions, with no support yet existing for dynamic discovery. Extending our approach to support this could be handled at two levels. On the one hand, at the level of modelling using our existing tools, it would be very straightforward to add support for interrogating a UDDI repository to provide candidate services to be added as model elements when designing compositions, i.e. a form of dynamic library support for the design tools. On the other hand, and with more difficulty, additional modelling elements could be added to our modelling languages representing dynamically selected services. These could specify a UDDI repository that, in the generated implementation, is interrogated to supply an appropriate service which is dynamically connected to. These elements could also specify a strategy to follow in selecting candidate services. Performance estimation in such cases is more difficult as the performance will depend on the particular service discovered at any time. Problematically, standard UDDI repositories lack sufficient detail in their service descriptions to allow an informed choice of service. We have been investigating aspect oriented extensions to WSDL to provide this type of detail for aspects such as performance [18].

Secondly, and related to the latter point, the system currently makes no use of service level agreements or contract specifications to judge the results. This would be relatively straightforward for us to add and some existing tools could be leveraged to both support SLA authoring and management [20]. This is one aspect of a broader limitation: as it stands the system has very limited oracle capability. It is incumbent on the end user to interpret the results presented rather than being provided with assistance in understanding whether they are "reasonable". Our plan is to integrate aspect-based specifications of web services and their compositions into both our high-level composition tools and into MaramaMTE [18]. We have used these to support rich, concern-based querying and dynamic integration of dynamic web services. Using them to assist specifying SLAs for services will also allow dynamic service composition performance test generation and test evaluation.

Thirdly from MatamaMTE we generate to a limited set of implementation technologies, currently only to Java RMI and HTTP invocations. This is sufficient to prove the overall concept of our approach but insufficient for serious service-oriented architecture performance analysis. Adding code generation for extra technologies, particularly SOAP, .NET remoting and J2EE message-driven beans is relatively straightforward with the implementation framework and meta tool we are using. Indeed we supported all of these target technologies in our earlier Argo/MTE performance test bed generator [9].

We currently generate a state machine to implement the form chart-based client load tests using a stochastic model for client interaction and very limited use of historical usage data and server responses to requests [6]. We are planning further work to explore the use of web application session and history capture techniques [17] to provide highly realistic service-oriented software system loading.

Finally the system has no support for asynchronous performance evaluation. This has been shown to be a very challenging area for assessing component-based software system performance, both using (hand-coded) empirical performance test-beds and simulation models [4][15]. We are planning further work using

such techniques with our MaramaMTE test-bed generator to explore asynchronous service-oriented architecture performance analysis.

Key areas of future work include support for dynamic service discovery service adaptation, incorporation of aspect-based and other service level agreement approaches in both the specification of compositions and analysis of results, and extension to cover additional technologies. In addition, we have plans to improve the visualization of results. In particular we intend to add a more dynamic visualization of system performance providing the ability to capture and replay the load on services to observe temporal variations. This will assist architects locating potential performance issues arising from service compositions.

## 6. CONCLUSIONS

We have described a novel approach that allows specification and generation of both actual service compositions and test-beds for estimating service composition performance. The tool suite we have developed supports modeling of compositions using a variety of abstractions. The test-beds generate deployable code rather than using a simulation based approach. The tool suite is at a proof of concept stage, currently having been applied to only a limited range of problems and generating only to a limited set of implementation technologies. Nevertheless, results are encouraging and the tool suite is readily extensible to make it more practically usable.

## 7. REFERENCES


[1] R. Agrawal, R. Bayardo, D. Gruhl, and S. Papdimitriou. Vinci: A service-oriented architecture for rapid development of web applications. In WWW10, Hongkong, May 2001.

[2] F. Andolfi, F. Aquilani, S. Balsamo, and P. Inverardi, Deriving Performance Models of Software Architectures from Message Sequence Charts ACM Proc. Second Int'l Workshop Software and Performance, pp. 47-57, 2000.

[3] F. Aquilani, S. Balsamo, and P. Inverardi, Performance Analysis at the Software Architecture Design Level, Performance Evaluation, vol. 45, no. 4, pp. 205-221, 2001.

[4] S. Chen, Y. Liu, I. Gorton, A. Liu: Performance prediction of component-based applications. Journal of Systems and Software 74(1), 2005, pp. 35-43.

[5] V. Cortellessa, and R. Mirandola, Deriving a Queuing Network Based Performance Model from UML Diagrams ACM Proc. Int'l Workshop Software and Performance, pp. 58-70, 2000.

[6] D. Darheim, J. Grundy, J. Hosking, C. Lutteroth, G. Weber, Realistic Load Testing of Web Applications, In Proceedings of 10th European Conference on Software Maintenance and Reengineering, Bari, Italy, 22-24 March 2006.

[7] M. Gerndt, Automatic performance analysis tools for the Grid, Concurrency and computation: practice and experience, 2005, Volume 17.

[8] I. Gourlay, M. Haji and K. Djemame. Performance Evaluation of a SNAP-based Grid Resource Broker, Proceedings of the 1st European Performance Engineering Workshop (EPEW'2004), Toledo, Spain, September 2004.

[9] J. Grundy, Y. Cai and A. Liu, SoftArch/MTE: Generating Distributed System Test-beds from High-level Software Architecture Descriptions, Automated Software Engineering, Kluwer Academic Publishers, vol. 12, no. 1, January 2005, pp. 5-39.

[10] J.C. Grundy and J.G. Hosking, Engineering plug-in software components to support collaborative work, Software – Practice and Experience, vol. 32, Wiley, pp. 983-1013, 2002.

[11] J.C. Grundy, G. Ding, and J.G. Hosking, Deployed Software Component Testing using Dynamic Validation Agents, Journal of Systems and Software: Special Issue on Automated Component-based Software Engineering, vol. 74, no. 1, January 2005, Elsevier, pp. 5-14.

[12] R. Heckel, R. and M. Lohmann, Towards Contract-based Testing of Web Services, Electronic Notes in Theoretical Computer Science Vol. 82 No. 6, 2003.

[13] M. Kano, A. Koide, T.-K. Liu, and B. Ramachandran, Analysis and simulation of business solutions in a service-oriented architecture, IBM Systems Journal, Volume 44, Number 4, 2005.

[14] N. Liu, J.C. Grundy, J.G. Hosking, A visual language and environment for composing web services, In Proceedings of the 2005 ACM/IEEE International Conference on Automated Software Engineering, Long Beach, California, Nov 7-11 2005, IEEE Press, pp. 321-324.

[15] Y. Liu, I. Gorton, Performance Prediction of J2EE Applications Using Messaging Protocols. Proceedings of 2005 Symposium on Component-based Software Engineering, 2005, pp. 1-16.

[16] A. Mos and J. Murphy, A Framework for Performance Monitoring, Modelling and Prediction of Component Oriented Distributed Systems, Proc. of ACM 3rd International Workshop on Software and Performance, pp. 235-236, ACM Press, Rome, Italy, July 2002.

[17] S. Sprenkle, E. Gibson, S. Sampath, L. Pollock, Automated Replay and Failure Detection for Web Applications, In proceedings of the 2005IEEE/ACM International Conference on Automated Software Engineering, November 2005.

[18] S. Singh, J.C. Grundy, J.G. Hosking, and J. Sun, An Architecture for Developing Aspect-Oriented Web Services, In Proceedings of the 2005 European Conference on Web Services, Vaxjo, Sweden, Nov 14-16 2005, IEEE Press.

[19] J. Skene and W. Emerick, Model Driven Performance Analysis of Enterprise Information Systems, Electronic Notes in Theoretical Computer Science Vol. 82 No. 6 (2003).

[20] J. Skene, D. Lamanna, W. Emerick, Precise service-level agreements, In 26th International Conference on Software Engineering, 2004, pp. 179-188.

[21] S.A. White, An introduction to BPMN, IBM, May 2004, http://www.bpmn.org/