Journal of Computer Science 3 (10): 780-785, 2007
ISSN 1549-3636

Mapping UML Component Specifications to JEE Implementations

Jyhjong Lin
Department of Information Management, Ming Chuan University
Kweishan, Taoyuan County, Taiwan 333

Abstract: Component-based Software Engineering (CbSE) has become a well-accepted approach for developing complex software systems due to its significant advantages on composition and reuse. In practice, however, its use still requires the conjunction of a component specification method that describes how system requirements are satisfied in terms of software components. Such a component specification is then implemented in a variety of software component models (e.g., COM+、CORBA、EJB). To achieve this, a sound mapping from the specification to a designated component model is critical. In addition, for rapid advances on Internet technologies, software systems have gradually been architected as processing in distributed environments. Since a distributed environment involves often synchronous/asynchronous messages communicating among various processes, this paper focuses therefore on the mapping issue from a component specification to a component model that particularly takes into consideration of the communicating of synchronous/asynchronous messages. To illustrate, an on-line e-Learning curriculum order system is modeled for demonstrating the mapping idea. In completing the component specification, we adopt the well-known UML Components method, while in the component model we use the Enterprise Java Beans (EJB) standard in that EJB is the core component model of the JEE (J2EE) platform which supports well distributed operations/services. With such a practical mapping, software systems can be developed in a more effective way by specifying requirements in UML Components and implementing software components in EJB with the communicating of synchronous/asynchronous messages among various processes.

Key words: Component-based software engineering, design mapping, asynchronous message, UML components, EJB

INTRODUCTION

Component-based Software Engineering (CbSE) has become a well-accepted approach for developing complex software systems. One of its commonly recognized advantages is the support of quick reuse and composition of preexisting com- ponents with only, if necessary, a few new ones in building these systems. In practice, however, reusing and compositing preexisting components still needs a complete analysis of system requirements and then an intrinsic design of software com- ponents that collaboratively satisfy these requirements before reuse and composition may take place. It is therefore crucial to employ a sound component specification method in CbSE for the requirements analysis and software components design.

In the literature, many discussions about CbSE and its associated component specification methods have been presented such as Catalysis[1], SCIPIO[2], O2BC[3], and UML Components[4]. In general, these approaches provide well an effective way to specify and design desired software components that take advantages of CbSE in developing complex software systems, and hence result in a complete component specification that is then easily endeavored for implementation (i.e., via reuse and composition of preexisting components) in a variety of software component models (e.g., COM+ [5]、CORBA[6]、EJB[7]). To achieve the implement- ation, as one may conceive, a sound mapping from the specification to a designated component model is critical. Most of these existing methods, nevertheless, pay very little attention on such a design mapping issue and therefore make the implementation difficult in realizing the specification.

For rapid advances on Internet technologies, software systems have gradually been architected as those with transactional operations in distributed environments where various processes need to communicate by exchanging messages between each

other. Among such messages, synchronous ones are most usually used for synchronizing relevant operational activities across the communicating processes. In addition to this synchronous collaboration mode, however, asynchrony- ous messages are also often required for several applications due to their inherent advantages on collaborative operations, e.g., parallel processing of relevant activities between various processes for expediting these operations. To recognize the aforementioned trend of exploiting CbSE in developing distributed software systems with messages exchanged in a synchronous/asynchronous manner, this paper focuses therefore on the specific issue of design mapping in CbSE from a component specification to a component model that particularly takes into consider- ation of the communicating of synchronous/ asynchron- ous messages among various processes in distributed environments. In the proposed discussion, the com- ponent specification will be completed by applying the UML Components method[4] that uses the most well-known UML[8-11] as its modeling tool, while in the component model, the Enterprise Java Beans (EJB) standard[7] will be adopted in that EJB is the core component model of the JEE (J2EE) platform[12] which supports well distributed operations/services. With the practical mapping, software systems can be developed in a more effective way by specifying requirements in UML Components and implementing software com- ponents in EJB with the communicating of synchronous/ asynchronous messages in distributed environments.

This research is organized as follows. Section 2 presents the existing work related to the proposed mapping issue. Section 3 introduces the mapping approach with two specific discussions. Finally, section 4 has the conclusions and future work.

RELATED WORK

UML Components: In the context of Component-based Software Engineering (CbSE), software systems can be built quickly by reusing and composing preexisting components with only, if necessary, a few new ones. Since the selection and adoption of adequate preexisting components depends much upon the comprehension of system requirements and then the intrinsic design of required software components that collaboratively satisfy these requirements, it is therefore crucial to have a sound component specification method in CbSE for the requirements analysis and software components design.

In the literature, many discussions about component specification can be found in CbSE approaches such as Catalysis, SCIPIO, O2BC, and UML Components[1-4]. Among these approaches that in general provide well an effective way to specify and design desired software components in developing complex software systems, UML Components in our knowledge is a simple but practical one that provides the most natural way to deal with the specification and design of software components. Based on this method, the specification and design work is divided into four phases:

- Requirement Definition - captures user requirements with a use case model and then represents these requirements with a domain (class) model;
- Component Identification - takes the use case and domain models as inputs to define desired software components and their associated interfaces; two types of components are identified: (a) business components that provides system components, through associated business interfaces, with business services for accessing core business information; and (b) system components that provides users, through associated system interfaces, with transactional operations by sequences of interactions with various interfaces of business components;
- Component Interaction-considers how the system and business components work together to deliver required functionality for system users; the implementation design for the transactional operations and business services provided by these components through their associated interfaces is decided;
- Component Specification-completes the specification of components by modeling in details the transactional operations and business services provided by these components through their associated interfaces; possible constraints to be applied on the transactional operations and/or business services are also identified.

Upon completion of the above four phases, desired system/business components for collaboratively delivering required functionality can be identified, specified, and designed with their interfaces, collaborative interactions, and constraints on collaborations being completely modeled. Figure 1 shows the system architecture of system/business components together with other ordinary user interaction components.

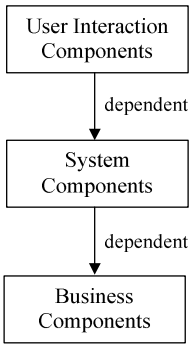

Figure 1: the system architecture of component based model

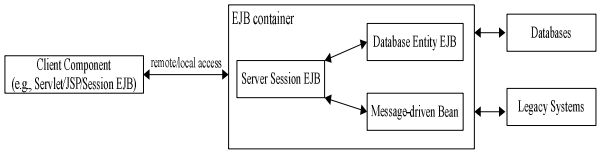

Figure 2: the EJB model in J2EE runtime architecure

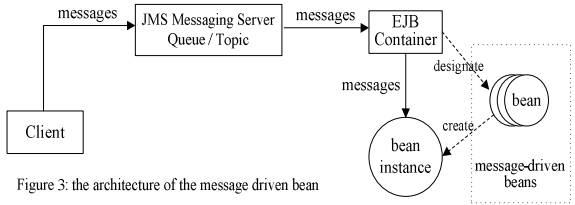

Figure 3: the architecture of the message driven bean

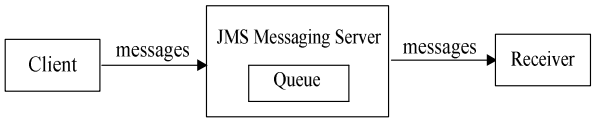

Figure 4: the PTP JMS model

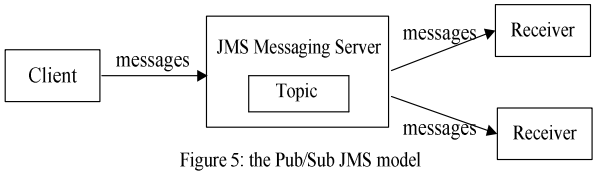

Figure 5: the Pub/Sub JMS model

EJB technology and synchronous/ asynchronous messages: Enterprise JavaBeans (EJB)[7] is the core component model of the JEE (J2EE) platform[12] developed by Sun Microsystems for full supports of distributed operations/services. Figure 2 shows the EJB model and the access relationships among its various parts. More specifically, *EJB beans* and the *EJB container* play most critical roles in EJB where the EJB container provides EJB beans with a runtime environment over the EJB server (e.g., arranging such issues as accesses of these EJB beans and services of desired security/persistence/ transactions/concurrency

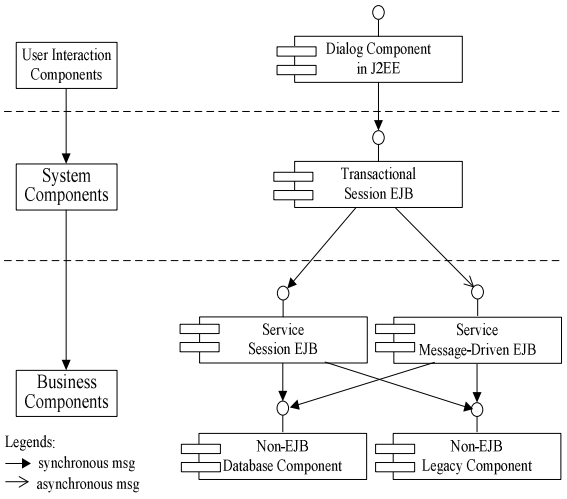

Figure 6: the non-persistent approach

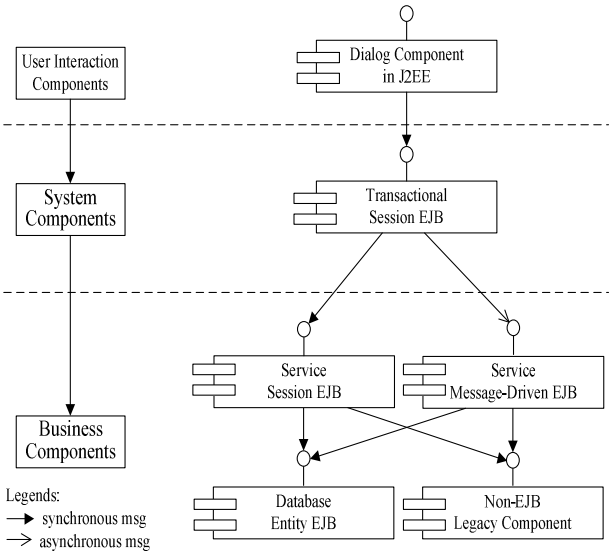

Figure 7: the persistent approach

controls). Hence, EJB beans are server-side components deployed in the EJB container for local/remote accesses by client components. EJB is thus suitable for providing distributed operations/services[7]. There are three kinds of EJB beans: session, entity, and message-driven ones.

- A s*ession bean* is designed to perform business processes on behalf of its prospective client without shared accesses across other clients. A session bean can be either stateful or stateless where *stateful* holds session states on behalf of prospective clients, and *stateless* does not maintain states information in order to process immediately access requests from clients. A session bean consists of a home interface, a remote interface, and a bean implementation where

the home interface declares remotely callable 'create' methods for creating new bean instances, the remote interface defines remotely callable methods in created bean instances, and the bean implementation carries out those methods defined in the remote interface.

- Unlike session beans, an *entity bean* allows shared accesses across multiple clients. It is used to represent data stored in a database (i.e., synchronization of its contents with those stored in the database) for multiple invocations from various clients. An entity bean also consists of a home interface, a remote interface, and a bean implementation where the home interface declares remotely callable methods for creating and locating new bean instances, the remote interface defines remotely callable methods in created bean instances, and the bean implementation carries out those methods defined in the remote interface.
- A *message-driven* bean is designed to achieve asynchronous invocations in EJB. All methods in the message-driven bean can be invoked only by the EJB container and hence do not have corresponding interfaces as those in session or entity beans. When clients send out messages toward the message-driven bean, these messages are received first by the Messaging Server in a Java Message System (JMS) that puts them into a Queue or Topic for further forwarding to the EJB container. Based on messages received, the EJB container invokes designated methods in the message-driven bean with these messages declared as the parameters into these methods. Figure 3 shows the architecture of the message-driven bean.

Synchronous/asynchronous messages: In distributed environments, various processes often need to communicate by exchanging messages between each other. Among such messages, synchronous ones are most usually used for synchronizing relevant operational activities across the communicating processes. In addition to this synchronous collaboration mode, however, asynchronous messages are also much popular for several applications due to their inherent advantages on collaborative operations such as parallel processing of relevant activities between various processes for expediting collaborative operations. In EJB, synchronous messages are handled by a Remote Method Invocation (RMI) interface where request messages from client components are transmitted into the remote interfaces of destination components for invoking any designated methods specified in these interfaces. In general, such a synchronous collaboration mode facilitates the use of remote components and methods almost exactly as that of local accustomed ones. On the other hand, asynchronous messages are handled by a Java Message System (JMS) in which a Messaging Server receives messages from client components and then based on two transmitting models forwards these messages to destination components.

- Point-To-Point (PTP): by utilizing specific Queues, a message received from a client component is stored in a queue for transmitting then to one and only one destination component. The message will be in the queue until it is successfully transmitted to the destination component or otherwise invalidated. In particular, the client and destination components are time independent; that is, these two components need not be activated at the same time, their executions do not have any sequential relationships. Fig. 4 illustrates the PTP model.
- Publish/Subscribe (Pub/Sub): by utilizing specific Topics, a message received from a client component is stored in a topic (i.e., message publishing) for broadcasting then to all interested components (i.e., message subscribing). As in PTP, the message will be in the topic until it is successfully broadcasted to interested components or otherwise invalidated. Similar to PTP again, the client and interested components are time independent; that is, these components need not be activated at the same time, their executions do not have any sequential relationships. Figure 5 shows the Pub/Sub model.

MAPPING APPROACHES

For the design mapping in CbSE from a component specification to a component model with emphasis on the communicating of synchronous/asynchronous messages, the component specification is described by applying the UML Components method, while the component model is presented by using the EJB standard. Two mapping approaches are presented that illustrate how specification components in UML Components are implemented by corresponding software components in EJB among which a/synchronous messages are communicated.

A non-persistent approach: In this way, system components in UML Components are mapped into session EJB beans to collaboratively provide users with transactional operations for satisfying their functional requirements. Further, business components in UML

Components are mapped into session and/or message-driven EJB beans (in conjunction of other Non-EJB components) to collaboratively provide system level session EJB beans with business services by accessing core business information. Figure 6 shows the mapping idea.

For each system component that is purposed to provide users, through its associated interfaces, with transactional operations by sequences of interactions with various interfaces of business components, specific transactional session EJB beans are imposed to collaboratively achieve those transactional operations where these session EJB beans are designed to perform and control operational processes within which desired activities are accomplished by sequences of interactions with business level (session/message-driven) EJB beans through synchronous/asynchronous message calls to these business level EJB beans. As stated above, a synchronous collaboration mode (messages receipt by usual session beans) would allow the use of remote components and methods almost exactly as that of local accustomed ones, while an asynchronous collaboration mode (messages receipt by special message-driven beans) would activate parallel accomplishment of various local/remote activities that expedites operational processes.

For each business component that provides system components, through its associated interfaces, with business services for accessing core business information, specific session and/or message-driven EJB beans are imposed to collaboratively provide business services by accessing core business information within which two common information resources are considered: databases and legacy systems. More specifically, for each database, a Non-EJB database component is imposed to represent the database accessed by these session/message-driven beans. Further, for each legacy system, a Non-EJB legacy component is devised to represent the legacy system that provides other business information required for these session/message-driven beans.

A persistent approach: In this approach, system components in UML Components are still mapped into transactional session EJB beans to collaboratively provide users with transactional operations for satisfying their functional requirements. However, business components in UML Components result herein not only session and message-driven EJB beans, but particularly also entity EJB beans. These EJB beans (in conjunction of other Non-EJB legacy components) collaboratively provide system components with business services by accessing core business information. Figure 7 shows the mapping idea.

As in the above non-persistent approach, each system component is mapped into specific transactional session EJB beans that collaboratively achieve transactional operations by controlling operational processes within which desired activities are accomplished by business level session/ message-driven EJB beans through synchronous/ asynchronous message calls to these business level EJB beans.

For each business component, specific session and/or message-driven EJB beans are imposed to collaboratively provide business services by accessing core business information. However, unlike the above non-persistent approach that employs Non-EJB database components to represent the databases accessed by these session/message-driven EJB beans, entity EJB beans are particularly introduced herein to avoid the performance and persistent problems arising from using Non-EJB database components in the business level. More specifically, in JEE, an entity EJB bean keeps its contents synchronized with the data stored in a database; this significantly promotes its performance and persistent capabilities for accessing core business information stored in external databases. Therefore, for each data store in a database, an entity EJB bean is imposed to represent the data store accessed by session/message-driven beans. Finally, as in the above approach, a Non-EJB legacy component is derived from each legacy system to provide session/message-driven beans with other required business information.

It should be particularly noticed that compared to the above non-persistent approach, the persistent approach by employing entity EJB beans offers better performance and persistent capabilities for accessing core business information stored in external databases. In contrast, this employment nonetheless results in a disadvantage of more complex and difficult implementation due to the inherent complexity of realizing entity EJB beans.

CONCLUSIONS

Component-based Software Engineering (CbSE) has been recognized as a fast and effective approach for developing complex software systems due to its emphasizing on quick reuse and composition of preexisting components. In its practical use, however, a complete analysis and design of required software components is crucially needed for capturing what and

how these preexisting components are reused/ composed in building system functions. After a component specification is completed, its implementation (i.e., via reuse and composition of preexisting components) can then be easily achieved by a corresponding mapping into a designated software component model (e.g., COM+ or CORBA or EJB). Such a design mapping is therefore another critical issue in CbSE that makes the implementation easy in realizing the specification. To address this issue, we presented in this paper two mapping approaches that illustrate how specification in UML Components is implemented by a corresponding mapping into software components in EJB. Further, since software systems have gradually been architected as distributed ones where synchronous/asynchronous messages are frequently exchanged among various processes, our approaches thus takes particularly such communication modes into consideration with session and message-driven EJB beans mapped from specification in UML Components.

Comparing our two mapping approaches, the non-persistent approach is simpler in implementation due to its imposing Non-EJB database components to represent the database accessed by business level session/ message-driven EJB beans. This however results in some performance and persistence problems that are alleviated in the persistent approach by employing entity EJB beans (although this unavoidably results in a disadvantage of more complex and difficult implementation due to the inherent complexity of realizing entity EJB beans). As our future work, we will continue to explore the applicability of our approaches on some application systems including for instance distributed sales-, marketing-, and services-related systems. As one may conceive, while developing these systems, experiences about the applicability of our approaches can be collected correspondingly for validating their usefulness and effectiveness. Since CbSE is gradually popular in developing complex (especially distributed) software systems, analysis of software components and their mapping into a designated implementation model have begun to gain many attentions in the literature; our work presents a possible discussion on these needs.

REFERENCES

1. D'Souza, D. and A.. Wills, 1999. Catalysis: Objects, Frameworks, and Components in UML. Addison Wesley.
2. Veryard, R., 1998. SCIPIO: Aims, Principles, and Structures, SCIPIO Consortium.
3. Ganesan, R. and S. Sengupta, 2001. O2BC: A Technique for The Design of Component-Based Applications. Proc. of 39th Int'l Conf. and Exhibition on Technology of Object-Oriented Languages and Systems.
4. Chessman, J. and J. Daniels, 2001. UML Components. Addison Wesley.
5. Rofail, A. and Y. Shohoud, 1999. Mastering COM and COM+. Sybex.
6. OMG, 1999. CORBA Component Model Specification, Version 1.3. http://www.omg.org/docs/ad/99-06-08.pdf.
7. Blevins, D., 2001. Overview of The Enterprise Java Beans Component Model, Component-based Software Engineering: Putting The Pieces Together. Addison Wesley.
8. Rumbaugh, J., I. Jacobson, and G. Booch, 2004. The Unified Modeling Language Reference Manual. 2nd Edition. Addison Wesley.
9. Booch, G., I. Jacobson, and J. Rumbaugh, 2005. The Unified Modeling Language User Guide. 2nd Edition. Addison Wesley.
10. Quatrani, T., 2002. Visual Modeling with Rational Rose 2002 and UML. 3rd Edition. Addison Wesley.
11. Miles, R. and K. Hamilton, 2006. Learning UML 2.0. O'Reilly.
12. Singh, I., et al., 2002. Designing Enterprise Applications with The J2EE Platform. 2nd Edition. AddisonWesley.