George T. Heineman Ivica Crnkovic
Heinz W. Schmidt Judith A. Stafford
Clemens Szyperski Kurt Wallnau (Eds.)

LNCS 3489

Component-Based Software Engineering

8th International Symposium, CBSE 2005
St. Louis, MO, USA, May 2005
Proceedings

3

Springer

TP311.5-53
C834
2005

George T. Heineman Ivica Crnkovic
Heinz W. Schmidt Judith A. Stafford
Clemens Szyperski Kurt Wallnau (Eds.)

Component-Based Software Engineering

8th International Symposium, CBSE 2005
St. Louis, MO, USA, May 14-15, 2005
Proceedings

Springer

Volume Editors

George T. Heineman
WPI, Department of Computer Science
100 Institute Road, Worcester, MA 01609, USA
E-mail: heineman@cs.wpi.edu

Ivica Crnkovic
Mälardalen University, Department of Computer Science and Engineering
Box 883, 72123 Västerås, Sweden
E-mail: ivica.crnkovic@mdh.se

Heinz W. Schmidt
Monash University, School of Computer Science and Software Engineering
Wellington Road, Clayton VIC 3800 , Australia
E-mail: Heinz.Schmidt@csse.monash.edu.au

Judith A. Stafford
Tufts University, Department of Computer Science
161 College Avenue, Medford, MA 02155, USA
E-mail: jas@cs.tufts.edu

Clemens Szyperski
Microsoft
One Microsoft Way, Redmond, WA 98053, USA
E-mail: cszypers@microsoft.com

Kurt Wallnau
Carnegie Mellon University, Software Engineering Institute
Pittsburgh, Pennsylvania 15213-3890, USA
E-mail: kcw@sei.cmu.edu

Library of Congress Control Number: Applied for

CR Subject Classification (1998): D.2, D.1.5, D.3, F.3.1

ISSN 0302-9743
ISBN-10 3-540-25877-9 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-25877-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Boller Mediendesign
Printed on acid-free paper SPIN: 11424529 06/3142 5 4 3 2 1 0

Preface

On behalf of the Organizing Committee I am pleased to present the proceedings of the 2005 Symposium on Component-Based Software Engineering (CBSE). CBSE is concerned with the development of software-intensive systems from reusable parts (components), the development of reusable parts, and system maintenance and improvement by means of component replacement and customization. CBSE 2005, "Software Components at Work," was the eighth in a series of events that promote a science and technology foundation for achieving predictable quality in software systems through the use of software component technology and its associated software engineering practices.

We were fortunate to have a dedicated Program Committee comprised of 30 internationally recognized researchers and industrial practitioners. We received 91 submissions and each paper was reviewed by at least three Program Committee members (four for papers with an author on the Program Committee). The entire reviewing process was supported by CyberChairPro, the Web-based paper submission and review system developed and supported by Richard van de Stadt of Borbala Online Conference Services. After a two-day virtual Program Committee meeting, 21 submissions were accepted as long papers and 2 submissions were accepted as short papers.

We are grateful for the assistance provided by the organizers of the ICSE conference, in particular the General Chair, Gruia-Catalin Roman, and the Workshops and Co-located Events Co-chair André van der Hoek. We also wish to thank the ACM Special Interest Group on Software Engineering (SIGSOFT) for their sponsorship of CBSE 2005. The proceedings you now hold were published by Springer and we are grateful for their support. Finally, we must thank the many authors who contributed the high-quality papers contained within these proceedings. As the international community of CBSE researchers and practitioners continues to grow, we expect the CBSE Symposium series to similarly attract widespread interest and participation.

March 2005
Worcester, MA
USA

George T. Heineman

Organization

CBSE 2005 was sponsored by the Association for Computing Machinery (ACM) Special Interest Group in Software (SIGSOFT). CBSE 2005 was a co-located event with the 27th International Conference on Software Engineering (ICSE 2005).

Organizing Committee

Program Chair	George T. Heineman (WPI, USA)
Steering Committee	Ivica Crnkovic (Mälardalen University, Sweden) Heinz W. Schmidt (Monash University, Australia) Judith A. Stafford (Tufts University, USA) Clemens Szyperski (Microsoft Research, USA) Kurt Wallnau (Software Engineering Institute, USA)

Program Committee

Luca de Alfaro	University of California, Santa Cruz, USA
Rob Armstrong	Sandia National Laboratories, USA
Uwe Aßmann	Dresden University of Technology, Germany
Jakob Axelsson	Volvo Car Corporation, Sweden
Mike Barnett	Microsoft Research, USA
Judith Bishop	University of Pretoria, South Africa
Jan Bosch	Nokia Research Center, Finland
Michel Chaudron	University Eindhoven, The Netherlands
Ivica Crnkovic	Mälardalen University, Sweden
Susan Eisenbach	Imperial College London, UK
Wolfgang Emmerich	University College London, UK
Dimitra Giannakopoulou	NASA Ames, USA
Richard Hall	LSR-IMAG, France
Dick Hamlet	Portland State University, USA
George T. Heineman	WPI, USA
Tom Henzinger	EPFL, Switzerland and UC Berkeley, USA
Paola Inverardi	University of L'Aquila, Italy
Bengt Jonsson	Uppsala University, Sweden
Magnus Larsson	ABB, Sweden
Kung-Kiu Lau	University of Manchester, UK
Nenad Medvidovic	University of Southern California, USA
Rob van Ommering	Philips Research, The Netherlands

Program Committee (cont.)

Otto Preiss	ABB Corporate Research Centers, Switzerland
Ralf Reussner	University of Oldenburg, Germany
Douglas Schmidt	Vanderbilt University, USA
Heinz W. Schmidt	Monash University, Australia
Jean-Guy Schneider	Swinburne University of Technology, Australia
Judith A. Stafford	Tufts University, USA
Kurt Wallnau	Software Engineering Institute, USA
Dave Wile	Teknowledge, Corp., USA

Co-reviewers

Eddie Aftandilian
Mikael Åkerholm
Somo Banerjee
Steffen Becker
Dirk Beyer
Egor Bondarev
Ivor Bosloper
Guillaume Brat
Reinder J. Bril
Arindam Chakrabarti
Robert Chatley
Sybren Deelstra
Viktoria Firus
Kathi Fisler
Eelke Folmer
Johan Fredriksson
Esther Gelle
Falk Hartmann
Mugurel T. Ionita
Vladimir Jakobac
Anton Jansen
Xiaohong Jin
Merijn de Jonge
Hugo Jonker
Thomas E. Koch
Emanuel Kolb
Sten Löcher
Rikard Land
Ling Ling
Markus Lumpe
Frank Lüders
Wolfgang Mahnke
Sam Malek
Antinisca Di Marco
Chris Mattmann
Hailiang Mei
Raffaela Mirandola
Johan Muskens
Martin Naedele
Ioannis Ntalamagkas
Owen O'Malley
Fernando C. Osorio
Joachim Parrow
Corina Pasareanu
Paul Pettersson
Roshanak Roshandel
Chris Sadler
Johanneke Siljee
Marco Sinnema
James Skene
Antony Tang
Faris M. Taweel
Perla Velasco Elizondo
Björn Victor
Erik de Vink
Lucian Voinea
Anders Wall
Zheng Wang
Wang Yi
Yang Yu

Previous CBSE Workshops and Symposia

7th International Symposium on CBSE, Lecture Notes in Computer Science, Vol. 3054, Crnkovic, I.; Stafford, J.A.; Schmidt, H.W.; Wallnau, K. (Eds.), Springer, Edinburgh, UK (2004)

6th ICSE Workshop on CBSE: Automated Reasoning and Prediction `http://www.sei.cmu.edu/pacc/CBSE6`. Portland, Oregon (2003)

Previous CBSE Workshops and Symposia (cont.)

5th ICSE Workshop on CBSE: Benchmarks for Predictable Assembly
`http://www.sei.cmu.edu/pacc/CBSE5`. Orlando, Florida (2002)

4th ICSE Workshop on CBSE: Component Certification and System Prediction. Software Engineering Notes, 26(10), November 2001. ACM SIGSOFT Author(s): Crnkovic, I.; Schmidt, H.; Stafford, J.; Wallnau, K. (Eds.)
`http://www.sei.cmu.edu/pacc/CBSE4-Proceedings.html`. Toronto, Canada, (2001)

3rd ICSE Workshop on CBSE: Reflection in Practice
`http://www.sei.cmu.edu/pacc/cbse2000`. Limerick, Ireland (2000)

2nd ICSE Workshop on CBSE: Developing a Handbook for CBSE
`http://www.sei.cmu.edu/cbs/icse99`. Los Angeles, California (1999)

1st Workshop on CBSE
`http://www.sei.cmu.edu/pacc/icse98`. Tokyo, Japan (1998)

Table of Contents

Prediction, Analysis and Monitoring of System Architecture

Performance Prediction of J2EE Applications Using Messaging Protocols 1
Yan Liu, Ian Gorton

EJBMemProf – A Memory Profiling Framework for Enterprise JavaBeans 17
Marcus Meyerhöfer, Bernhard Volz

Model-Driven Safety Evaluation with State-Event-Based Component Failure Annotations 33
Lars Grunske, Bernhard Kaiser, Yiannis Papadopoulos

Optimizing Resource Usage in Component-Based Real-Time Systems 49
Johan Fredriksson, Kristian Sandström, Mikael Åkerholm

Evaluating Performance Attributes of Layered Software Architecture 66
Vibhu Saujanya Sharma, Pankaj Jalote, Kishor S. Trivedi

Component-Level Dataflow Analysis 82
Atanas Rountev

Architecture and Design of Component-Based Systems

Exogenous Connectors for Software Components 90
Kung-Kiu Lau, Perla Velasco Elizondo, Zheng Wang

Qinna, a Component-Based QoS Architecture 107
Jean-Charles Tournier, Jean-Philippe Babau, Vincent Olive

Architecture Based Deployment of Large-Scale Component Based Systems: The Tool and Principles 123
Ling Lan, Gang Huang, Liya Ma, Meng Wang, Hong Mei, Long Zhang, Ying Chen

Component-Based Open Middleware Supporting Aspect-Oriented Software Composition 139
Bert Lagaisse, Wouter Joosen

An Empirical Study on the Specification and Selection of Components Using Fuzzy Logic 155
Kendra Cooper, João W. Cangussu, Rong Lin, Ganesan Sankaranarayanan, Ragouramane Soundararadjane, Eric Wong

Finding a Needle in the Haystack: A Technique for Ranking Matches Between Components 171
Naiyana Tansalarak, Kajal Claypool

Extra-Functional System Properties of Components and Component-Based Systems

A Contracting System for Hierarchical Components 187
Philippe Collet, Roger Rousseau, Thierry Coupaye, Nicolas Rivierre

Tailored Responsibility Within Component-Based Systems 203
Elke Franz, Ute Wappler

Efficient Upgrading in a Purely Functional Component Deployment Model 219
Eelco Dolstra

Real-Time Scheduling Techniques for Implementation Synthesis from Component-Based Software Models 235
Zonghua Gu, Zhimin He

A Component-Oriented Model for the Design of Safe Multi-threaded Applications 251
Reimer Behrends, R.E. Kurt Stirewalt, Laura K. Dillon

TESTOR: Deriving Test Sequences from Model-Based Specifications 267
Patrizio Pelliccione, Henry Muccini, Antonio Bucchiarone, Fabrizio Facchini

Components at Work

A CCA-compliant Nuclear Power Plant Simulator Kernel 283
Manuel Díaz, Daniel Garrido, Sergio Romero, Bartolomé Rubio, Enrique Soler, José M. Troya

Experience with Component-Based Development of a Telecommunication Service 298
Gregory W. Bond, Eric Cheung, Healfdene H. Goguen, Karrie J. Hanson, Don Henderson, Gerald M. Karam, K. Hal Purdy, Thomas M. Smith, Pamela Zave

Reusable Dialog Component Framework for Rapid Voice Application Development 306
Rahul P. Akolkar, Tanveer Faruquie, Juan Huerta, Pankaj Kankar, Nitendra Rajput, T.V. Raman, Raghavendra U. Udupa, Abhishek Verma

Unlocking the Grid 322
Chris A. Mattmann, Nenad Medvidovic, Paul M. Ramirez, Vladimir Jakobac

Experience Report: Design and Implementation of a Component-Based Protection Architecture for ASP.NET Web Services 337
Konstantin Beznosov

Concept Index 353

Author Index 357

Performance Prediction of J2EE Applications Using Messaging Protocols

Yan Liu, Ian Gorton

National ICT Australia (NICTA),
1430, NSW, Australia
{jenny.liu, ian.gorton}@nicta.com.au

Abstract. Predicting the performance of component-based applications is difficult due to the complexity of the underlying component technology. This problem is exacerbated when a messaging protocol is introduced to create a loosely coupled software architecture. Messaging uses asynchronous communication, and must address quality of service issues such as message persistence and flow control. In this paper, we present an approach to predicting the performance of Java 2 Enterprise Edition (J2EE) applications using messaging services. The prediction is done during application design, without access to the application implementation. This is achieved by modeling the interactions among J2EE and messaging components using queuing network models, calibrating the performance model with architecture attributes associated with these components, and populating the model parameters using a lightweight, application-independent benchmark. Benchmarking avoids the need for prototype testing in order to obtain the value of model parameters, and thus reduces the performance prediction effort. A case study is carried out to predict the performance of a J2EE application with asynchronous communication. Analysis of the resulting predictions shows the error is within 15%.

1 Introduction

Many software component models utilize synchronous communication protocols, such as Enterprise JavaBeans (EJB) based on RMI, and RPC-based CORBA or COM+ components. Synchronous communication dictates that the client process blocks until the response to its request arrives. More loosely coupled software architectures can be constructed using asynchronous invocations. These place an intermediary messaging service between the client and server, decoupling their execution. In addition, asynchronous invocations are desirable for applications with high performance and scalability requirements. For these reasons, component technologies have been integrated with messaging protocols to support the development of applications with asynchronous architectures.

Messaging services are implemented by message-oriented middleware (MOM), such as Microsoft MSMQ, IBM WebSphere MQ, CORBA Notification Services and Sun's JMS (Java Messaging Service). JMS is a Java interface specification, which provides a standard way for Java applications to access enterprise messaging infrastructure. MOM typically supports two forms of messaging: point-to-point (PTP) and publish/subscribe (Pub/Sub). In the PTP model, the message producer posts a message to a *queue*, and the message consumer retrieves the message from the queue. In the Pub/Sub model, a message producer publishes a message to a *topic*, and all consumers subscribing to the same topic retrieve a copy of the message. MOMs also define a set of reliability attributes for messaging, including non-persistent or persistent and non-transactional or transaction queues [18].

A component-based application using messaging protocols hence exploits an asynchronous, queue-based communication paradigm. It must also address additional architectural considerations such as the topology of component connections, message persistence and flow control. All of these factors can heavily influence the resulting application's performance [18].

However, the choice of application architecture must to be made early in the application life cycle, long before substantial coding takes place. Unwise decisions at design-time are often very difficult to alter, and could make it impossible to achieve the required performance level once the system has been delivered [5][6]. Consequently, the designer needs to be able to predict the performance of asynchronous components, working from an abstract design but without access to a complete implementation of the application.

Our previous work in [9] develops an approach to predicting the performance of only synchronous J2EE applications from design-level descriptions. The contribution of this paper is the extension of our approach to predict the performance of applications comprising both synchronous and asynchronous communications. This is achieved by modeling the component infrastructure that implements the messaging service. We then execute benchmarks to obtain values of model parameters associated with the performance characteristics of the underlying component infrastructure and the messaging service. We validate our approach through a case study, in which we compare predicted versus actual performance of an example application.

2 Related Work

Our previous work in [9] integrates analytical modeling and benchmark testing to predict the performance of J2EE applications using EJB components. A case study showed that without access the application source code, prediction can be accurate enough (prediction error is below 15%) to evaluate an architecture design. However, this work only addresses synchronous communication between components.

Performance modeling is a useful approach for performance analysis [16]. Traditional performance modeling techniques can be manually applied to applications based on Message-Oriented Middleware (MOM). [17] analyzes a multilayered queue network that models the communication between clients and servers via synchronous and asynchronous messages. [11] applies a layered QNM for business process integration middleware and compares the performance for both synchronous and asynchronous architectures. However, explicit values for performance parameters are required to solve these models, such as the CPU time used by each operation.

However, such performance parameters cannot be accurately estimated during an application design. A common practice therefore is to build a prototype and use this to obtain measures for the values of parameters in the model. For a complex application, this is expensive and time-consuming. Progress has been made to reduce the prototyping effort with tool support for automatic generation of test beds [1][3]. Although prototype testing can produce empirical evidence of the suitability of an architecture design, it is inherently inefficient in predicting performance as the application architecture inevitably evolves. Under change, the test bed has to be regenerated and redeployed, and the measurement has to be repeated for each change.

In related research towards software performance engineering, many approaches translate architecture designs mostly in United Modeling Language (UML) to analytical models, such as Queuing Network models [7], stochastic Petri nets [14] or stochastic process algebras [2]. In these approaches, the application workflow is presented in a sequence or state diagram, and a deployment diagram is used to describe the hardware and software resources, their topology and characteristics.

Importantly however, the component infrastructure and its performance properties are not explicitly modeled. These approaches therefore generally ignore or greatly simplify the details of the underlying component infrastructure performance. As a result, the models are rather inaccurate or non-representative. [8] developed a simulated model of CORBA middleware but the work is specific to threading structure of a CORBA server. Hence, little work has been done to develop an engineering approach to predict during design the runtime performance of messaging applications.

3 Major Performance Factors of J2EE Applications

J2EE includes several different component types, including EJB. EJB components act as servers and execute within a component container. A request to an EJB is passed through a method invocation chain implemented by the container and finally reaches the EJB method specified in the request. The invocation chain is used by the container to call security and transaction services that the EJB methods specify.

The container provides the hosting environment for EJBs and manages their lifecycle according to the context information of the request. The container also coordinates the interaction between EJBs and other J2EE services and facilities access to external data source connection pools. To improve performance and scalability, the container is multi-threaded and can service multiple simultaneous EJB requests. Multiple instances of threads, EJBs and database connections are pooled to provide efficient resource usage in the container. Incoming requests are queued and wait for a container thread if none are available from the fixed size thread pool.

Concurrent EJB requests experience contention at three points inside the container. These are during request dispatching to an EJB, during container processing and during access to external data sources. As a result, apart from the underlying hardware and software environment, the performance of a deployed J2EE application depends on a combination of the following factors:

- behavior of its application-specific EJB components and their interactions;
- particular implementation of the component infrastructure, or container;

- selected configuration settings for the container (e.g. thread pool size);
- attribute settings of both the application components (e.g. the persistence attribute of EJBs) and the infrastructure components (e.g. the transaction isolation level of the container);
- simultaneous request load experienced at a given time by the application [5].

Integrating a JMS messaging service with EJB components introduces further performance considerations. These include the topology of component connections, message persistence needs, and flow control. For instance, non-persistent messaging has better performance than persistent messaging [18]. However persistent messaging creates an application that is guaranteed not to lose messages, and hence is more reliable. For an architect, the ability to quantify this performance/reliability trade-off without building each solution is desirable, as is determination of the level of performance that the overall system provides under load.

4 The Performance Prediction Approach

A performance prediction approach for J2EE applications with messaging protocol needs to encompass the following three aspects. First, the performance model should explicitly represent the component container, the MOM service and their communication with application components. Second, the service time of a request depends on the container and MOM attributes. For example, in MOM-based applications, the setting of a messaging attribute is an architectural design decision and the effect on performance should be modeled as a function of the messaging attributes of interest. Third, an application-independent performance profile of the container and the MOM infrastructure is required. This is because the container and message server implementation and the operating system/hardware platform must be taken into account to be able to make accurate application performance predictions.

The relationship between the performance model and the component container performance profile for a selected architecture model are represented as a performance prediction framework in [9]. In this framework, a queueing network model (QNM) P models the component infrastructure by identifying the main components of the system, and noting where queuing delays occur.

An architect has several alternatives to fulfill the same functionality using EJB technology. For example, a server side EJB component can be made either stateless or stateful, simply by setting an attribute. Each architecture alternative impacts the service time of the component container. Therefore the component architecture model f^A is a function of the service time of the components participating in an architecture A. The output of f^A is the input to P.

Performance profiles are required to solve the parameter values of the performance model. They are obtained from benchmarking measurements. The benchmark application differs from an application prototype in that the business logic of the benchmark is much simpler than a prototype. The operations of the benchmark are designed simply to exercise performance sensitive elements of the underlying component container. The aim is to determine the performance profile of the container itself, and not to evaluate the overall application performance (the traditional aim of benchmarking).

By using a simple benchmark application, we can remove any unpredictability in performance due to application business logic.

The model is finally populated using the performance profile and used for performance prediction. This approach enables performance prediction during the design of software applications that are based on a specific component technology.

A comprehensive description of this approach can be found in [9][10]. It is designed to support the following use cases during performance engineering:

- Support efficient performance prediction under architecture changes where components are added or modified.
- Capacity planning of the system, such as predicting the average response time, throughput and resource utilization under the expected workload.
- Reveal performance bottlenecks by giving insight into possible flaws in architecture designs.

The requirements of this approach are:

- Ensuring a reasonable level of accuracy for performance prediction. According to [12] (page 116), prediction error within 30% is acceptable.
- Cost effective. The approach must be faster than prototyping and testing.

5 The Performance Model

A performance model should capture the component container behavior when processing a request from a client. For this reason, we focus on the behavior of the container in processing EJB method invocation requests. As EJB containers process multiple simultaneous requests, the threading model utilized must also be represented. The QNM in Fig. 1 models the main infrastructure components involved and their interactions.

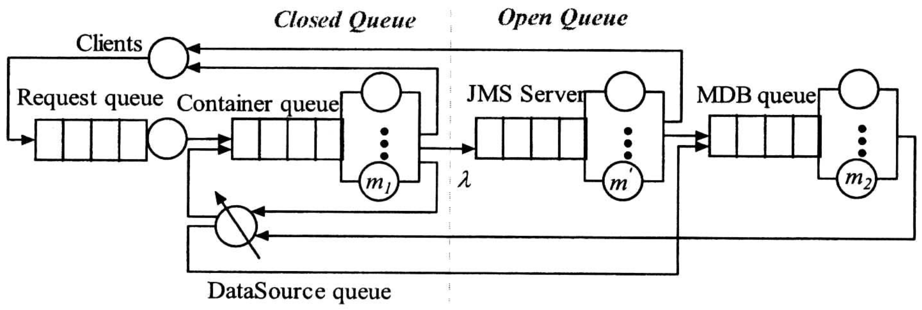

Fig. 1. The QNM model of a J2EE server with a JMS Queue

The model comprises two sub-networks, a closed and an open QNM. A closed QNM is appropriate for components using synchronous communication, as component containers employ a finite thread pool that effectively limits the maximum requests active in the server. An open QNM models asynchronous communication as a component container sends a message to a JMS queue, and the message is forwarded to a message driven bean (MDB) to process the business logic.

In the closed model, application clients represent the 'proxy clients[1]' (such as servlets in a web server) of the EJB container. Consequently, a client is considered as a delay resource and its service time equals the *thinking* time between two successive requests. A request to the EJB container is interpreted and dispatched to an active container thread by the request handler. The request handler is modeled as a single server queue with no-load dependency. It is annotated as *Request queue* in the QNM.

The container is multi-threaded, and therefore it is modeled as a multi-server *queue* with the thread capacity m_1 and no load dependency. It is annotated as *Container* in the QNM. The database clients are in fact the EJBs that handle the client request. Database access is therefore modeled as a delay server with load dependency. Its active database connections are denoted as k in the QNM, and the operation time at the database tier contributes to the service demand of the *DataSource* queue.

In the open model, asynchronous transactions are forwarded by the container to a queue managed by a JMS server. The JMS server is multi-threaded, and has a threshold for flow control to specify the maximum number of the messages pending in the JMS server. Assuming that the arrival rate of requests is a Poisson distribution with rate λ requests per second and the service time is exponential, we can model the JMS server as an $M/M/m'/W$ queue, where m' is the number of JMS server threads and W is its flow control threshold.

A message is subsequently removed from the queue by a MDB instance, which implements the business logic. MDBs are asynchronous message-handling façades for data access carried out in entity beans. MDB instances are also managed by the EJB container and are associated with a dedicated server thread pool. So the *MDB* queue is modeled as a load-independent multi-server queue.

The implementation of an EJB container and JMS server is complex and vendor specific. This makes it extremely difficult to develop a performance model that covers all the relevant implementation-dependent features, especially as the EJB container source code is not available. For this reason, our quantitative model only covers the major factors that impact the performance of applications, and ignores many other factors that are less performance sensitive. Specifically, we do not currently consider workloads that include large data transfers. As a result, the network traffic is ignored and the database contention level is reduced.

5.1 The Architecture Model

The task of solving the QNM in Fig. 1 involves obtaining the service demand of each queue. We need to calibrate the component container that will host the alternative designs in order to obtain the service demands of each request on each queue.

The service demand of the *Request* queue equals the service time of a request being accepted by the server and dispatched to the *Container* queue. It can thus be considered as a constant. The *Container, DataSource, JMS* and *MDB* queues are responsible for processing the business logic and this comprises the majority of the service demands on these queues.

[1] As opposed to clients driven by human interaction, proxy clients such as servlets continually handle requests that arrive at a web server.

Fig. 2 shows the state diagram for processing transactions in an EJB container. The container has a set of operations that a request must pass through, such as initializing a set of container management services, invoking the generated skeleton code for a bean instance, registering a transaction context with the transaction manager, finalizing a transaction and reclaiming resources.

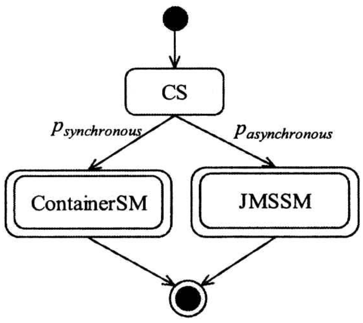

Fig. 2. Overall state diagram of the J2EE server

These container operations are identical for all requests, and the service time can again be considered constant, denoted as T_0. For convenience, these states as a whole are referred to as a composite, CS. Synchronous transactions are processed by the *Container* queue, modeled as a compound state machine *ContainerSM* with a probability $p_{synchronous}$, while asynchronous messages are posted to the JMS server, modeled as a compound state machine *JMSSM* with a probability $p_{asynchronous}$. *ContainerSM* and *JMSSM* are further decomposed into models of different component architectures. The service times of operations in *Container, DataSource, JMS* and *MDB* queue are modeled as f_c, f_d, f_j and f_m respectively.

From the above analysis, we know that f_c and f_d are determined by the component architecture in *ContainerSM* (e.g. optimizing data access to an entity bean by different cache management schemes). The comprehensive architecture models are developed in [9][10]. The models for container managed persistence of entity beans are listed below as an example:

$$f_c = hT_1 + (1-h)T_2 \quad (1)$$

$$f_d = T_{find} + T_{load} + pT_{store} \quad (2)$$

- h is the entity cache hit ratio;
- p is the ratio of operations updating entity data;
- T_1 is the service time for the container to access the entity data in its cache;
- T_2 is the service time of the container to load/store an entity bean instance from/to disk;
- T_{find} is the service time of identifying the entity instance by its primary key;
- T_{load} is the service time of loading data from the database into the entity bean cache;
- T_{store} is the service time of storing updates of an entity bean data.