Completing UML Model of Component-Based System with Middleware for Performance Evaluation

Yong Zhang, Ningjiang Chen, Jun Wei, Tao Huang

Institute of Software, Chinese Academy of Sciences
{yzhang, river, wj, tao}@otcaix.iscas.ac.cn

Abstract. To free analysts from knowing about the internal details of middleware when evaluating the performance of component-based system (CBS), this paper proposes a framework to automatically integrate middleware component interactions and their performance attributes to application Unified Model Language (UML) model. The framework defines a general sub-model template library of middleware, a middleware usage description file, and an approach to compositing application-specific sub-model instances and application UML models. The process is illustrated by a case study.

1 Introduction

Performance effect of software architectural decision can be evaluated at an early phase by constructing and analyzing quantitative performance model, which capture the interaction between the main components of the system as well as the performance attributes of the component themselves. Some approaches to deriving performance model from architecture description have been proposed [1] [2].

A range of component-based server-side technologies, such as Enterprise Java Beans, CORBA, and COM+/.NET, support the design and deployment of application components in a component container environment, which help to CBS faster development cycles, decreased effort, and greater software reuse. The application component's behavior is a combination of the application-specific code and the underlying container services it utilizes, which will obviously impact the architecture and the performance of component application. The container and supporting platform must be taken into account for accurate performance predictions [3] [4] [5].

At times, container environment (or middleware) is not included as a part of application architecture description, and its performance information is missing. Some works to address this question have already been undertaken [6]-[10]. Using these approaches require analyst know detailed knowledge of middleware internals and modeling language itself, which decreases the ease of use of modeling, then hinders successful application of early performance analysis.

From performance evaluation perspective, this paper proposes a framework to automatically integrate middleware information to application UML models. The resulting UML models can represent structure and behaviors of both application and middleware, from which performance model including the impact of middleware can

be derived by some existing methods. So, performance analysts do not have to know about middleware internal details, when evaluating the performance of component-based system hosted by middleware. The process is illustrated by a case study based on EJB container middleware.

The rest of this paper is organized as follows. In section 2, we briefly describe the structure of the framework. The details of proposed framework and techniques to integrate middleware performance information to UML models are presented in Section 3. The proposed approach is demonstrated by a case study in Section 4. We introduce some related work in Section 5 and conclude with a summary in Section 6.

2 Structure of Framework

To reflect the architectural changes incurred by using middleware, as well as the impact on the overall system performance, this paper propose a framework to integrate middleware information to application UML models, as shown in Fig.1.

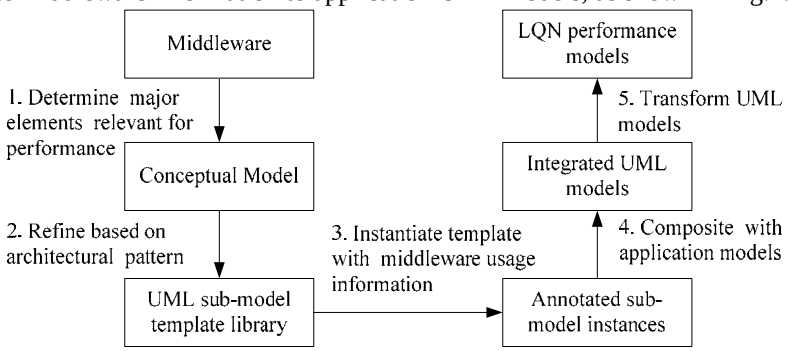

Fig. 1. Structure of framework integrating middleware performance information

The foundation of framework is a sub-model template library, which includes core middleware elements impacting performance. This helps designer to rapidly model middleware without knowing its internal details. Template library is built through architectural pattern-based refinement.

In the model of component application, the information about communication patterns, component container, its configuration setting, and middleware services to invocate, are missing, whereas these are major factors affecting performance. Involved components, their interaction relations, and relative performance properties need to be complemented. We will use UML Activity diagram to model the behaviors of middleware components, and UML Profile for Schedulability, Performance and Time (SPT Profile) as annotation to capture its performance requirements [11].

The performance impact of middleware is application-specific, which relates with concrete usage of middleware. In our work, a XML-based middleware usage description file is used to provide necessary middleware usage information, including aspects of functionality and performance. With middleware usage information, the sub-model template can be instantiated.

Here gives an approach to compositing the sub-models instances of middleware and application UML models. The resulting UML models include architecture and performance impact of middleware, from which performance model can be derived by using existing methods. The performance model used in this work is Layered Queueing Network (LQN) model [12], just one of several possible target formalisms.

3 Model Integration Techniques

The conceptual architecture model of container middleware can be described as in Fig.2. The processing of interaction between distributed components can be divided into two phases: components communication happening outside of the container, and the processing inside the container.

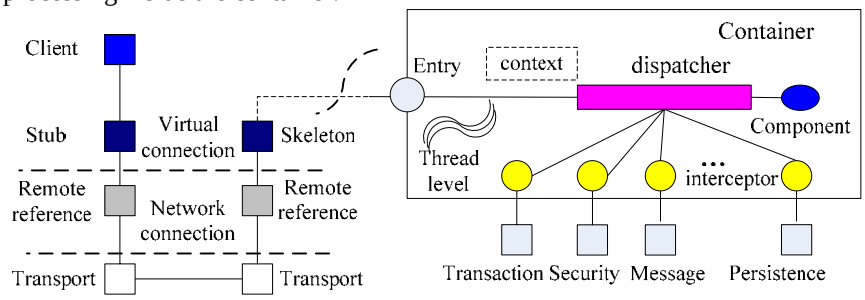

Fig. 2. Conceptual architecture model of Container middleware

3.1 Behavior Specifications and its Performance Annotation

Refinement of Communication. Distributed component communication generally bases on client-proxy-server pattern, which addresses the distribution and location transparency. During communication, client-side and server-side components (like stub, remote reference, skeleton in Fig.2) will perform some additional operations on the request and the response, such as marshaling and unmarshaling, to transform the data (e.g., parameter values) from the native format to a language independent wire format and back. These operations will incur performance overhead.

Communication process can be modeled at different abstract level with different internal details. More detailed models which can reflect the exact software architecture of the middleware, more accurate performance estimates can be got from it. At the same time, the system model will be more complex. Considering our performance modeling goals, we will refine it to functionality level, in particular, the virtual connection layer, showing how it interacts with the application system.

To enable users to capture time and performance requirements, SPT profile extends UML by providing stereotypes and tagged values to represent performance requirements, the resources used by the system and some behavior parameters [11]. Here, middleware components can be stereotyped as `<<PAresource>>`, and key

actions impacting performance can be stereotyped as <<PAstep>>, which demand can be tagged with PAdemand.

Interaction details of component communication are illustrated in Fig.3. (Here only synchronous call is illustrated with an example SPT profile annotation). At the same time, UML Collaboration representing high level view is also illustrated.

In addition, before sending the first invocation, client component needs to get the reference and local stub of remote server component (e.g., by Naming Service). Here, a *stub_init* action of stub models this operation.

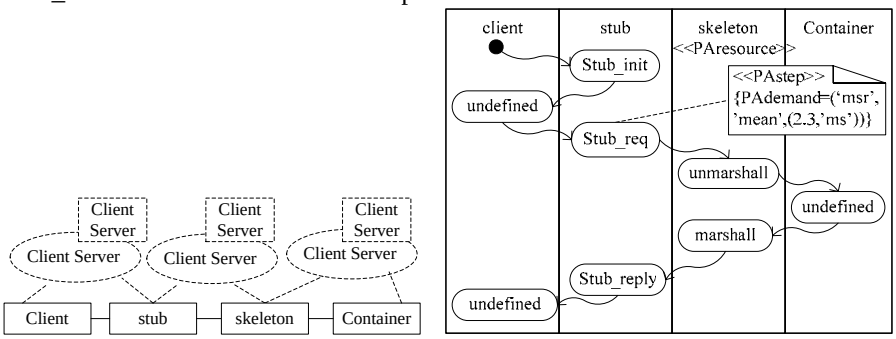

Fig. 3. Sub-model Template for Communication

Refinement of Container. Container must be flexible enough to integrate, manage, reuse, and extend middleware services. So, pattern similar to chains of interceptor (or responsibilities of chain) is generally employed, which enables dynamically adding middleware services to component system [13].

When the request arrives at server side container, container will create invocation context for request, providing operation information for accessing resource, security, the current transaction, or server component instance-specific information. Invocation context will be passed through chains of interceptor, triggering related interceptors (middleware service) in turn. Different middleware services serve the request concurrently under the control of different processes/threads. Triggered various middleware services all will affect the performance. At last, request will be sent to component business method. After processing, response result will be sent back to container entry, which continues sending back to client.

Container implements concurrency mechanisms so that multiple instances of components can be utilized simultaneously. Configuration settings of container (e.g., thread level) will impact performance. In this work, we use *instanceHandler* component modeling instance processing; and configuration setting can be represented with PAcapacity tag of SPT profile.

The collaboration relation and interaction sequence of components in container are represented in Fig.4. Each kind of middleware service will be abstracted as a service component (representing as placeholders *service_1… service_n* in Fig.4), using one action modeling the service behavior, instead of modeling its internals.

Middleware service placeholders (*service_1… service_n*) can be instantialized according to specific middleware usage information. And, the performance requirements can be annotated with SPT stereotype and tagged value.

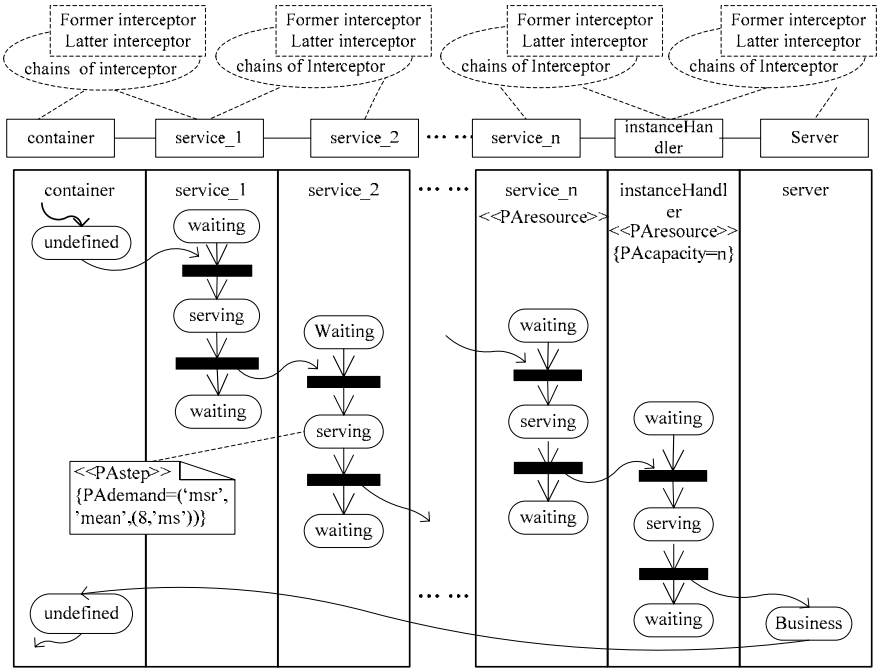

Fig. 4. Sub-model Template for Container

3.2 Middleware Usage Description

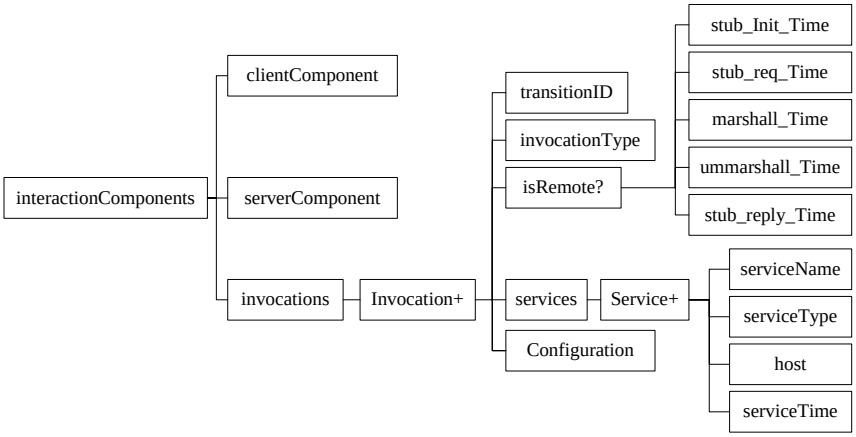

Fig. 5. XML Schema for middleware usage information

The performance impacts of middleware relate with specific application. It is necessary to provide relative middleware usage information, such as, which

invocations are remote, what middleware services will be used, and their executing demands, etc. In this paper, the middleware usage description will be provided in XML-based file, the Schema of which is shown in Fig.5.

The elements are declared according to interacting components that use middleware. For each invocation between client component and server component, there is a <invocation> declaration: element <transitionID> represents the transition referring to this call in UML activity diagram; element <invocationType> represents remote or local call; element <isRemote> specifies the service demand of several processing phases of remote invocation; <services> represents middleware services to use during invocation, in which the details are specified; Element <configuration> declares middleware configuration setting, such as thread level.

3.3 Model Composition

The composition process of application UML models and middleware can be described as follows.

1. Parsing XML-based middleware usage describing file;
2. For each pair of interacting Components {
   If invocation between them is remote then
   Instancing communication template with middleware usage description;
   Instancing the container template with middleware usage description;
   }
3. Combining Collaboration parts of above instances with *client-server* relation;
4. Combining Activity diagram parts of above instances;
5. Redirect the call between the interacting components;
   Deleting original collaboration relations in application UML Collaboration;
   Inserting combined UML Collaboration with client-server relation;
   Deleting original invocations in application UML Activity diagram;
   Inserting combined UML Activity Diagram of instances;
6. Changing original UML deployment diagram;
   Adding *stub* component to Client node;
   Adding middleware service components to Server node;

4 Case Study

As an illustration of proposed process, a case study was conducted, modeling the performance of an online store based on EJB container middleware. Fig.6 shows the UML models of this case. The scenario can be described as follows: *client* component makes a remote synchronous invocation to *CustomerControlBean* component to find the required customer information, in which need use middleware security service; and then updates email address of customer to *database*, in which need middleware transaction service supporting.

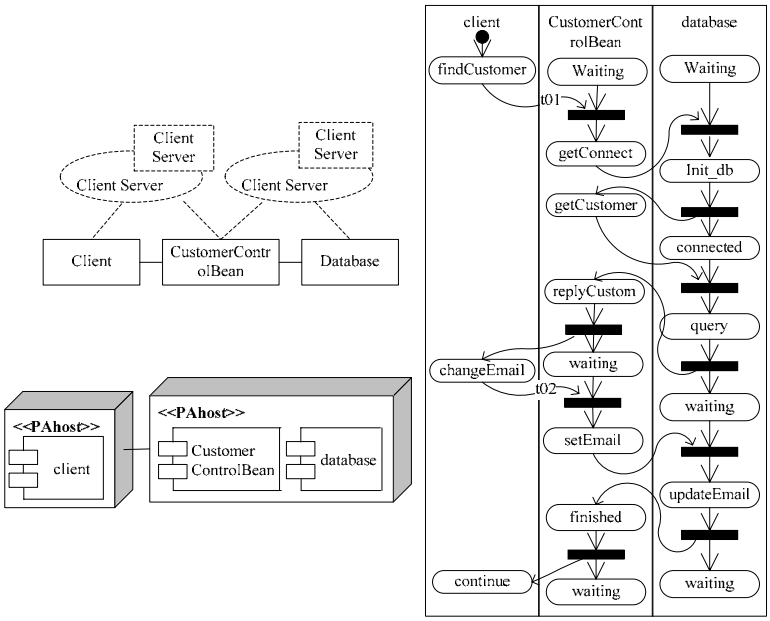

Fig. 6. UML diagram for case study

The major middleware usage information of this case is given in Table.1. In our work, we provide an input tool which can assist in formatting XML file according to schema in Fig.5. With the help of a profiling toolkit *OptimizeIt*, service demands were obtained from a prototype implementation of the case based on a J2EE Application Server called OnceASv2.0 [14].

Table 1. Middleware usage description information for case study

Configuration setting: thread level=30
Interacting Components: Client=client; Server=customerControlBean;
Invocation descriptions: transitionID=t01; invocation_Type=remote; init_Overhead=3.2ms; client_Overhead=2.1ms; server_Overhead=2.3ms;
Used services descriptions: service_Name=secService; service_Type=security; host=servernode; overhead=3.3ms;
Invocation descriptions: transitionID=t02; invocation_Type=remote; init_Overhead=0; client_Overhead=2.5ms; server_Overhead=2.8ms;
Used services descriptions: Service_Name=TXService; service_Type=Transaction; host=servernode; overhead=9.5ms;

According the composition process given in Section 3.3, the resulting UML models are shown in Fig.7-Fig.9. In this case, we take the response time index as illustration. The information is annotated by giving the actions the <<PAstep>> stereotype and specifying a tagged value PAdemand to represent execution time, which is provided in Table.1. For clarity, the performance information of application components is omitted in diagram.

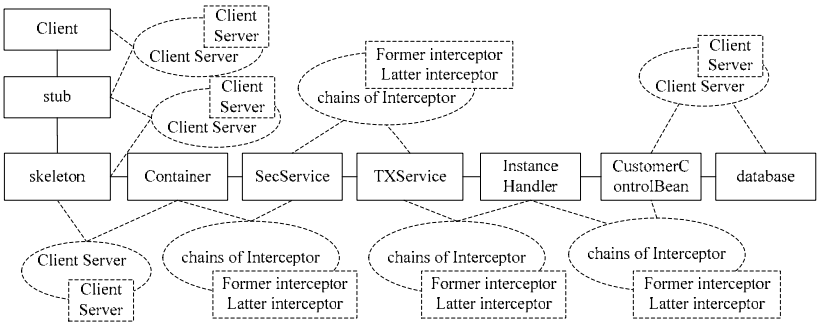

Fig. 7. Integrated high level view of the case

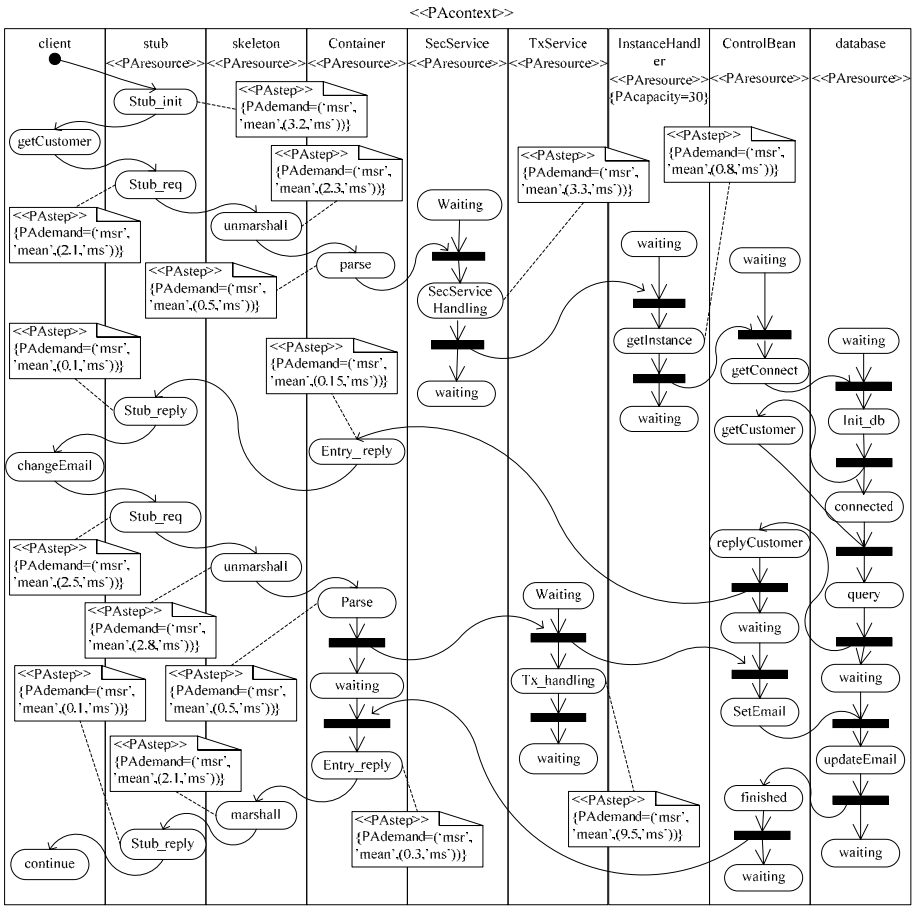

Fig. 8. Integrated activity diagram annotated with performance information

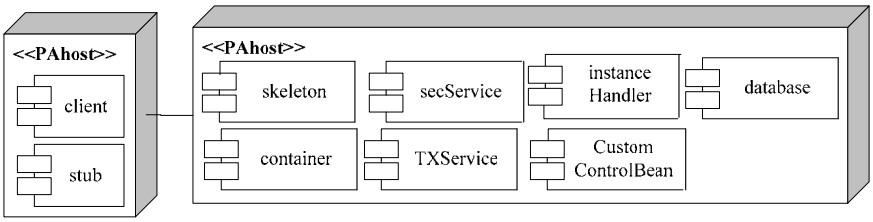

Fig. 9. Integrated deployment diagram of the case

Using transforming method, like proposed in [15], a LQN performance model can be derived from the UML models of Fig.7-Fig.9, which can be read directly by existing LQN solvers [16]. Then, performance estimates can be extracted for varying system parameters by using the LQN model.

To validate the presented performance model, we conducted measurements with our benchmark implementation. Fig.10 shows the response time of updating customer email information as a function of the total number of clients, including model prediction and experiment measurement result. We let the number of concurrent clients vary between 10 and 200 with the increment of 10 clients. Fig.10 indicates that the model is able to predict the response time of system reasonably well-the greatest difference between the measurement and the prediction is only 10%.

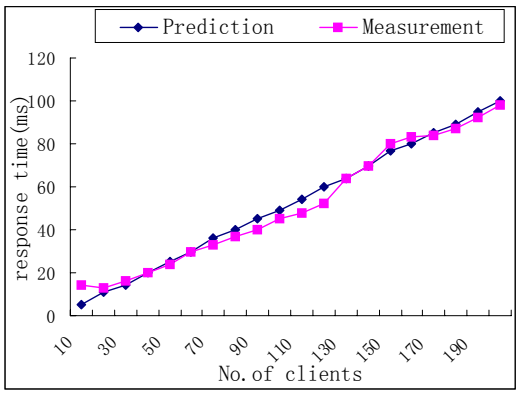

Fig. 10. Comparison of LQN prediction with measurement results

5 Related Work

To reflect the effect of middleware to system performance, one straightforward method is that inclusion performance overhead of middleware components into application components, for example, directly adding the overhead of stub creation and marshaling to business method execution demand. This coarse-grained dealing approach is simple and does not increase the complexity of performance model;

however, the precision of model is not enough. Moreover, the resulting model cannot efficiently identify performance bottlenecks occurred in middleware layer.

Another method is directly modeling entire system including middleware and application. As in [10], author describes a framework for constructing LQN performance model based on the modular structure of Application Server and the application components. In [6] [7] [8], author model the performance for CORBA-based distributed object system using QN/LQN formalism. Compared with the first method explained above, the method helps to improve the accuracy of performance model. However, it requires performance analyst to be familiar with internal details of middleware, which decreases the ease of use.

To predict the performance for component-based system hosted by middleware infrastructure, in [9] authors propose a solution based on empirical testing and mathematical modeling. The models describe generic behaviors of application server components running on COTS middleware technologies, the parameters value in model are discovered through empirical testing. In this solution, incorporating application-specific behavior into the equation is difficult, and the results from the empirical testing cannot be generalized across different hardware and software platforms, so different platforms need different test cases, which is economically impractical.

In order to derive performance model from UML, a first approach based on architectural patterns for client/server systems is presented in [17]. The authors, rather than proposing a transformational methodology, describe the pattern through Class and Collaboration diagrams and directly show their corresponding EQN models. The aim of our work mainly obtains integrated UML descriptions based on architectural patterns, then derives performance model from existing method automatically.

In [18], the authors propose automatic inclusion of middleware performance attributes into architectural UML software models, and a method based on Model Driven Architecture that transform a middleware-independent UML model into a middleware-aware UML model. Their idea is like ours. However, the transformation method is different. Our proposed method bases on composition of sub-model, and architecture pattern-based refinement can be extended to deal with different style middleware. In addition, in [18] mainly address the impact of remote communication, considering middleware services very simply; whereas ours emphasize the effect of middleware services and give the solution, besides remote invocation communication.

6 Conclusion

To reflect the performance effect of middleware to component-based system, this paper proposes an approach integrating middleware component interactions and performance attributes into application UML model. Thus, derived performance model from resulting UML models can efficiently represent the impact of middleware. In the future work, we will deal with bottleneck identifying, configuration setting choosing, and automatic tools supporting.

References

1. Williams, L.G., Smith, C.U.: Performance Evaluation of Software Architecture. In: Proceedings of the First International Workshop on Software and Performance WOSP98. ACM, New York, NY (1998)164-177
2. Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, Marta Simeoni: Model-Based Performance Prediction in Software Development: A Survey. IEEE Transactions on Software Engineering. Vol.30, No.5, May (2004) 295 – 310
3. Ward-Dutton, N: Containers: A Sign Components are Growing Up. Application Development Trends. January (2000) 41-46
4. Wolfgang Emmerich: Software engineering and middleware: a roadmap. In: Proceedings of the 22nd International Conference on Software Engineering, on the Future of Software Engineering. ACM, New York, NY(2000) 117-129
5. M. Woodside, Petriu, D., Khalid Siddiqui: Performance-related Completions for Software Specifications. In: Proceedings of the 24th International Conference on Software Engineering. ACM, New York, NY (2002)22-32
6. Kahkipuro,P.: Performance Modeling Framework for CORBA Based Distributed Systems, PhD thesis, Department of Computer Science, University of Helsinki (2000)
7. Petriu, D., Amer, H., Majumdar, S., Abdull-Fatah, I.: Using analytic models for predicting middleware performance. In: Proceedings of the Second International Workshop on Software and Performance WOSP2000.ACM, New York, NY (2000) 189-194.
8. Williams, L.G., Smith, C.U.: Performance Engineering Models of CORBA-Based Distributed-Object Systems. In: Proceedings of International CMG Conference, Computer Measurement Group (1998) 886-898
9. S. Chen, Y. Liu, I. Gorton, and A. Liu: Performance Prediction of Component-Based Applications. J. Systems and Software. Vol. 74, No. 1, January (2005) 35-43
10. Jing Xu, A. Oufimtsev, M. Woodside, L. Murphy: Performance Modeling and Prediction of Enterprise JavaBeans with Layered Queuing Network Templates. In: Proceedings of Workshop on Specification and Verification of Component-Based Systems, ACM, New York, NY (2005)
11. Object Management Group: UML Profile for Schedulability, Performance, and Time. 2003.
12. M. Woodside, Tutorial Introduction to Layered Modeling of Software Performance, Edition 3.0, Carleton University, http://sce.carlton.ca/rads ,2005.
13. Douglas Schmidt, Michael Stal, Hans Rohnert, Frank Buschmann, Pattern-Oriented Software Architecture, Volume 2, Patterns for Concurrent and Networked Objects. John Wiley & Sons, New York, NY (2000).
14. http://www.once.com.cn
15. D.C. Petriu and H. Shen: Applying the UML Performance Profile: Graph Grammar-Based Derivation of LQN Models from UML Specifications. In: Proceedings of 12th International Conference Computer Performance Evaluation, Modeling Techniques and Tools, LNCS 2324, Springer-Verlag, Berlin (2002)159-177.
16. Franks,G.,Hubbard,A.,Majumdar,S.,Petriu,D.C.,Rolia,J.,Woodside,C.M: A toolset for Performance Engineering and Software Design of Client-Server Systems. Performance Evaluation, Vol.24, No.1-2, February (1995)117-135
17. H.Gomaa and D. Menasce: Performance Engineering of Component-Based Distributed Software Systems. Performance Engineering. LNCS2047, Singer, (2001) 40-55
18. Tom Verdickt, Bart Dhoedt, Frank Gielen,and Piet Demeester, Automatic Inclusion of Middleware Performance Attributes into Architectural UML Software Models. IEEE Transactions on Software Engineering. Vol. 31, No.8, August (2005) 695-711.