RC 22091 (W0106-020) 13 June 2001 Computer Science

# IBM Research Report

## Layered Queueing Models for Enterprise JavaBean Applications

**Te-Kai Liu, Santhosh Kumaran, Zongwei Luo**
IBM Research Division
Thomas J. Watson Research Center
P. O. Box 704
Yorktown Heights, NY 10598

**Research Division**
**Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich**

# Layered Queueing Models for Enterprise JavaBean Applications

Te-Kai Liu, Santhosh Kumaran, and Zongwei Luo
*IBM T. J. Watson Research Center*
*Yorktown Heights, NY 10598*
*tekailiu@us.ibm.com*

## Abstract

*Traditional capacity sizing of enterprise systems relies on benchmarking a system using benchmark clients that generate a workload pattern similar to real-world workload. When new functions are added to the system or when the workload pattern changes, benchmarking has to be performed again. This is a costly and time-consuming approach for capacity planning. Layered queueing models have been used to study the performance of software systems. The approach is able to identify major performance parameters of software systems. Given a workload pattern, the models can be solved analytically to predict the system performance quickly.*

*This paper proposes a layered queueing model for predicting the performance of distributed enterprise applications built on Enterprise JavaBeans (EJB) technology. We show how such models can be applied for capacity sizing of distributed enterprise systems. We demonstrate this by using this methodology to predict the performance of a sample application built on an EJB-based business-to-business e-commerce platform. We compare deployment options and study the effect of different workload patterns on system capacity.*

## 1. Introduction

In just over two years since its introduction, Enterprise JavaBeans (EJB) technology has gained significant acceptance among platform providers and enterprise development teams. This is primarily because the EJB server-side component model simplifies development of distributed object applications that are transactional, scalable, and portable. Enterprise JavaBeans servers provide automatic support for middleware services such as transactions, security, database connectivity, and thus reduce the complexity of application development.

Despite its growing popularity, performance of EJB-based applications has remained a concern. This is especially true for large scale B2B e-Commerce applications such as Private Trading Exchanges and e-Marketplaces, where a company has to ensure that the Service Level Agreements (SLA) with its trading partners are being met. The growing use of this technology in many e-Commerce applications makes the performance of EJB-based systems an important issue to be considered.

Benchmarking is commonly used to measure the performance of software systems. But benchmarking can be done only after the systems are operational. The challenge of the system designers and integrators is to predict the performance of the system before they are fully built so that the right topology and design decisions can be made to ensure that service level agreements can be met. This can be achieved only by using reliable performance prediction models. This paper investigates the performance prediction of EJB-based applications using analytic models. In particular, we focus on the modeling of an EJB-based B2B e-Commerce application framework.

The rest of the paper is organized as follows. We begin with a brief overview of EJB technology and a description of the application framework. Next we introduce our modeling approach and describe the modeling of the framework-based applications using this approach. We discuss a methodology for calibrating these models. We show how this methodology can be applied for capacity sizing of applications built on the framework for a given topology and workload pattern.

## 2. EJB Overview

There are two types of Enterprise JavaBeans: Entity Beans and Session Beans. Entity beans model data objects; these objects usually correspond to persistent records in a database. Session beans model business objects; they typically work with entity beans or other resources to implement the business logic. The EJB technology differentiates the remote interface of a bean from its home interface. The remote interface defines the exposed business methods of an EJB. The home interface defines the life cycle methods of the bean typically used by the EJB container for locating, creating, and removing the beans. A good introduction to the EJB technology can be found in [6].

## 3. Application Framework for B2B e-Commerce

The goal of the framework is to enable rapid development of B2B e-Commerce applications. The framework achieves this by creating a brokering layer between the B2B clients and the enterprise systems. This layer provisions service requests from the clients by mapping them to appropriate enterprise systems such as workflow engines, legacy applications, ISV applications, and business objects. Major components of the framework are shown in Figure 1. We describe the framework using the fundamental design patterns [8] on which it is based.

The BFMAdmin component is designed based on the Façade design pattern [8]. Its purpose is to serve as an entry point for all client requests and route them to the appropriate brokering components. The Façade is implemented as a Session Bean.

The brokering components are called Adaptive Documents (ADocs). ADocs provide domain-specific and state-dependent service brokering. In the B2B e-Commerce area, examples of ADocs include Registration, Purchase Order, Request-For-

Quote, etc. ADocs are based on the State design pattern [8], as they demonstrate state-dependent behavior. An ADoc implementation consists of an ADoc Entity that captures the state information and an ADoc Controller that encapsulates the state-dependent ADoc behavior. The behavior is shared among all instances of an ADoc type. For example, all Purchase Orders exhibit the same behavior and we need only one controller for the Purchase Order ADoc. An ADoc Entity exists for each instance of an ADoc. Using the Purchase Order example, there is an instance of a Purchase Order ADoc Entity for each purchase order in the system. ADoc Entity is implemented using an Entity Bean. The ADoc Controller is described using an XML script, which BFMAdmin reads in and internalizes as Java objects.

The framework supports web transactions by means of ADoc behavior. When a client issues a web transaction, it reaches some ADoc in the system as a service request. As defined in the behavior of the ADoc, the ADoc controller executes a set of actions determined by the current state of the ADoc and the contents of the service request. These actions are implemented as a transaction. A service request may also result in changing the state of an ADoc. The fulfillment of a service request follows the Command design pattern. The participants in the command design pattern [8] and their roles are as follows:

i. Command
   A command declares an interface for executing an operation. Commands are invoked by the ADoc Controller.

ii. ConcreteCommand
   A ConcreteCommand defines a binding between a Receiver object and an action. It implements a specific operation by invoking the corresponding operations on Receiver.

iii. Receiver
   A Receiver knows how to perform the operations associated with a request. Any class may serve as a Receiver. Commonly used Receivers in B2B e-Commerce include workflow engines, enterprise applications, business objects, and trading partner gateways.

The commands are implemented as Java objects. The BFMAdmin initializes these objects based on the XML-based script files. The Receivers could be EJBs or Java objects in the local JVM. A Java object known as an Application Adapter can be used as a receiver to communicate with applications that support a messages-oriented API. An Entity Bean is used as a receiver to communicate with data objects. A Session Bean is used as a receiver to communicate with Business Objects. More details on the framework can be found in [9].

An application platform for B2B e-Commerce can be realized by augmenting the framework with a set of services beyond those provided by an EJB server. These include directory services, solution management services, and messaging services. More details on such a platform can be found in [9].

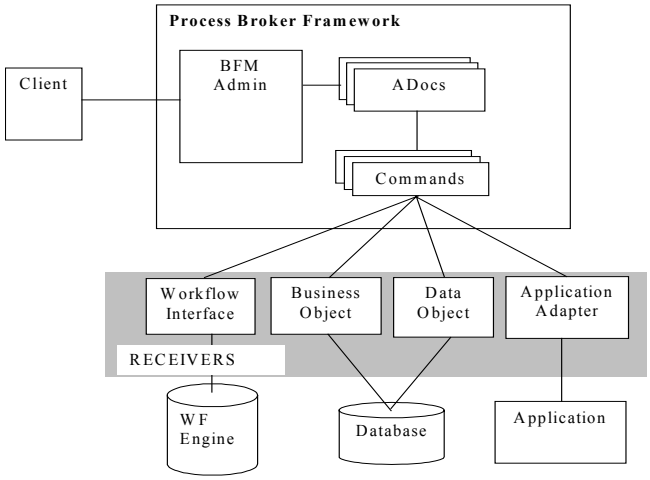


**Figure 1: Process Broker Framework**

## 4. The Modeling Approach

The layered queueing modeling (LQM) is chosen as the modeling approach in this paper. The LQM has been used to study the performance of distributed software systems [1], [2], [5]. It is able to identify major performance parameters of software components [3]. It can determine software queueing effects (requests queued at servers) and competition between applications. It can also detect software bottlenecks [4]. These observations have led us to LQM as the modeling mechanism for modeling EJB applications such as the B2B e-Commerce application framework described above.

In LQM, a server process is modeled as a "task" which has one or more entries corresponding to the major method calls that the server process exposes to its clients. We model an EJB by three layers of tasks. The task at the top layer has one or more entries, each corresponding to a business method of the EJB. The second layer has a task corresponding to a database management system (DBMS) which accepts database connection requests from the EJB, processes SQL statements, and retrieves/updates the data stored in the persistent store, i.e. a disk. The resource demands of the entries of the DBMS task could vary from one EJB method to another. So in general there will be an entry in the DBMS task for each method of the EJB being modeled. The third layer has a task corresponding to a disk subsystem that is modeled in LQM as a processor with one or more entries, each of which corresponds to a type of disk demand.

Figure 2 shows a three-layer model for an EJB. Each entry of the 3 tasks has a value corresponding to its CPU demand, except for the entries, volume1 and volume2, which correspond to different disk demands. Each arrow has a value associated with it, which is the average number of calls to the entry being pointed to.

Before a client can invoke a method call on the business methods of an EJB, it needs to get a reference to an EJBObject of the bean. For an entity bean, the client first does a lookup for the home interface of the bean via JNDI (Java Naming and Directory Interface) and then invokes the find method of the bean's home interface, which is typically a findByPrimaryKey()

method. For a session bean, a client invokes the create method of the bean's home interface before the business methods can be called. To take into account the overhead of getting a reference to an EJBObject of the bean, we add an additional entry to the first layer of the model. The entry name is "find" for an entity bean and "create" for a session bean. Note that this entry represents the overhead incurred by the EJB server. The overhead incurred by the client is not explicitly modeled. It is assumed to be part of the resource demands of the calling clients.

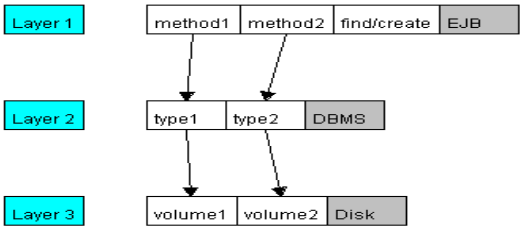


**Figure 2: LQM for an EJB**

Although not representing data stored in a database, session beans can make JDBC calls to access (i.e., read/update/delete) data in a database. In such cases, the model for a session bean is very similar to the model for entity beans. The difference is that session beans calls DBMS directly, whereas entity beans (specifically container managed persistence beans) rely on an EJB container to make the calls. For session beans that do not access databases, the model reduces to one layer only.

## 5. LQM for the Framework-Based Applications

In this section, we discuss how to form an LQM for applications built on the framework. We show a simple application with two business transactions.

In the first transaction, the value of an attribute is set in a backend data object in response to a service request from the client. An entity bean with container-managed persistence is used for implementing the data object.

The request first arrives at the façade (BFMAdmin), which gets the current state from the appropriate ADoc instance. Based on the current state and the business event specified in the service request (Event1 in this example), BFMAdmin executes the appropriate commands on the backend objects. In this example, only one command is executed, which is to set an attribute in a purchase order data object. The receiver for this command is an entity bean. The client passes the attribute value as a parameter in the service request. Figure 3 shows the collaboration diagram.

The second scenario involves fetching the value of an attribute by invoking a method on a business object. This scenario differs from the earlier one primarily by the type of receiver used. Here the receiver is a session bean that uses an entity bean for persisting the data. Figure 4 shows the collaboration diagram.

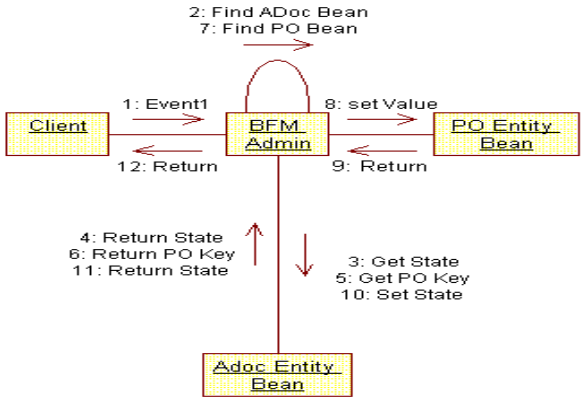


**Figure 3: Collaboration diagram for request: Event 1**

We use the collaboration diagram to develop an LQM to model the response time for provisioning these business events. Figure 5 shows the LQM for Event1. When BFMAdmin receives the event1 service request, it invokes the methods of the ADoc entity bean and PO entity bean according to the sequence shown in Figure 3. Since the ADoc bean and the PO bean are container managed persistence beans, methods of DBMS will be invoked by the container.

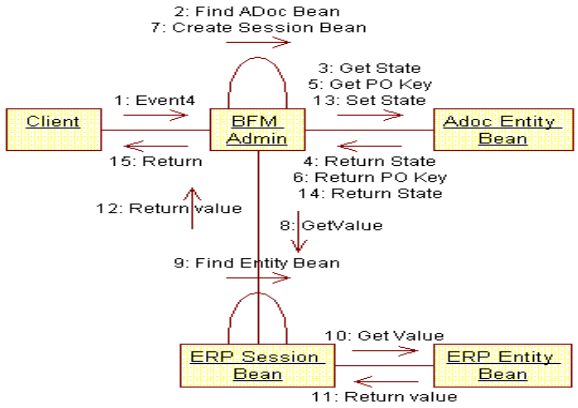


**Figure 4: Collaboration diagram for the request: Event2**

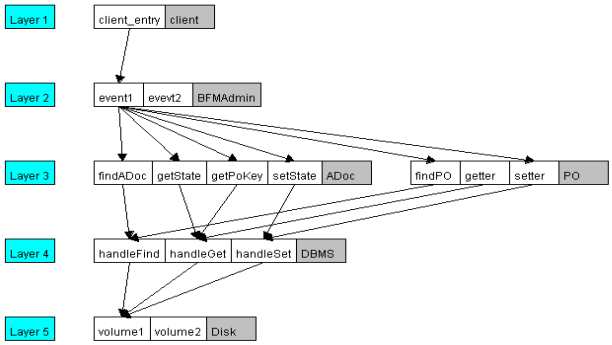


**Figure 5: LQM for Event1**

## 6. A Method for LQM Calibration

The LQM models need to be calibrated before it can be used for capacity sizing in which the impact of workload pattern on system capacity is evaluated during infrastructure planning prior to deployment. A calibrated LQM models can be used to compare EJB deployment options in scalability studies as well. We will discuss some of these applications in Section 7.

Calibration of the model translates to determining the values of the parameters for the tasks at each layer in the model. The first step in this process is profiling the application so that resource usage can be determined at method level for all EJBs in the application. Additionally, we benchmark the application to obtain response time for a service request versus the workload. The workload is defined as the number of concurrent requests handled by the system. In this paper, we are primarily focused on the steady state performance of EJB systems. To that end, we have developed a multi-threaded benchmark client that spawn a specified number of client threads to drive the EJB application according to the scenarios specified in the collaboration diagrams. Each thread submits a service request of the type specified in the scenario and wait for a response. As soon as a response arrives, the thread will promptly submit another service request with zero think time to ensure that workload of the EJB server is constant throughout the measurement period.

The values of the model parameters are selected based on the profiling data and the benchmark data. This is an iterative process since profiling data will give only a range of values for the model parameters. The selection of a value within this range is accomplished by solving the LQM analytically using an initial set of values, matching the predicted response time against the benchmark data, and repeating this process till convergence.

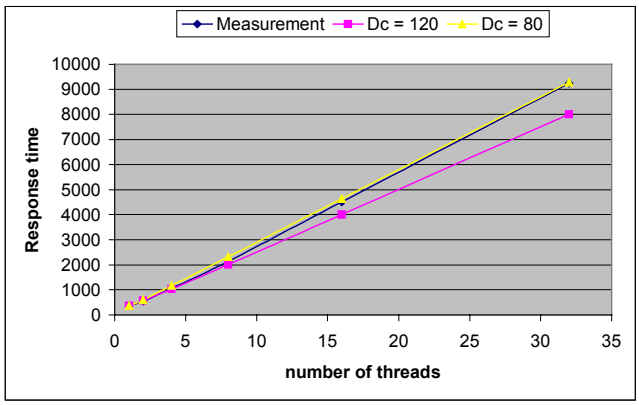


**Figure 6: Steady state response time (msec) for Event1**

Figure 6 shows observed and predicted response time versus workload for Event1, where *D*c is the total CPU demand of the event1 service request. Figure 6 shows that *D*c =80 gives a better match to the measurement data. The response time shows a linearly growing trend as the number of threads increases. This is due to the fully utilized disk resource which throttles the throughput of the system. The utilization of the disk and CPU is given in Figure 7. The heavy utilization of the disk is in part due to the internal transaction logging of the BFMAdmin, which is a functional requirement of B2B e-Commerce applications.

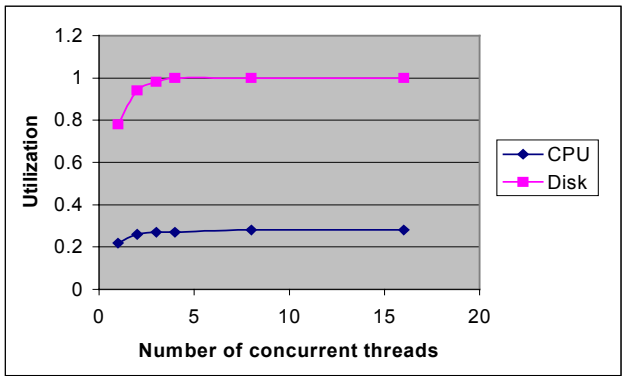


**Figure 7. Utilization of Disk and CPU.**

The linear trend, observed in Figure 6, is due to the zero think time assumption made in the client model. As the think time increases, we start to see the exponential growth of the response time as the number of threads increases.

## 7. Applications of the Model

In this section, we show two numerical examples to illustrate applications of the performance model. First we consider a capacity-sizing scenario. Figure 8 shows a LQM for the EJB application described in Section 5.

In this example, all of the beans and the DBMS are deployed on the same machine, which has a single processor and a single disk. The workload of the system is modeled by $n$ concurrent clients with $np$ of them submitting the Event1 service request and $n(1-p)$ of them submitting the Event2 service request, where $p$ is between 0 and 1. In other words, the variable $n$ models the workload intensity and the variable $p$ models the workload pattern. A client thread of type 1 will invoke the Event1 service request and wait for a response. When a client thread receives a response, it will make another call after $t$ second of think time.

The resource demands used in the examples are summarized as follows. All demands are in seconds.

| | |
|---|---|
| volume1 | 0.005 |
| handleFind | 0.01 |
| handleGet | 0.01 |
| handleSet | 0.01 |
| Find | 0.01 |
| Create | 0.02 |
| Getter | 0.001 |
| Setter | 0.001 |
| Event1 | 0.02 |
| Event2 | 0.02 |

getValues of the ERP session bean will call getValue of the ERP entity bean 10 times for its 10 attributes.

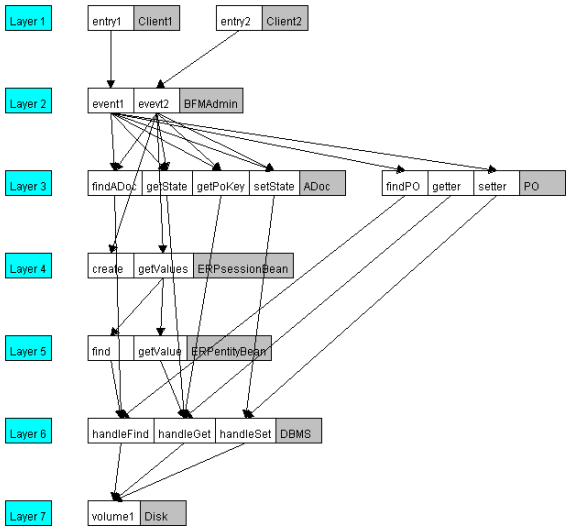


**Figure 8. LQM for the example EJB application.**

The performance metric is the average response time $T$, which is defined to be the weighted sum of the response times, $T1$ and $T2$, of the 2 types of service requests given by $T = p*T1 + (1-p)*T2$.

Figure 9 shows that the average response time versus the number of concurrent clients for $p = 0.5$ and $p = 0.3$. Since the type-2 requests demand more resources, the average response time is longer when the percentage of type-2 clients is higher (i.e., $p$ being smaller). The system capacity is defined as the maximum number of concurrent clients the system can support while keeping the average response time less than a specified value. From Figure 9 we see that given the maximum tolerable average response time equal to 1 sec, the system capacity is 40 and 50 for $p = 0.3$ and 0.5, respectively. This shows that the system capacity depends on the workload characteristics, currently modeled by the variable $p$.

In the second example, we assume that the same EJB application is deployed on two machines of the same processing speed. There are several configurations possible. In this example, we have the BFMAdmin and ADoc entity bean deployed on one machine while the PO entity bean, ERP session bean and ERP entity bean deployed on the other machine. Both machines have a local DBMS installed. The average response time as a function of the number of concurrent clients is shown in Figure 9. Due to the increased processing power, we see the capacity of the system (i.e. 90 for $p = 0.5$) is improved but less than double as expected.

## 8. Conclusion

We have developed a methodology based on layered queueing models for capacity sizing of Enterprise JavaBean applications. We demonstrated its use in capacity sizing of EJB applications and evaluating the effect of workload characteristics on system capacity. Future work includes improving calibration techniques, extending the models to represent end-to-end B2B e-commerce systems, and using the model to analyze and streamline deployment topologies.

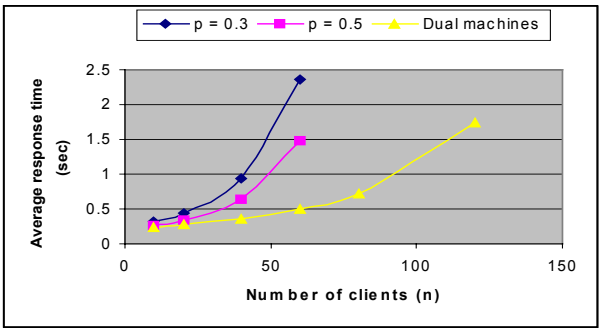


**Figure 9. Effect of workload pattern and deployment option on system capacity.**

## 9. Acknowledgement

The authors would like to thank Prof. C. M. Woodside for providing the LQNS tool [7], which was used for performance prediction in this paper.

## 10. References

[1] L. G. Williams and C. U. Smith, "Performance Evaluation of Software Architecture", *Proceedings of Workshop on Software Performance*, 1998.

[2] C. M. Woodside et al., "The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software", *IEEE Trans. On Computer*, vol. 44, no. 1, pp. 20-34, 1995.

[3] J. Dilley et al., "Web Server Performance Measurement and Modeling Techniques", *Performance Evaluation*, vol. 33, pp. 5-26, 1998.

[4] J. E. Neilson et al., "Software Bottleneck in Client-Server Systems and Rendezvous Networks", *IEEE Trans on Software Engineering*, vol. 21, no. 9, pp. 776-782, 1995.

[5] M. Woodside, "Software Performance Evaluation by Models", *Performance Evaluation*, LNCS 1769, pp. 283-304, 2000.

[6] R. Monson-Haefel, *Enterprise JavaBeans*, second edition, O'Reilly, 2000.

[7] R. G. Franks et al., "A Toolset for for Performance Engineering and Software Design of Client Server Systems," *Performance Evaluation*, vol. 24, no. 1-2, pp. 117-135, 1995.

[8] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns – Elements of Reusable Object Oriented Software*, Addison-Wesley Publishing Company, NY, 1995.

[9] K. Bhaskaran, J-Y Chung, R. Das, T. Heath, S. Kumaran, P. Nandi, "An e-Business Integration & Collaboration Platform for B2B e-Commerce," *Proceedings of the Third International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS 2001),* IEEE Computer Society Press, June 2001.