

# A Performance Analysis Infrastructure for Component-Based System Hosted by Middleware

Yong Zhang, Tao Huang, Jun Wei, Ningjiang Chen

Institute of Software, Chinese Academy of Sciences, Beijing 100080, China  
{yzhang, tao, wj, river}@otcaix.iscas.ac.cn

**Abstract.** An infrastructure is proposed for automatically modeling the impact of middleware to component-based system at architectural level performance evaluation. The basic ideas behind infrastructure are separation of performance modeling concerns between application and middleware, and declarative performance modeling manner. Taking container style middleware for example, the details of proposed infrastructure are illustrated in this paper, which are well-founded on other existing and proven approaches, such as UML Profile for Schedulability, Performance and Time (UML SPT profile) and Layered Queueing Network performance model generation techniques. To validate proposed infrastructure, a case study is conducted.

## 1 Introduction

Software middleware helps to alleviate complexity associated with developing distributed software, enables separation of concerns between application logic and system services, such as distributed communication, transaction, message, security, concurrency control, component life cycle management, etc. At the same time, middleware will obviously impact the architecture and performance of component application, which must be taken into account for evaluating the performance of component application from early design specification [1,2,3,4].

At times, middleware as supporting platform is not included as a part of application design description, performance information of which is missing. In order to accurately predict the performance of middleware-based system, some works have been undertaken [5-10], but these approaches require analyst familiar with middleware internal structure and performance modeling technique itself. The steep learning curve behind these methods is one of the major impediments to their adoption.

Our viewpoint is that the modeling process for middleware-based system should be usable in everyday practice by software developer with the help of supporting tool. It should be necessary to automatically construct platform dependent model directly from given application model description and deployment platform. Such tools should be able to read respective model, process it, and produce the composite result suitable for further analysis.

In this paper we propose an infrastructure to support the process, based on separation of concerns and a kind of declarative performance modeling method. Middleware performance concerns are given by platform provider in a manner suitable for

declarative modeling. In this paper Extensible Markup Language (XML) Schema is adopted for the purpose. According to predefined Schema, application specific middleware usage and performance information are declared by analyst. Based on proposed infrastructure, the information is automatically weaved into design description of component application. The composite models which include middleware impact and be suitable for derivation of performance analysis model can be got. Thus, different performance modeling concerns of application and middleware are dealt with by different roles in analysis process.

The rest of this paper is organized as follows: a brief survey of related work is given in Section 2; the general description of proposed infrastructure is shown in Section 3; taking a container style middleware for instance, the details of infrastructure are described in Section 4, and a case study based on container middleware is demonstrated in Section 5. Section 6 gives the conclusions of the work.

## 2 Related Work

Some works have been conducted to investigate the impact of middleware to performance modeling. In [5,6,7], authors model the performance for CORBA-based distributed object system using QN (Queueing Network) / LQN (Layered Queueing Network) formalism. In [8], authors describe a framework for constructing LQN performance model based on the modular structure of Application Server and application components. One of the major drawbacks of these methods is that one must manually construct the performance model by studying underlying middleware, which requires analyst to master middleware details and performance modeling method itself.

In [9,10] authors propose a solution based on empirical testing and mathematical modeling: models describe generic behaviors of application components running on COTS middleware; parameter values in model are discovered through empirical testing. In this solution, incorporating application-specific behavior into the equation is difficult, and the results from empirical testing cannot be generalized across different platforms, which is economically impractical.

In [3,11,12], authors propose applying Model Driven Architecture (MDA) paradigm to analyze the impact of middleware. In [3], authors describe the supporting middleware as a kind of completion information to application design, and suggest the use of MDA idea to supplement it. In [11], authors propose using the Model Driven performance analysis to a distributed middleware based enterprise application developed using EJB. But, neither of [3] and [11] give the concrete transformation details. In [12], authors propose automatic inclusion of middleware performance attributes into architectural UML software models, and the process is illustrated taking a CORBA-based system for an example. But the transformation process of [12] lacks necessary tool supporting. Moreover, only the impacts of distribution communication are discussed.

### 3 Overview of Infrastructure

The structure of proposed infrastructure is shown in Figure 1, which includes three layers. The top layer is application performance concerns description, and the bottom layer is middleware specific performance concerns, which can be a description library including different kind of middleware. The middle layer will be responsible for finishing the assembling process of application and middleware concerns.

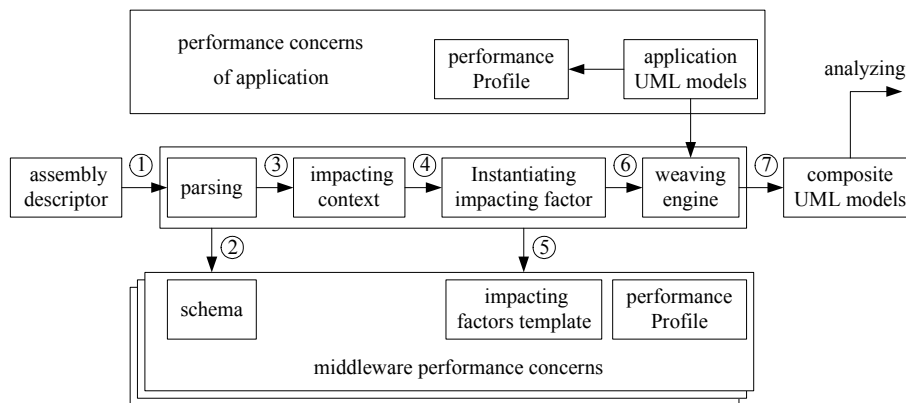


Fig. 1. Structure of proposed infrastructure

In our work we use Unified Modeling Language (UML) as application design specification, and UML SPT performance profile as performance annotation, which has been adopted by Object Manage Group. SPT performance profile extends UML by proving stereotypes and tagged values to represent performance requirements, the resources used by the system and some behavior parameters, to be applied to certain UML behavior model [13,14]. To model performance concerns of middleware, performance profile is also necessary. Here we use UML SPT profile as performance annotation for both application layer and middleware layer.

Performance concerns of different middleware are different, which can be described respectively by platform provider. But a kind of uniform form should be adopted. We present a kind of description in structured XML file in this paper, and illustrate how to use it by container style middleware, by which the information about impacting factors is organized. According to given XML Schema format of a concrete platform, application specific middleware usage information can be provided in a XML file, called assembly descriptor file in our work.

The input of middle layer includes application UML models and application specific assembly descriptor file. According to corresponding XML Schema in middleware model library, an XML parser in infrastructure will parse the assembly descriptor. Then, a middleware impacting context is created, which includes concrete middleware usage and relative performance information. With impacting context, the predefined impacting factors templates of middleware are instantiated. After that, weaving engine will analyze application UML models and insert these impacting factor instances into proper position. The output of infrastructure is composite UML

model including middleware impact, from which performance analysis model can be derived. In this paper, we use LQN as target formalism, just one of several possible modeling formalisms [15,2].

### 4 Modeling for Container Style Middleware

The performance concerns of different type middleware can be different. In this section we will use container style middleware as an example to illustrate the cores of proposed infrastructure. Container style middleware is a kind of supporting environment in common use for server-side component technologies, such as Enterprise Java Beans and CORBA Component Model, which enables separation of concerns between application logic and system services [16]. The components interacting process based on container middleware can be described as Figure 2.

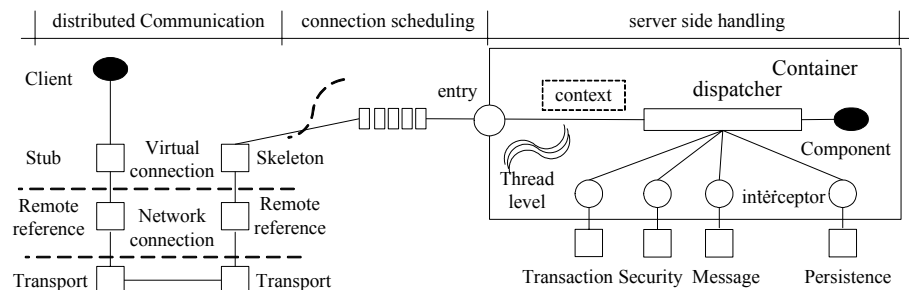


Fig. 2. Component interacting process based on Container middleware

#### 4.1 Modeling Performance Concerns

As illustration, here only three major impacting factors of container middleware are considered: distributed communication, server side connection contention and request processing inside container. We try to build some templates for impacting factors, which can be instantiated according to specific function requirement. The modeling follows the UML SPT profile concepts. We will use scenario-based analysis method [2,13], and scenarios are described by using UML activity diagram with stereotypes and tagged values annotation defined in SPT profile.

The granularity of a scenario step depends on the level of abstraction chosen by the modeler. If finer granularity is required, a step at one level of abstraction can be resolved into a corresponding scenario comprising finer-grained scenario steps. More detailed model which reflects the exact software architecture of middleware, from which more accurate performance estimate can be derived, on the other hand, the system model will be more complex. The tradeoff need be considered.

Distributed communication generally bases on client-proxy-server pattern [17]. Client side and server side components (like stub, remote reference, and skeleton shown in Figure 2) will perform some additional operations on request and response,

such as marshaling and un-marshaling. These operations will incur overhead, the impacts of which are modeled in Figure 3. (Here only synchronous call is illustrated).

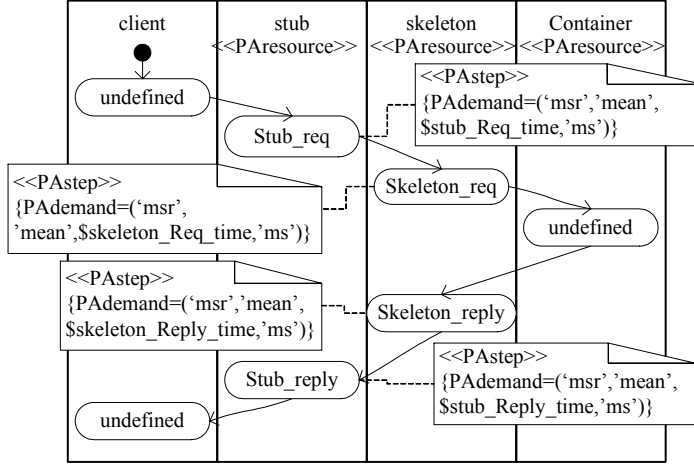


Fig. 3. Modeling overhead of distribution

Relative middleware components are stereotyped as `<<PAresource>>`, and key actions impacting performance are stereotyped as `<<PAstep>>`, which demand can be tagged with `PAdemand`. UML SPT profile provides a useful facility supporting symbolic variables, expressed as `$name`, as well as values for parameters. Here, the demand of each step is described using variable, which will be determined according to application specific model assembly descriptor.

Generally, middleware can process multiple connections concurrently. Here we suppose container middleware using thread pool model. Calls from different client connection will execute in different threads. When the number of client connections exceeds the number of available threads, connections will wait in queue for obtaining thread. The connections can be scheduled based on specific scheduling policy. The size of thread pool and scheduling policy will impact application performance.

Middleware services to be used during invocation usually are declared in application deployment descriptor. To add these services to component system dynamically, architectural pattern similar to chains of interceptor (responsibility) is generally employed in Container middleware [17]. Middleware services serve the request concurrently under the control of different processes/threads. Triggered services will cause overheads.

For convenience, the impact of connection contention and cost of middleware services are modeled in a single UML activity (as shown in Figure 4). The attribute multiplicity, representing the size of thread pool, is described by SPT profile tagged value `PAcapacity`, and scheduling discipline is described by `PAchdPolicy`. Each kind of middleware services is abstracted as a service component described by variable `$serviceNamei`. These variables are placeholders for middleware services that will be used in specific application. Here we model each service as a whole, instead of

modeling its internals, service demand of which is represented by variable *\$serviceTime*.

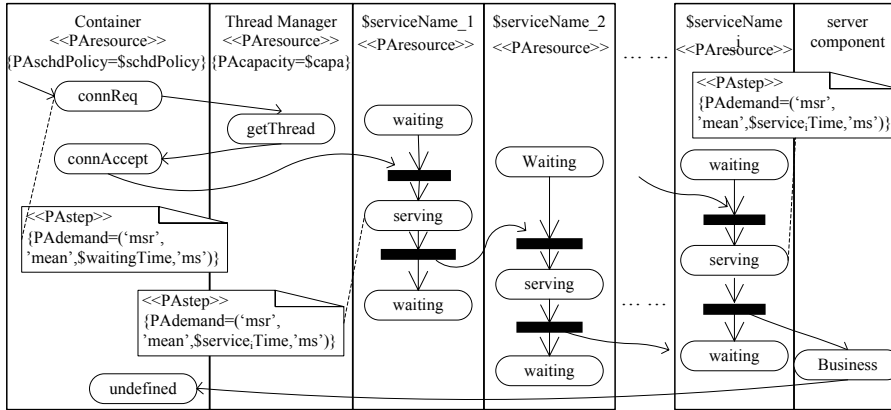


Fig. 4. Modeling connection contention and server side processing cost

#### 4.2 Organizing Concerns Information in XML

The performance concerns discussed in last section should be provided in a manner suitable for declarative modeling. In this paper, we use XML file to organize the information, which complies with the habit of application developer using middleware. At the same time, XML-based description is easily extensible, which provides convenience for further abstracting other impacting factors and refining current factors. The Schema of which is shown in Figure 5.

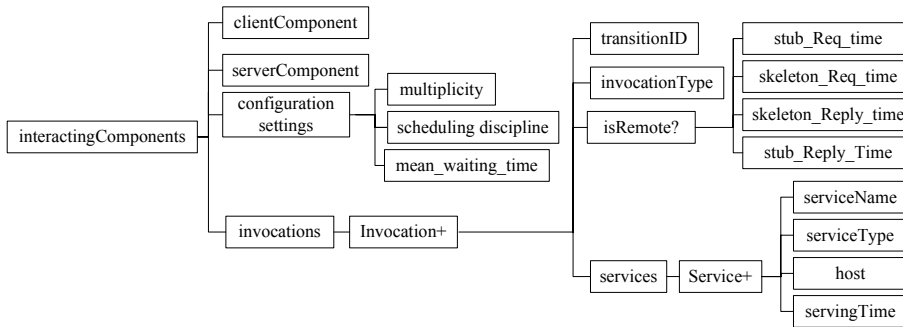


Fig. 5. Schema of model assembly descriptor

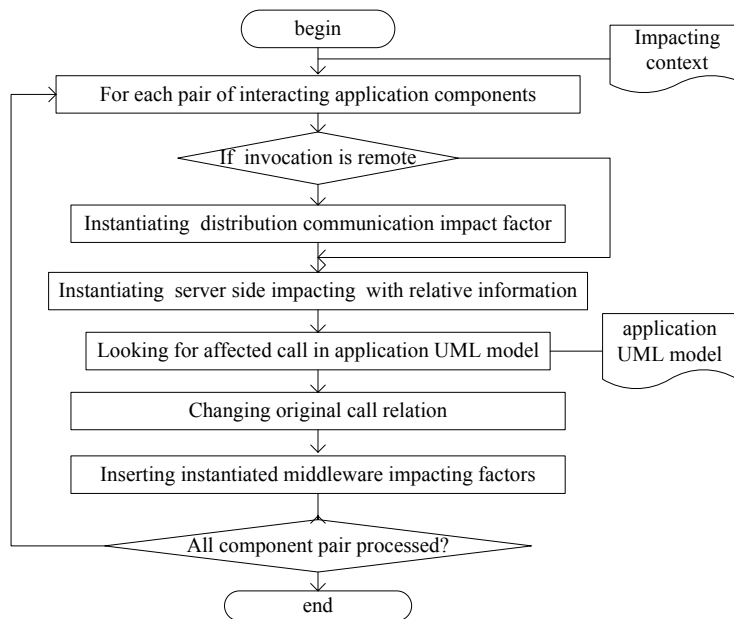
The elements in Figure 5 are declared according to each pair of interacting components that use middleware. Element `<configuration settings>` provides information about modeling connection contention, a sub-element of which `<mean_waiting_time>` declares the mean waiting time to get a thread for accepting connection. For each in-

vocation between application components, there is a <invocation> declaration: <transitionID> represents the transition (described in application UML activity diagram) referring to this call; element <invocationType> shows that the invocation is synchronous or asynchronous; element <isRemote> specifies service demands of distribution communication phases; Element <services> represents which middleware services will be used by this invocation, in which the service details are specified.

According to the given Schema, application specific model assembly descriptor can be constructed, which is one of inputs to infrastructure. The assembly descriptor provides metadata for assembling component application UML model and middleware impacting factors.

### 4.3 Weaving Engine

From the assembly descriptor file, middleware impacting context that includes application specific middleware usage and performance information will be created. Using the information, the middleware impacting factors templates are instantiated. Based on impacting context, weaving engine will locate the affected invocations in application UML models, in particular UML activity diagram in this paper, and insert instantiated impacting factors into it. During the process, the original call in activity diagram can need to redirect. The processing steps are illustrated in Figure 6.



**Fig. 6.** The weaving process

In addition, the UML deployment diagram of component application need also be changed to reflect the allocation of middleware components to hardware devices.

### 5 Case Study

To illustrate proposed infrastructure, a case study is conducted, modeling the performance of an online store application based on EJB container middleware. The scenario can be described as Figure 7: *customer* component makes a remote synchronous invocation to *businessBean* component to find required customer information, which need middleware security service; then updates email address of customer to database, during the process middleware transaction service is needed. We will predict the response time of updating customer email information at varying number of clients. *Customer* and *businessBean* are implemented as EJB components. The deployment platform we employed is a J2EE Application Server, called *OnceAS* [18]. Client machine generates a number of users who repeatedly generate random requests. There is a think-time distribution that each user uses to determine how long to wait between requests. We let the number of clients vary between 10 and 200 with the increment of 10 clients. The workload is described with `PApopulation` tagged value. Database is modeled indirectly by specifying as external resource with UML SPT tagged value `PAextOp`, attached to steps that access database, *getCustomer* and *setEmail*. The detail modeling of database is outside the scope of this paper.

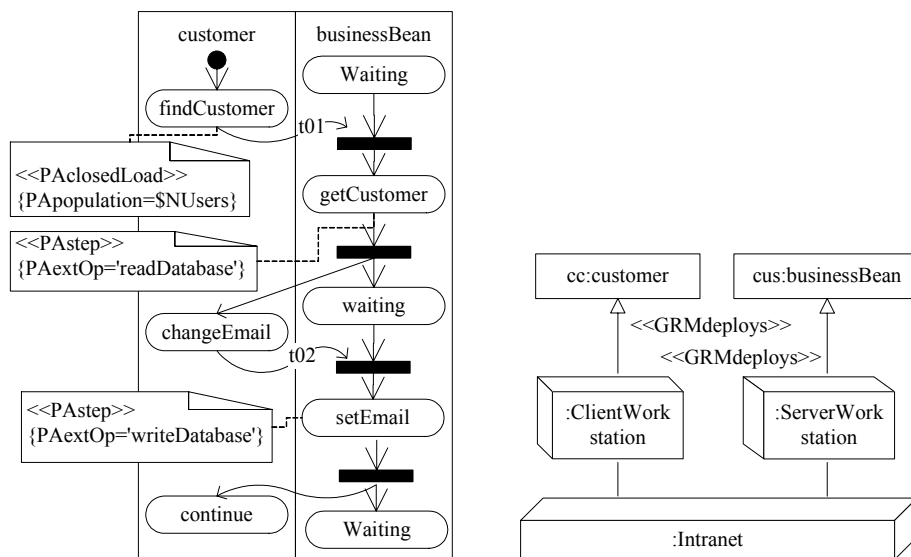


Fig. 7. UML models of case study

With the help of a profiling toolkit *OptimizeIt* [19], service demands of middleware are obtained from a prototype implementation of the case. According to the Schema shown in Figure 5, the model assembly descriptor file of case study is given below. The file, together with the UML diagrams in Figure 7 (in XML format according to XMI standard transformation), will be the inputs to infrastructure.



```

<interactingComponents clientComponent=customer serverComponent=businessBean>
<configuration settings multiplicity=10 scheduling discipline=FIFO mean_waiting_time=2.8ms/>
<invocations>
<invocation transitionID=t01 invocationType=synchronous>
<isRemote stub_req_time=1.5ms skeleton_req_time=1.9ms
skeleton_reply_time=3.4ms stub_reply_time=3.2ms getConn=2.8ms/>
<services>
<service serviceName=SecService serviceType=Security
Host =serverWorkstation servingTime=5.3ms/>
</services>
</invocation>
<invocation transitionID=t2 invocationType=synchronous>
<isRemote stub_req_time=2.1ms skeleton_req_time=2.5ms
skeleton_reply_time=3.3ms stub_reply_time=2.8ms/>
<services>
<service serviceName=TxService serviceType=Transaction
Host =serverWorkstation servingTime=19.5ms/>
</services>
</invocation>
</invocations>
</interactingComponents>
    
```

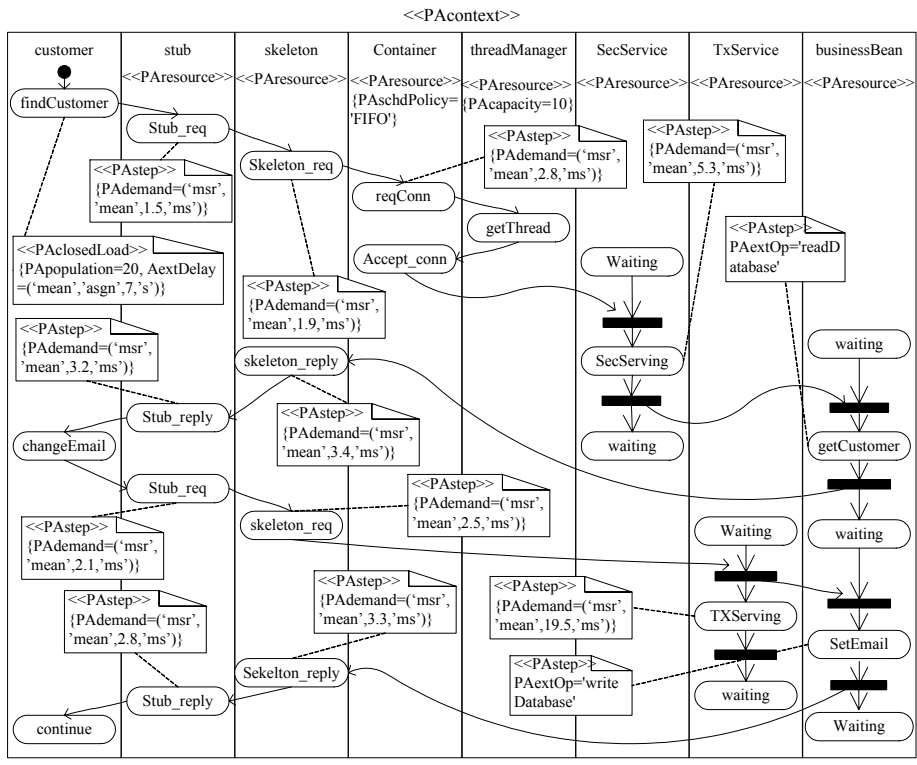


Fig. 8. Composite UML activity diagram annotated with performance information

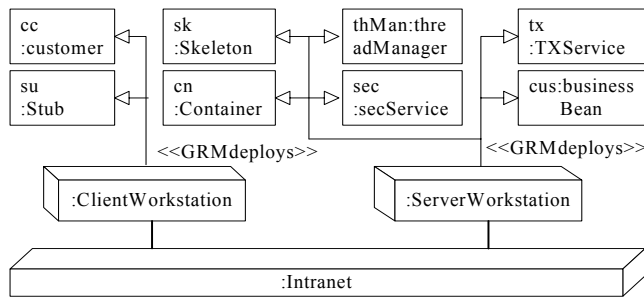


Fig. 9. Composite deployment diagram

By weaving engine, the composite UML models are obtained, as shown in Figure 8 and Figure 9, including activity diagram and deployment diagram. For the sake of clarity, the performance information of application components is not shown in figures.

Using the method proposed in [20], the LQN performance model is obtained from Figures 8 and 9, which can be solved with the model solving tools provided in [21]. For space limitation, the resulting LQN performance model is omitted here. Performance estimates of the resulting LQN model are extracted for varying parameters. To validate the prediction results, we conducted measurements with our benchmark implementation. The comparison of predictions and measurements are shown in Figure 10. Two group data are given under different configuration settings with different size of thread pool: Configuration A has 10 threads, and configuration B has 30 threads.

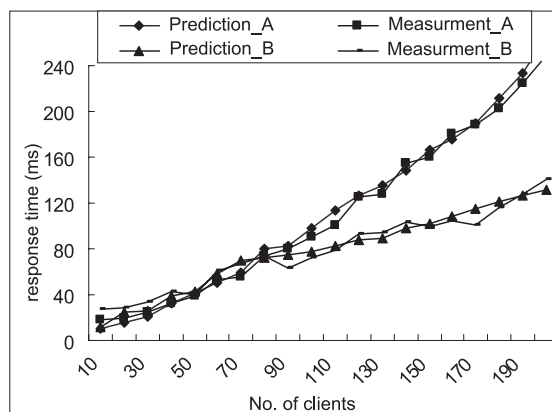


Fig. 10. Comparison of predictions and measurements

The prediction differences shown in Figure 10 are about 12% and 10% under two different configurations respectively. This comparison demonstrates the validity of proposed approach. At the same time, the differences also illustrate that the impacting factors of container middleware can be further extended and refined, such as, the impact of component instantiating, database connection contention, etc. Their aggregate impact may explain the differences.

## 6 Conclusion

In this paper we propose an infrastructure supporting automatically modeling the middleware performance impact to component-based system. Taking container style middleware as an example, the cores of proposed infrastructure are illustrated. The ideas behind this are separation of performance modeling concerns between application and middleware, and declarative performance modeling manner. We suggest the performance concerns of middleware be provided in a manner of suitable for declarative modeling by platform provider. In this paper, major performance impact factors of container middleware are abstracted, and modeled following the UML SPT concepts. Through a model assembly descriptor file, the model weaving engine is responsible for assembling the performance impacting factors into component application UML model. The prototype of proposed infrastructure has been realized. Using a case study, we validate proposed method.

In the future, several aspects of studying will be conducted. Firstly, we plan to continue to refine and abstract other performance impacting factors of container middleware that are not considered currently, such as more precise thread pool mechanism, instance pool mechanism, database connection contention, and component persistence mechanism, etc. Next, we will improve the abstraction level of middleware impacting factors description. We will try to build UML profile for middleware impacting factors which can be applied to application UML models directly, by using the standard extension mechanism provided by UML [22]. In addition, we will extend middleware concerns library using other type middleware platform, based on the analysis to an open source CORBA middleware project.

## References

1. 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
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. M. Woodside, Petriu, D., Khalid Siddiqui: Performance-related Completions for Software Specifications. In: Proceedings of the 24<sup>th</sup> International Conference on Software Engineering. ACM, New York, NY (2002)22-32
4. Pooley, R., Software engineering and performance: A road-map, In: Proceedings of the 22nd International Conference on Software Engineering, on the Future of Software Engineering. ACM, New York, NY (2000) 189-200
5. Kahkipuro,P.: Performance Modeling Framework for CORBA Based Distributed Systems, PhD thesis, Department of Computer Science, University of Helsinki (2000)
6. 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.
7. 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

8. 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)
9. Liu, Y., Fekete, A., and Gorton, I.: Design-Level Performance Prediction of Component-Based Applications. IEEE Transactions on Software Engineering, Vol.31, No.11, November (2005) 928-941
10. 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
11. James Skene and Wolfgang Emmerich, Model Driven Performance Analysis of Enterprise Information System, Electronic Notes in Theoretical Computer Science 82 No.6, 2003.URL: <http://www.elsevier.nl/locate/entcs/volume82.html>
12. 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.
13. Object Management Group, UML Profile for Schedulability, Performance, and Time, v1.1, 2005.
14. C.M.Woodside and D.C. Petriu, Capabilities of the UML Profile for Schedulability Performance and Time, In Workshop SIVOES-SPT held in conjunction with the 10th IEEE RTAS'2004.
15. M. Woodside, Tutorial Introduction to Layered Modeling of Software Performance, Edition 3.0, <http://www.sce.carleton.ca/rads/lqn/lqn-documentation/tutorialg.pdf>, Carleton University, 2002
16. Ward-Dutton, N: Containers: A Sign Components are Growing Up. Application Development Trends. January (2000) 41-46
17. 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).
18. <http://www.once.com.cn>
19. <http://www.borland.com/us/products/optimizeit/>
20. D.C. Petriu and H. Shen: Applying the UML Performance Profile: Graph Grammar-Based Derivation of LQN Models from UML Specifications. In: Proceedings of 12<sup>th</sup> International Conference Computer Performance Evaluation, Modeling Techniques and Tools, LNCS 2324, Springer-Verlag, Berlin (2002)159-177.
21. 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
22. Object Management Group, Unified Modeling Language Specification, Version 1.4.2, <http://www.omg.org/docs/formal/04-07-02.pdf>