Modelling Dynamic e-Business Applications Using Historical Performance Data*

David A. Bacigalupo, James D. Turner, Stephen A. Jarvis and Graham R. Nudd

High Performance Systems Group, Department of Computer Science,
University of Warwick, Coventry, UK
daveb@dcs.warwick.ac.uk

Abstract. Distributed e-business application platforms such as the web services framework will require sophisticated workload management infrastructures for the matching of client requests to the most appropriate resources. This paper introduces a novel dynamic predictive framework that uses both historical data and analytical performance modelling to extrapolate a request's predicted performance under varying workloads and conditions on different systems. Such a framework can facilitate a workload manager in maximising the extent to which service level agreements can be met. Initial results are presented analysing how the response times of an e-business benchmark can be predicted at different system loads and on different application servers.

1 Introduction

The web services framework [1] will facilitate the performance modelling of e-business applications at higher levels of abstraction. Of particular importance will be using performance models to enable workload management systems to predict when Service Level Agreements (SLAs) are likely to be broken and so take appropriate management actions, whilst still making efficient use of the system's resources.

Quality of service (QoS) based resource allocation services with the ability to adapt to the continuous variations in user demands and resource availability will also be of great importance in the emerging field of Grid computing [2]; geographically dispersed resource-sharing networks and disparate heterogeneous resources whose management, rather than being centralised, is maintained through multiple administrative domains.

Traditionally, Grid computing has been focussed on 'e-science' applications, although more recently it has been proposed that Grids also provide a supporting infrastructure for e-business [3]. However, the different characteristics of these classes of applications (e.g. large 'run-once' jobs verses high-frequency request-

* The work is sponsored in part by the EPSRC e-Science Core Programme (contract no. GR/S03058/01), the NASA AMES Research Center administered by USARDSG (contract no. N68171-01-C-9012), the ESPRC (contract no. GR/R47424/01) and IBM UK Ltd.

driven services) necessitate specialised performance modelling and resource allocation methodologies for each.

This work proposes building on existing performance model-assisted scheduling of scientific jobs, and the performance monitoring and modelling of e-business systems so as to:

- model the potential dynamic and heterogeneous characteristics of e-business applications and platforms so historical performance data can be extrapolated, despite changes in system state (or workload being moved to new servers).
- facilitate the incremental modelling of an e-business application, while evaluating the complexity of the model at each step. This allows the analysis of how the increased accuracy of a more complex model can be correlated with the overhead of the extra monitoring required.
- apply this modelling to the *QoS* and *priority*-based self-management of e-business systems, so such systems can reconfigure themselves to maximise their ability to meet SLAs, whilst still making efficient use of underlying resources.

The remaining sections of this paper provide: an overview of the performance prediction framework for e-science Grid applications and how this is being adapted to model e-business applications; a description of the e-business system model; a discussion of the experimental validation of the modelling process and a conclusion.

2 Related Work

The analytical performance modelling and prediction of e-business systems for the purposes of capacity sizing has received much attention; for example modelling Java 2 Enterprise Edition (J2EE) applications using layered queueing models [4]. There has also been more recent work on QoS-oriented performance modelling, for example using Stochastic Activity Networks for the purposes of determining the QoS guarantees that can be offered [5]. However, this work is not automatically applicable for the kind of frequent 'on-the-fly what-if' predictions that are required by a dynamic self-managing system. Such predictions require modelling the current system state (which may be highly dynamic), and benefit from simplified models. Although there has been some work in this area, for example the historical performance data and QoS-based workload management system described in [6], such work tends to be focussed on closed proprietary systems.

The definition and monitoring of web service SLAs has also received some attention (i.e. [7]), however much of this work focuses on how to unambiguously and precisely define a SLA in terms of underlying system metrics. There has been less discussion of how a heterogeneous system might dynamically reconfigure itself to meet SLAs using selected performance metrics and control variables, which is the focus of this work.

Characterising and predicting the performance of parallel, scientific applications has been the main focus of the High Performance Systems Group's work for the last 10 years. A Performance Analysis and Characterisation Environment (PACE) [8,9] has been the most noted product of this research, providing a framework for

developers to create detailed analytical performance models that can be used to predict the computation and communication performance of C, Fortran and Mathematica codes. This environment has been verified by the UK Defence Electronic Research Agency (DERA) to have a predictive accuracy within 10% of measured values. Current work includes the extension of this environment for the modelling of Java-based scientific applications and the Java Virtual Machine (JVM).

A key feature of this approach to performance is the use of a flexible layered framework allowing performance critical aspects of the sequential code, parallisation strategy and platform to be modelled separately. This inherent separation allows predictions of the same application using different configurations, a case of simply inter-changing models at the appropriate layer. The layers used in PACE are as follows:

- an *application layer* for defining global model parameters;
- a *transaction map layer* that characterises the order and parallel execution of transactions and the inter-transaction communication;
- a *transaction layer* for the identification and characterisation of performance critical transactions within an application;
- a *platform layer* that describes the performance of the virtual machine and resource that the application is to run on.

PACE provides considerable modelling flexibility as transactions can represent any unit of work and computation/communication combination. Typically the kernel of Java e-science applications are modelled in detail at the method and communication API level. For example transactions might include the main computations on the data set and the communication of sub-sets of this data between nodes, and the transaction map might specify the order in which these actions take place across a cluster and the nodes which participate in each action. Java bytecode sequences and intra-cluster communications (assuming local communication time is a function only of data size; a common scenario in dedicated scientific clusters) are timed via benchmarks to provide the basis of a predicted execution time prior to execution. The model is then refined as the application executes through a process of runtime monitoring, which is facilitated by the automatic model-based instrumentation of the application, for example by manipulating the Java bytecode [10].

Currently an in-house XML vocabulary based on the Java ARM standard [11] is used to define each layer of the model, although as e-science applications start to be structured using the Open Grid Services Infrastructure [3] it is planned to update this so it is based on Grid, Web Service and Enterprise Computing standards. Of particular interest is defining transactions in terms of the application's WSDL [12] interfaces and the transaction map in terms of BPEL4WS [13] flows. Performance data will be represented and managed using emerging unifying technologies such as the CIM Metrics Model [14] and those being developed for the Java Environment [15]. Tool support will continue to be provided by the PACE toolset.

A key application of this performance modelling is the scheduling of distributed scientific applications over Grid architectures through the use of the TITAN predictive scheduler [16]. As these predictions can be calculated in real-time, TITAN can be continuously updated as new applications are launched and as the available resources change. TITAN uses iterative heuristic algorithms to improve a number of

scheduling metrics including makespan and idle time, while aiming to meet QoS requirements including dead-line time. The work is also differentiated from other Grid scheduling research through the application of A4 [17] - an agent-based resource discovery and advertisement framework.

3 Adapting the Predictive Framework for e-Business Applications

Adapting e-science based performance modelling techniques for business to business (B2B) and business to consumer (B2C) applications is challenging as these classes of application:

- tend to be more dynamic; the workload entering e-business systems can vary widely and a very high frequency of client requests is common, so systems must use dynamic mechanisms to divide and process workloads across different resources.
- often contain a more verbose codebase than e-science kernels, but the code is more likely to be well structured into modules with explicit interfaces, such as web services.

In such a scenario it is more appropriate to focus on predicting performance *during*, as opposed to *prior* to execution, based on historical performance data. It is also more appropriate to take a top-down approach when modelling e-business systems, as opposed to detailed source code-level modelling.

This research adapts an approach involving iteratively modelling the e-business application. Each iteration the modelling is made more fine-grained by breaking the processing required to handle a request down further into transactions, and specifying additional performance data to be associated with each sampled request (and how often that data is sampled). Data to be recorded includes: the client that initiated the request and the operation that was invoked at the application layer; transaction performance metrics (i.e. mean response time and variability) at the transaction and transaction map layers; and the servers that processed the request and the state they were in (i.e. load) at the platform layer.

The model is then used to populate a historical performance repository as the application is run, from which trends are extracted as to how performance metrics are affected by changes of server, system state, the client originating the request and the operation invoked. The overhead associated with the required monitoring is determined and a decision is made as to whether to make the modelling more fine-grained to obtain additional historical data, despite the associated increase in overheads. The aim is to stop the iterative process as soon as the modelling is sufficiently detailed to use historical data for the predictions required.

Previous work on applying PACE to e-business includes an investigation into the metrics that can be used for predicting response times and evaluating the accuracy of predictions [18,19]. This was used to enhance a workload management broker from the IBM Web Services Toolkit [20], based on the use of performance predictions in the TITAN scheduler. This has been extended so as to incorporate metrics for scaling response time measurements recorded at different server loads [21].

4 e-Business System Model

This section details how the framework described in section 3 is used to model a common architecture for e-business applications. This architecture is derived in part from a case study of the IBM Websphere platform [22]; a commercial Java implementation of a hosting environment for applications built using web, J2EE and web service standards[1]. Applications running on the Websphere platform are typically divided into a client tier, a tier of application servers that host the computation-centric part of the application, and a data access tier of one or more database servers and legacy systems, which are not directly accessed by the client tier.

The complete model, which will be used as a basis when iteratively modelling applications, is illustrated in Fig. 1. The application server transaction could also be broken down further into the Java thread pools, database connection pool and associated pool queues, which are used to process requests in parallel.

Workload is modelled as x clients who repeatedly submit requests to the system with 0 seconds think time between receiving a response and sending the next request. For simplicity it is assumed that the workload mix of the different operations being requested and the complexity of processing the data associated with those requests are, on average, constant. The main control point for the model is the number of clients that are directed at each application server.

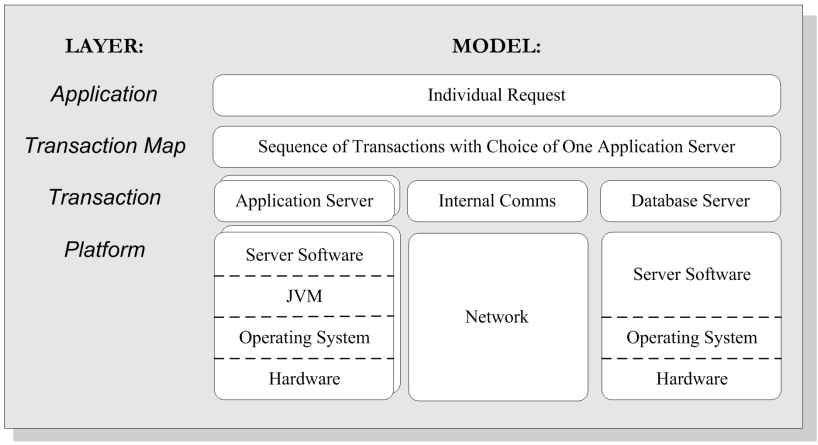

Fig. 1. The proposed 4-layer model of an e-business system. The performance of a request is broken down into three transactions: the processing on one of the application servers (assuming a copy of all optional application-tier functionality such as a dedicated Web/HTTP Server is hosted on every application server); the internal communication i.e. from the application server to the database server; and the data access on a single database server. The transaction map links the transactions sequentially in the order listed with a choice as to the application server used. The platform is modelled as consisting of application server, internal network and database server resources. The server resources are further divided into application/database server software, JVM, operating system and hardware sub-layers

[1] Although the current focus is on Java, the framework should be appropriate for other e-business and web service platforms, such as .NET.

5 Experimental Analysis

The Websphere platform includes a stock-trading application known as 'Trade' with an underlying design representative of current e-business applications [23]. Trade is used as a case study to test the proposed modelling process. It is run on a Websphere testbed consisting of a 'fast' application server ($AppServ_F$), a 'slow' application server ($AppServ_S$) and a database server capable of supporting the application-tier servers without causing any bottlenecks[2].

The aim of this initial set of experiments is to investigate the relationship between Trade/Websphere mean request response times and i.) the load on the system and ii.) the division of clients between the two application servers, when other variables remain constant. The first variable represents changes in the demand for the system, and the second is the main control point used by the system to adapt to changes in demand. These are important variables, as the proposed predictive framework will need to be able to predict the effect of alternative control point settings, given the current load on the system. This information will assist a workload manager in deciding how to divide the clients across the application servers so as to provide mean response time QoS levels. For simplicity, only one of the variables will be changed at a time. Future experiments will expand on this, so as to investigate predicting a response time at a different system load *and* workload division, than has been encountered historically.

Trade is accessed via HTTP requests to an application-tier interface consisting of 10 operations including 'buy', 'sell' and 'quote'. Requests to the buy operation are selected for detailed analysis as representatives of requests which read and write to multiple database tables. The performance of buy requests is measured in each experiment as follows. A background workload mix (defined as part of Trade as being representative of real e-business clients) is directed at the application-tier interface. An additional instrumented workload is used to sample response times of buy requests on all active application servers. This workload runs for 30 iterations, each iteration sending a 'create new user' request and then measuring the response times of 10 buy requests for a total of 300 samples per graph data point[3].

New users are created every 10 requests as the result of a buy operation displays a user's portfolio, so as a client performs a sequence of buy operations the amount of

---

[2] Testbed setup: Trade v2 running on Win2000 Adv. Servers: 2 Websphere Application Servers v4.0.1 ('Fast': Pentium4 1.8GHz 1GB RAM, 'Slow': Pentium3 450Mhz 416Mb RAM) and 1 DB2 Server v7.2 (Athlon 1.4Ghz, 512Mb RAM). Workload generated using between 1 and 20 homogeneous Red Hat Linux client machines (Pentium4 1.8MHz 512Mb RAM) running Apache JMeter 1.8. Each machine has a 100Mbit network connection to the switch.

[3] In our initial implementation using the Java 1.3.1 HTTP library and the Websphere v4.0.1 embedded Java HTTP server, excessive communication delays of up to 80 seconds are occasionally encountered. This is due to the implementation of the HTTP 1.1 keep-alive protocol, which was used so as to avoid having to open and close a connection for each request. (On a real system this would be done on a dedicated HTTP server tier.) This affected approximately 15% of buy request response times. Measurements suffering from this fault were removed prior to calculating the mean response time from the remaining measurements, which were not affected by bottlenecks (or any other errors/rejected requests). Future work will investigate enhancing the definition of QoS to explicitly report this kind of QoS failure.

data to be read and transmitted increases, as does the response time. This workload therefore represents a client population with an average portfolio size of 5.5. This illustrates the importance of associating an additional piece of data with a sampled request: the complexity of the data access and data processing that took place. Results suggest that the amount of data about the client in the database, that had to be processed (i.e. number of portfolio records), would be a useful metric.

5.1 Predicting Response Times at Different System Loads

In this section an experiment is conducted to examine if there is a simple relationship between response time and system load, when other variables remain constant. This is examined when a single application server ($AppServ_S$) is active, although similar results have been obtained for $AppServ_F$.

The first step is to create the simplest possible model of Trade so as to predict the response times of buy requests when there is no change of servers or system state, i.e. with a constant workload directed at $AppServ_S$, with the other server unused. The model consists of a single transaction to measure the response times of buy requests, and it is confirmed that this metric is an accurate predictor of future buy requests. A measure of the accuracy of the prediction is calculated based on the number of recent measurements available and their variability, as described in [18].

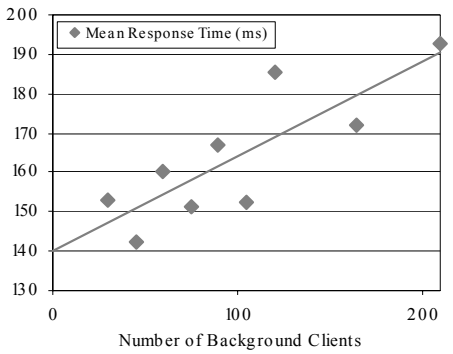

Fig. 2. Scalability graph for $AppServ_S$ when no other application servers are active. The mean response time is shown, for buy requests at increasing load in terms of the number of clients.

It is then necessary to determine which metrics can be used to scale historical request response time measurements, based on the difference between the historical and current 'load' on an application server. This technique will be used when a prediction is required at a load that has not been encountered in the historical performance data. Possibilities for representing load include % CPU Utilisation at the operating system level, and requests processed per second or number of clients at the server software level.

Fig. 2 shows the mean response time, for buy operations at different server loads as represented by the number of clients. It can be seen that there is a consistent increase in mean response time as the workload, as measured by the number of clients,

increases. However requests processed per second and % CPU utilisation are less useful as representations of system load, remaining constant at 117-120 requests per second and 100% CPU utilisation respectively, for all points on the graph. This is because these metrics do not properly capture the parallel request processing capacity that is available inside the servers. As a result of this, these metrics reach their maximum values whilst the application server still has parallel capacity available.

It is concluded that if number of clients is used to represent system load, it is likely that it will be practical to predict mean response times at system loads that have not yet been encountered in the historical performance data.

5.2 Predicting Response Times when the Division of Workload between Application Servers is Changed

The next step involves investigating if there is a simple relationship between the division of clients across heterogeneous application servers and the mean response times experienced by clients on those servers. (Assuming other variables remain constant and the application servers share the same database server.)

Fig. 3. shows the experimental setup which is used to measure the mean response times of $AppServ_S$ and $AppServ_F$. This procedure is performed repeatedly, with the division of clients between the servers changed each repetition. A constant workload of 225 clients is selected as this is the maximum number of clients each application server can support without having to reject any requests due to a lack of resources.

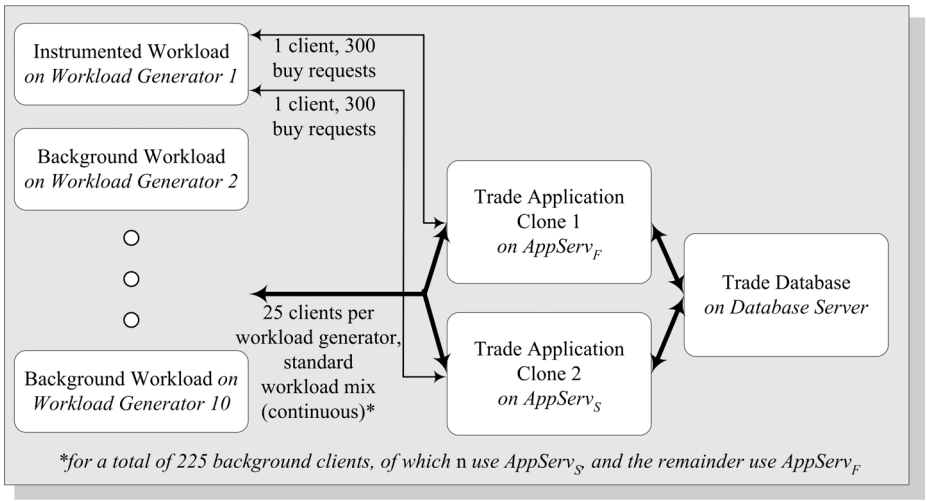

Fig. 3. The experimental setup used to obtain each point on the graph in Fig. 4. Initially n=75, so 75 background clients are directed at $AppServ_S$, whilst 150 background clients are directed at $AppServ_F$. After a warm-up period of 1 minute, the response times of 300 buy requests on each application server are measured. The process is then repeated, decrementing n by 25 until a point is found where the response times of the two servers are roughly equal.

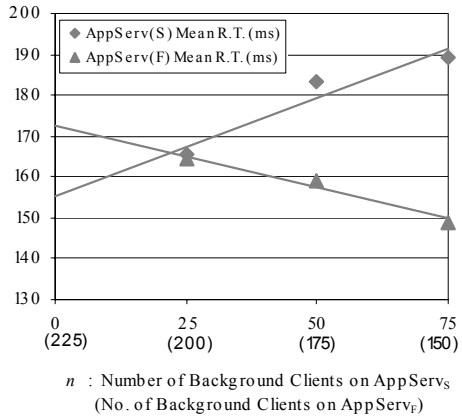

Fig. 4. Effect on $AppServ_F$ and $AppServ_S$ mean request response times, as the value of n is increased.

Fig. 4 shows the mean response times that are sampled as clients are progressively transferred from $AppServ_F$ to $AppServ_S$. It can be seen that the QoS levels provided by $AppServ_S$ and $AppServ_F$ appear to be steadily getting worse and better, respectively. This suggests that it is likely that it will be practical to predict response times, when the division of clients between application servers is changed.

When 25 clients are assigned to $AppServ_S$, and the remaining 200 are assigned to $AppServ_F$ the servers deliver similar mean response times of around 165ms. This would be a useful division of workload if all clients are to be treated equally. However if this is not the case, a workload manager can use this kind of graph (assuming there is sufficient historical data to generate it for the current number of clients), to determine the differential QoS levels that can be provided by allocating clients to application servers in different proportions.

The following is an example of how Fig. 4 might be used when the current load is 225 clients. A workload manager could give clients the option of paying for a QoS agreement with a guaranteed maximum mean response time, RT_{goal}, over some period of time. It is assumed that the value of RT_{goal} is the same for all QoS agreements and that $RT_{goal} < 165$ms. Each client's QoS agreement would also specify a financial penalty (perhaps determined by the price paid), that the service provider must pay if it misses the goal. The decision as to how to divide the clients would then be made based on minimising the total financial penalty to be paid.

If most of the clients decided to pay for a QoS agreement it might be appropriate to aim to limit the number of clients on $AppServ_F$ so as to keep the response time on this server below RT_{goal}. A value of n could be chosen so as to give an expected mean response time on $AppServ_F$ just under this goal. Clients could then be assigned to $AppServ_F$ starting with those with the highest financial penalties until $AppServ_F$'s capacity of (225-n) clients is reached. The remainder would have to be 'sacrificed' onto $AppServ_S$, and probably miss their QoS goals. However as a result of using this predictive strategy, the service provider would avoid paying penalties to the more expensive clients on $AppServ_F$.

6 Conclusions and Future Work

This paper reports on how an established performance modelling and predictive framework for e-science Grid applications is being enhanced for e-business applications. This will allow the dynamic and heterogeneous properties of e-business systems to be modelled; a request's expected performance on proposed new servers can then be extrapolated from historical data recorded on established servers in a range of states. Results are presented showing how an e-business benchmark is being analysed so as to explore the metrics that can be used to scale response times recorded at different system loads. It is also shown how the analysis is being extended to extrapolate response times when clients are allocated to heterogeneous application servers in different proportions.

Future work will investigate applying the performance analysis described in this paper to the dynamic SLA-based workload management of e-business requests. XML-based PACE modelling languages and associated tool support will also be extended for this new class of application.

Acknowledgments

The authors would like to express their gratitude to IBM's T. J. Watson Research Center and Hursley Laboratory for their contributions towards this research, and in particular to Robert Berry, Donna Dillenberger and Beth Hutchison.

References

1. F. Curbera, W. Nagy and S. Weerawarana, "Web Services: Why and How", OOPSLA 2001 Workshop on Object-Oriented Web Services, Florida USA, 2001
2. I. Foster and C. Kesselman, The Grid : Blueprint for a New Computing Infrastructure, Morgan Kaufmann Pub., 1998, pp. 279-290
3. I. Foster, C. Kesselman, J. Nick, S. Tuecke, "Grid Services for Distributed Systems Integration", IEEE Computer, Vol 35, No. 6, 2002, pp. 37-46.
4. T. Liu, S. Kumaran, Z. Luo, "Layered Queueing Models for Enterprise JavaBean Applications", 5th IEEE International Enterprise Distributed Object Conference, Seattle Washington, 2001, pp. 174-178
5. D. Daly, G. Kar, W.H. Sanders, "Modeling of Service-Level Agreements for Composed Services", 13th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 2002), Montreal Canada, 2002
6. J. Aman, C. K. Eilert, D. Emmes, P. Yocom, and D. Dillenberger, "Adaptive algorithms for managing a distributed data processing workload", IBM Systems Journal, Vol. 36, No. 2, 1997, pp. 242-283.
7. A. Keller, H. Ludwig, "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services", IBM Research Report, May 2002, Available at: http://www.research.ibm.com/

8. J. Cao, D.J. Kerbyson, E. Papaefstathiou and G.R. Nudd, “Modelling of ASCI High Performance Applications using PACE", 19th IEEE International Performance, Computing and Communication Conference, Phoenix USA, 2000, pp. 485-492
9. G.R. Nudd, D.J. Kerbyson, E. Papaefstathiou, S.C. Perry, J.S. Harper and D.V. Wilcox, “PACE - A Toolset for the Performance Prediction of Parallel and Distributed Systems", International Journal of High Performance Computing Applications, Special Issues on Performance Modelling. Vol 14, No. 3, 2000, pp. 228-251
10. J.D. Turner, D.P. Spooner, J. Cao, S.A. Jarvis, D.N. Dillenberger and G.R. Nudd, “A Transaction Definition Language for Application Response Measurement", International Journal of Computer Resource Measurement, Vol. 105, 2001, pp. 55-65
11. The Open Group, “Application Response Measurement (Issue 3.0 - Java Binding)", Open Group Technical Specification, October 2001. Available at http://www.opengroup.org/tech/management/arm/
12. E. Christensen, F. Curbera, G. Meredith and S. Weerawarana, “Web Services Description Language (WSDL) 1.1", W3C Note, March 2001, Available at http://www.w3.org/TR/wsdl
13. S. Thatte, “Business Process Execution Language for Web Services version 1.0”, July 2002, Available at: http://www.ibm.com/developerworks/webservices/library/ws-bpel/
14. A. Keller, A. Koppel, K. Schopmeyer, “Measuring Application Response Times with the CIM Metrics Model”, 13th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 2002), Montreal Canada, 2002
15. W.P.Alexander, R.F.Berry, F.E.Levine, and R.J.Urquhart, “A Unifying Approach to Performance Analysis in the Java Environment”, IBM Systems Journal, Vol 39, No.1, 2000, pp.118-134
16. D.P. Spooner, S.A. Jarvis, J. Cao, S. Saini and G.R. Nudd, “Local Grid Scheduling Techniques using Performance Prediction", IEE Proceedings – Computers and Digital Techniques, 2003
17. J. Cao, S.A. Jarvis, S. Saini,, D.J. Kerbyson and G.R. Nudd, “ARMS: an Agent-based Resource Management System for Grid Computing", Scientific Programming, Special Issue on Grid Computing, Vol 10, No. 2, 2002, pp 135-148
18. J.D. Turner, D.A. Bacigalupo, S.A. Jarvis, D.N. Dillenberger and G.R. Nudd, “Application Response Measurement of Distributed Web Services”, International Journal of Computer Resource Measurement, Vol 108, 2002, pp. 45-55
19. J.D. Turner, D.A. Bacigalupo, S.A. Jarvis, and G.R. Nudd, “Using a Transaction Definition Language for the Automated ARMing of Web Services”", UK CMG Conference on Technology and Performance Evaluation of Enterprise-Wide Information Systems, Reading UK, 2002
20. IBM Web Services Toolkit. Available at http://www.alphaworks.ibm.com/tech/webservicestoolkit
21. D.A. Bacigalupo, J.D. Turner, S.A. Jarvis, and G.R. Nudd, “A Dynamic Performance Prediction Framework for e-business Applications”, Invited Paper at SCI2003 7th World Multiconference on Systemics, Cybernetics and Informatics, Florida USA, 2003
22. M. Endrel, IBM WebSphere V4.0 Advanced Edition Handbook, IBM International Technical Support Organisation Pub., 2002. Available at: http://www.redbooks.ibm.com/
23. IBM Websphere Performance Sample: Trade2. Available at http://www.ibm.com/software/webservers/appserv/wpbs_download.html