

# Enhancing EJB Component Model<sup>\*†</sup>

Vladimír Mencl, Jiří Adámek, Adam Buble,  
Petr Hnětynka, Stanislav Višňovský

## Abstract

Component Based Software Development (CBSD) aims to lower software development costs by providing sophisticated facilities for software reuse. The Sun Microsystem's Enterprise Java Beans (EJB) [4] is one of the currently used component based platforms for development of distributed, object-oriented applications.

While EJB is successful at providing the basic features of software reuse, several aspects are not addressed by the EJB specification, although they have been already heavily explored in theoretical research.

This paper aims at capturing the prospects and propositions for further evolution of the SOFA project [12, 6]. This project aims at improving the EJB architecture, mostly by incorporating results of theoretical research, leveraging the results acquired in the SOFA project. Such a task requires a deep analysis of the EJB architecture, the theoretical research has to go on to adapt its results specifically to the EJB architecture. Also, in certain cases, it is desirable to further explore the theoretical results.

## 1 Introduction

To support rapid software development, applications are currently constructed from reusable components. Using this approach, the architecture of an application can be considered as a collection of interconnected components, usually in a distributed environment. The Sun Microsystem's Enterprise Java Beans (EJB) [4] is one of the currently used component based platforms for development of distributed, object-oriented applications.

While EJB is successful at providing the basic features of software reuse, several aspects are not addressed by the EJB specification, although they have been already heavily explored in theoretical research.

This paper aims at capturing the prospects and propositions for further evolution of the SOFA project [12, 6]. This project aims at improving the EJB architecture, mostly by incorporating results of theoretical research, leveraging the results acquired in the SOFA project. Such a task requires a deep analysis of

---

<sup>\*</sup>partially supported by the Grant Agency of the Czech Republic (project 201/99/0244)

<sup>†</sup>partially supported by the Grant Agency of the Academy of Sciences of the Czech Republic (project number A2030902)

the EJB architecture, the theoretical research has to go on to adapt its results specifically to the EJB architecture. Also, in certain cases, it is desirable to further explore the theoretical results.

## 1.1 Formal specifications

In recent time, formal methods have gained acceptance as one of the most promising ways of software engineering. By using proven methods of building mathematical theories, formal methods provide an almost usable way to develop software systems. A wide area of research deals with the problem how to bring these rigorous methods into practice.

One of the most successful attempts of formal reasoning in practice is based on a description of behavior by means of the communication among software components. Based on the description, the behavior can be evaluated to test, whether it has given properties.

In the SOFA project [5], the idea of behavior protocols as a regular-like notation for describing behavior of software components was introduced [6, 7]. It is not a completely new issue: the idea is based on behavior protocols known from the area of object oriented programming [8, 9, 10] and component-based architecture description languages [11].

Although the concepts of EJB [4] differ from the principles of SOFA, there are some important similarities: both SOFA and EJB use entities which can be viewed as components; in both the systems, the entities communicate through interfaces, which are described as sets of methods. Method descriptions are very similar as well (in both the cases Java-like notation is used to describe parameters and types). According to these facts, we propose to embed behavior protocols into EJB. Once it would be done, the enhancements of behavior protocols and results of a research in related areas can be applied to both SOFA and EJB.

## 1.2 Component Updating

Throughout running of a distributed system, a necessity of an update of components can arise. In a large continuously running systems with an amount of always connected clients, stopping of a part of the system could be improper or even infeasible from many reasons (unavailability of the system, propagating an information about change, reconnection of other communicating components and clients, etc.). So, it would be very useful to accomplish an update of a component during run time of a system in a such way that client could not make any special action and even recognize the change.

Most of the current component based systems do not deal with the dynamic change of components.

### 1.3 Enhancing Component Composition

It is generally agreed that a basic property of a software components is the list of components requirements – services to be provided by a different component. A component providing such a service may not be known to the developer at the time the component using the service is being developed; it can be provided later and assembled with the component requiring the service at a later stage, usually called *component assembly*. We demonstrate that in certain cases, especially when independent service providers and/or third-party components are considered, the composition paradigm has to be extended by introducing different composition modes. We propose to apply these principles to the EJB component composition model, aiming to provide a more sophisticated component composition framework.

### 1.4 Component Model Interoperability

Current EJB implementations lack support for interoperability with other component models, such as CORBA Component Model and/or DCOM. As DCOM is a proprietary Microsoft standard, we focus on the CCM, an open standard, only. The CCM standard, however, is not yet finished. In spite of that, there are already several implementations; we suppose that CCM will follow the CORBA's way and will become widely used, mainly because of its platform and programming language independence.

The CCM proposed standard includes parts regarding the EJB interoperability. To finalize the process of the standardization of CCM, there should be an implementation available. The current implementations, though, focus mainly on the core parts of the specification. Therefore (at least) the details of the proposed interoperability model cannot be tuned sufficiently to meet real usage requirements.

## 2 State of the art

### 2.1 Formal specifications

All versions of the EJB specification incorporate a possibility to describe “functionality” of a bean by adding a textual comment describing the bean and other parts specified by its deployment descriptor in plain English.

The EJB 2.0 specification significantly improves a description of the overall structure and relations among enterprise beans, mostly by introduction of the new Container Manager Persistence specification. By describing the relationships between entity beans, developer can specify how beans depend on each other.

From previous versions of specification, the bean environment description could incorporate information on references to home interfaces of required beans. This information resembles the way software architectures in academia specify the requirements of a component. However, the important difference is that,

naturally, the deployment descriptor specifies only requirements of the home interface (the bean factory) whereas in software architectures one needs to provide a description of number of instances of components, e.g., beans.

## 2.2 Component Updating

The current EJB 2.0 specification does not discuss updating beans. It only contains a very short (three lines) paragraph, which states that the container provider typically provides support for versioning the installed beans and may allow enterprise bean classes to be upgraded without invalidating existing clients or losing existing enterprise Bean objects. The specification does not provide any further information.

In the other currently used industrial components models — the OMG's CORBA Component Model [13] and the Microsoft's Distributed Component Object Model (DCOM) [1], there is also no mention about the dynamic update nor normal (non-dynamic) update. If a component in DCOM should be updated, the developer must create a new component with a new identification, which does not have any relation to the previous one and the client applications must also be updated in order to use the new version.

There are several projects, mostly academic, which aim at providing dynamic updating of components.

The SOFA (Software Appliances) is a project aiming at providing a platform for development of distributed applications composed from software components. A part of the SOFA project is the DCUP (Dynamic Component UPdating) project, which is a specific architecture of SOFA components and allows to update components at run time. This update, transparent to the other components, is possible due to special implementation objects, which control the update and can temporarily block calls to the updated component.

The Java Distributed Run time Update Management System (JDrums) [2], allows to update Java programs without any need of user interaction; this way, systems can even be updated while they are running. The updating in JDrums is allowed by a modified Java virtual machine, which is able to update classes and objects on the fly.

In the area of non-component based systems, several systems supporting the dynamic update can be found too. The Solaris 8 Operating Environment [3] enables to change the kernel while it is running. It allows to reduce the downtime of a system and change kernel without requiring a system reboot.

## 2.3 Enhancing Component Composition

The EJB component composition model allows a bean-developer to specify a list of services required by the bean. These requirements have to be satisfied during the *assembly stage*; for each requirement, a decision has to be made as to which bean is to provide the service required. Although this decision may be altered in subsequent assembly stages, it may not be altered at or after the deployment stage. The bean selected to satisfy the requirement may not be

updated afterwards (unless an additional assembly stage takes part) and the beans have to be packaged and shipped together.

## **2.4 Component Model Interoperability**

As the CCM is Adopted Specification and not the Available Specification, according to the OMG vocabulary, there is a little of related research available. The EJB/CCM interoperability is not a focus of the implementors today, and there are not many other researchers interested in non-finished standard. However, interoperability of different component models is still an important topic. The interoperability of middleware is common nowadays, but not of different component models. This is chiefly caused by relative novelty of widely used, non-academic, component models.

# **3 Goals and Future work**

## **3.1 Integrating Behavior Protocols with EJB**

None of the possibilities of description in EJB targets a description of "functionality" by means of formal description. Although the area of formal methods research is very active in academia and partially in practice, the research to bring formal methods into the world of EJB is very limited.

EJB is a widely recognized and employed standard with very good (although plain English only) specification. The description of beans centralized in a deployment descriptor, which is defined partially by developer of a bean, by assembler and deployer. Because of XML definition it is very easy to enhance deployment descriptor to provide means of extended description of a bean, in this context, by formal description.

We aim to integrate a formal description for deployment descriptor of a bean. Specifically, based on our previous work on behavior protocols [6, 7], we will research the possibility to describe functionality of local and remote interfaces of beans by behavior protocol notation. It should describe the communication "protocols" provided and required by a bean. Further, we will investigate the consequences of additional information provided by deployment description, e.g., the relationship description. This work should be based on formal model of enterprise javabeans, e.g., based on the agent concept [6, 7]. The model would provide means of explicitly identify possibilities for formal verification of the description.

## **3.2 Enhancing Behavior Protocols: Component Composition and Dynamic Architectures in EJB**

As a part of the project, we plan to target the following two problems related to behavior protocols: component composition and description of dynamic architectures.

### 3.2.1 Component composition

In SOFA, the idea of composed components was introduced. Not like a primitive component, which is implemented in an underlying implementation language, a composed component is implemented by connecting several more simple components together in such a way that the resulting entity implements all specified interfaces. Every of these (sub)components can be primitive or composed as well. Thus, the architecture of a system based on SOFA can be viewed as recursively nested hierarchy of components.

Using this approach, whole SOFA application can be viewed as a composed component as well (but it has no interfaces, because there is no another component to connect with). This is the reason, why component composition is an issue interesting for EJB. We can think of enterprise java beans as of components which (composed together) constitute architecture of the whole EJB application.

From the point of view of behavior protocols, component composition brings three problems:

1. For every component, a developer specifies behavior protocol, which describes all possible sequences of actions on all component's interfaces. When two components are composed together, how to detect if one component's behavior conforms to the behavior protocol of the second one (and vice versa)? How can we detect it when more than two components are composed together?
2. When two or more components are composed together, how to obtain a protocol which describes the behavior of the result of the composition?
3. For a composed component, how we can find out if the behavior of the composition result conforms to the protocol specified in the component specification.

As EJB doesn't use the idea of nested components (and for the whole application, viewed as component, there is no protocol specified), the third problem cannot occur in EJB. However, it is completely resolved in [4].

Targeting the first two problems, we are working on:

1. Specifying all the problems, which can occur during component composition (from the point of view of behavior protocols). The result of this analysis should be definition of which situations (during component composition) are correct and which are not. Moreover, we are developing the algorithm, which can detect incorrectness and report information, which can be used by developer to remove the incorrectness from the architecture specification.
2. We are developing an algorithm which gives the behavior of component composition (in the form of behavior protocol), taking behavior protocols

of subcomponents as its input. It is used as a part of the correctness-testing algorithm mentioned above as well. This is the reason, why this issue relates to EJB (which hasn't nested components).

### 3.2.2 Dynamic architectures

Currently, the architecture of a SOFA application is static in the sense that all components and connections among their interfaces are specified in the source-code by a developer. However, typical architecture of an EJB application is dynamic: instances of components can arise or expire in the run-time, as well as connections between components. We are planning to embed dynamic behavior into SOFA as well, because it is generally a very useful feature. Thus, behavior protocols notation and the communication model have to be changed as well.

## 3.3 Component Updating

Currently the EJB platform has a strong runtime support, provides a big set of runtime services available (transactions, security, persistence, etc.) but EJB does not specify the update of beans and leaves it for a container provider. The dynamic update does not specify at all.

This project will aim to enhance the EJB architecture in order to it allow dynamic (i.e., during run time) update of the beans and specify all necessary things linked with this update ability.

Updating of a bean at run time means disconnecting it from other parts of an application and connecting a new version of the bean back into the application. But this disconnecting should be hidden for the currently connected clients and also should be feasible on the standard (unmodified) java virtual machine.

There are several issues, which have to be solved to dynamic updates could be achieved.

First, it should be investigated if a dynamic update of a bean can be practicable without any action on a client of the bean.

Next, a functionality of the EJB execution environment, the EJB container, should be enhanced by ability to stop a currently running bean, temporarily block client calls, obtain code of a new version of the bean and launching this new version. Attention has to be paid to a state of the updated bean (with respect to a kind of the bean — stateless and stateful session bean, entity bean). And also if all kinds of the beans can be dynamically updated.

In conjunction with updating (not only dynamic but also static) of beans, the versioning of bean implementations has to be drawn up. The EJB 2.0 specification leaves versioning, just as updating, for a container provider and does not specify it. A version model has to determine, if the versions of beans will create single line, tree or acyclic graph and how the update between nonsuccessive versions can be accomplished. In distributed environment, global uniqueness of version identifiers must also be ensured to avoid creating two beans with the same identification.

### 3.4 Enhancing Component Composition – Introducing Composition Modes and Autonomous Points into EJB

We believe that in certain cases, the component composition decision should be postponed from the assembly stage to a later stage, either the deployment stage or the runtime stage. By analyzing possible scenarios of component updates, we found it useful to distinguish two special cases:

1. Components may be provided as standalone services with a well defined interface (including also, e.g., QoS parameters). In such a case, the service provider may wish to hide changes made to components realizing the service. As far as the agreed properties of the service are held, the provider should not be obliged to reveal the changes made.
2. Component developers may wish to allow custom replacements of designated subcomponents, e.g., to provide a point for customization of the application, or integrating a third-party component.

It is important to note that only certain composition points are suitable for providing such a greater flexibility in component composition. In other composition points, such a later composition change is undesirable and the choice made by the initial bean assembler and/or the bean developer should be enforced. We aim providing facilities for the bean developers and assemblers to denote the stage at which the composition decision has to be made, and also to denote the stage after which the decision cannot be changed. This can include a sequence of assembly stages, as well as the deployment stage and the runtime stage.

### 3.5 Component Model Interoperability

The goal of the project is to precisely and correctly specify the required mapping of component models on each other, as well as specification of needed interlaid software components. This will be achieved by implementing such an interoperable EJB/CCM test application. Most probably, there will be implemented the bridges (in CCM for CCM clients, or in EJB for EJB clients). Still, there should be defined the mapping of OMG Trading service to JNDI (the same is already done for OMG Naming service) and vice versa. Of course, if any other problem will arise, it must be solved as well to reach the main goal - the correct specification of the EJB/CCM interoperability model. The special emphasis should be laid on the openness of the specification in the sense of upgrading both the CCM and the EJB specification. It should be still straightforward to map the interoperability model to new component model specification. The reasoning behind the real interoperability should remain unchanged.

By interoperability we mean both the client/component and the component/component communication. Of course client/client interoperability is not a topic of CCM/EJB interoperability model. However, the component/component

case is more important, as the second one can be easily transformed to this one (using delegation/wrappers). Moreover, the component/component case will be crucial for an application programmer, because of reuse of components. The client application usually communicates with "mother" component(s) only.

## **4 Expected Results**

This project aims at improving the EJB architecture; mostly by incorporating results of theoretical research. Such a task requires a deep analysis of the EJB architecture; the theoretical research has to go on to adapt its results specifically to the EJB architecture.

The project consists of several loosely-tied tasks; the following sections describe the expected results of each of these tasks.

### **4.1 Integrating Behavior Protocols with EJB**

An extension to the EJB 2.0 deployment descriptors will be proposed, allowing to augment the bean specification with formal behavior specifications, expressed as behavior protocols for both the bean and the home interface. Algorithms for verifying protocol compliance; these algorithms may be implemented in a bean assembly tool.

### **4.2 Component Composition and Dynamic Architectures in EJB**

Extensions of behavior protocols will be derived as stated in 3.2, supporting dynamic architectures, as well as providing more correctness tests in component compositions. EJB will be considered in these extensions; the extensions may be implemented as an extension to the assembly and deployment tools.

### **4.3 Component Updating**

The proposed enhancements can significantly improve usability of the EJB platform. With the dynamic update mechanism, the changes in applications can be done quickly, without stopping of the whole system, clients of beans do not require any special activity, the update should be transparent for them.

The obtained experiences and techniques can be applied to enhance other systems by dynamic updating features.

### **4.4 Enhancing Component Composition**

Focus will be put on enhancing the EJB composition model. The deployment descriptor will be extended to support introducing different composition modes. An assembly tool employing these new features will be developed. Also, the

deployment tool will have to be altered. Should component updating be considered (see other sections of this report), the EJB runtime would have to be modified. The outcome of this project will include suggestions to enhance the EJB specification and might contain sample implementation for an EJB server.

To demonstrate the benefit of the extensions proposed, a set of sample scenarios will be provided.

## 4.5 Component Model Interoperability

The primary result is correct EJB/CCM interoperability model, i.e., mapping of one component model to the other one. As a proof of the concept, it should be implemented at least component/component interoperability scenario. As we are primarily interested in EJB, the most wanted case is EJB component uses (requires) CCM component, but the other case (EJB provides for CCM) should be implemented as well to complete the proof. The client/component scenario is not needed to be implemented directly, as it can be converted to component/component scenario easily using delegates.

## References

- [1] F. E. Redmond III: *DCOM: Microsoft Distributed Component Object Model*, IDG Books Worldwide, Inc., Foster City, USA, 1997
- [2] Andersson J. and Ritzau T.: Dynamic Code Update in JDrums, The ICSE'00 Workshop on Software Engineering for Wearable and Pervasive Computing, Limerick 2000.
- [3] Reliability, Availability, and Serviceability (RAS) for the Solaris 8 Operating Environment, Sun Microsystems, Inc., 2000, <http://www.sun.com/software/solaris/ds/ds-ras/>
- [4] Enterprise JavaBeans Specification, Version 2.0, Sun Microsystems, 2001
- [5] SOFA project, <http://nenya.ms.mff.cuni.cz/thegroup/SOFA/sofa.html>
- [6] Plášil, F., Višňovský, S., Bešta, M.: *Bounding Component Behavior via Protocols*, In proceedings of TOOLS USA '99, pp. 387-398, Aug 1999
- [7] Plášil, F., Višňovský, S., Bešta, M.: *Behavior Protocols*, TR 2000/7, Dept. of SW Engineering, Charles University, Prague. Revised 08/07/2001
- [8] Mikušík, D.: Implementation of Synchronization Protocols. Master thesis 1998, MFF UK, <http://nenya.ms.mff.cuni.cz/thegroup/>
- [9] Plášil, F., Mikušík, D.: Inheriting Synchronization Protocols via Sound Enrichment Rules. Proceedings of Joint Modular Programming Languages Conference, Springer LNCS 1204, March 1997

- [10] J. van den Bos, C. Laffra: PROCOL: A Concurrent Object-Oriented Language with Protocols Delegation and Constraints. Acta Informatica, Springer-Verlag, 1991, pp. 511-538
- [11] D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, D. Bryan, W. Mann: Specification and Analysis of System Architecture Using Rapide. IEEE Transactions on SW Engineering, 21(4), 1995
- [12] Plášil, F., Bálek, D., Janeček, R.: *SOFA/DCUP: Architecture for Component Trading and Dynamic Updating*, Proceedings of ICCDS'98, May 4-6, 1998, Annapolis, Maryland, USA, IEEE CS Press 1998
- [13] OMG: *CORBA Component Model Specification*, orbos/99-07-01