Testing Conformance of EJB 3 Enterprise Application Servers

Sander de Putter, Serguei Roubtsov and Alexander Serebrenik
Department of Mathematics and Computer Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Keywords: Enterprise JavaBeans, Specification Conformance, EJB Application Server.

Abstract: Enterprise JavaBeans (EJB) is a component technology used for enterprise application development. EJB is currently being implemented by such application servers as GlassFish, OpenEJB, JBoss, WebLogic and Apache Geronimo. Through the entire history EJB claimed its adherence to the "write once, run anywhere" philosophy of Java suggesting that an application developed for and deployed on one application server should be easily portable to a different application server. Therefore, one could have expected different application servers to adhere to the EJB specification. Adherence to this and related Java EE specifications is subject of the "Java EE 6 Full Profile" compatibility testing carried by Oracle. However, anecdotal evidence of discrepancies between the specification and certified implementations such as GlassFish, has been reported in the literature. In this paper we present an approach allowing one to go beyond the level of anecdotal knowledge and test requirements for EJB application servers with focus on portability. We apply the approach developed to test how well two popular "Java EE 6 Full Profile"-compatible EJB application servers, GlassFish and JBoss, conform to the requirements in the EJB specification. The results are alarming: both application servers failed on a number of tests, violating the specification. Moreover, in GlassFish conformance to a requirement varies depending on whether a local or a remote application is used. Lack of conformance to the EJB specification compromises the portability of the EJB applications, deviates from the portability philosophy of Java, leads to unexpected behaviour, and hinders the learning process of novice EJB developers.

1 INTRODUCTION

Enterprise JavaBeans™(EJB) is a component technology used for enterprise application development. While the first version of the specification was released in 1998 (Matena and Harper, 1998), its most recent version has been released in 2009 (EJB 3.1 Expert Group, 2009) and it is currently being implemented by such application servers as GlassFish, OpenEJB, JBoss AS and WebLogic.

Through the entire history EJB claimed (Matena and Harper, 1998; EJB 3.1 Expert Group, 2009) its adherence to the "write once, run anywhere" philosophy of Java (Hamilton, 1996) suggesting that an application developed for and deployed on one application server should be easily portable to a different application server. Application portability is essential for reuse of Enterprise Beans across application servers (Dorda et al., 1999), helps EJB developers and users to avoid vendor lock-in and fosters integration with additional technologies chosen by the customer (Krastev and Galletly, 2003). Therefore, one could have expected different application servers to adhere to the EJB specification. Adherence to this and related Java EE specifications is subject of the "Java EE 6 Full Profile" compatibility testing carried by Oracle.[1] However, anecdotal evidence of discrepancies between the specification and certified implementations such as GlassFish, has been reported in the literature (Serebrenik et al., 2009).

Conformance of application servers to the EJB specification is important for understanding EJB applications. Deviation of the application servers from the EJB specification is not documented and difficult to discover. Moreover, these discrepancies can put in jeopardy specification-based tools and techniques intended to facilitate the developers' comprehension. Thus, testing conformance of an application server to the EJB specification helps to indicate deviating behaviour, and contributes to improving the developers' understanding of EJB applications.

To address the problem of discrepancies between the EJB specification and its implementations we have formulated the following research questions.

[1] http://goo.gl/qjqa3.

de Putter S., Roubtsov S. and Serebrenik A..
Testing Conformance of EJB 3 Enterprise Application Servers.
DOI: 10.5220/0004864402720279
In *Proceedings of the 16th International Conference on Enterprise Information Systems* (ICEIS-2014), pages 272-279
ISBN: 978-989-758-029-1

RQ1. How should one test requirements for EJB application servers with focus on portability?

Rationale. An application server can be seen as a meta-program, *i.e.*, as opposed to traditional software testing an application server requires designing an input program. Moreover, due to the nature of an application server, the input program should be built, deployed and run as part of the testing process, *i.e.*, a novel approach to extracting information from the running application server is required. Compatibility testing also implies that input programs should be application-server independent.

Results. This paper applies the traditional testing methodology to the client-server architecture of EJB. To support multiplicity of requirements and application servers to be tested we specify application servers and test cases independently of each other, and provide a configuration file indicating how an application server should deploy and run a test case.

RQ2. How well do "Java EE 6 Full Profile"-compatible EJB application servers conform to the requirements in the EJB specification?

Rationale. Discrepancy between the specification and its implementations puts portability in jeopardy, leads to unexpected behaviour and can be confusing for novice developers accustomed to a different application server. Furthermore, as EJB technology is being taught and certified worldwide, clear relation between the specification and the implementation used during the training is a prerequisite for further successful employment of the trainees.

Results. We apply our approach to test the EJB specification conformance of two "Java EE 6 Full Profile"-compatible EJB application servers: GlassFish version 3.1.2[2], EJB reference implementation by Oracle, and JBoss AS version 7.1.1[3]. GlassFish is the reference implementation of an application server, *i.e.*, when novice developers learn EJB they are likely to encounter GlassFish first. JBoss AS, JavaBeans Open Source Software Application Server, recently renamed to WildFly is a popular (Kounev et al., 2004; Xian et al., 2006) open source application server. Of 21 tests GlassFish fails 8 and JBoss AS fails 2. Moreover, in GlassFish conformance to a requirement varies depending on whether a local or a remote application is used. In JBoss AS no local/remote discrepancies have been found.

[2]https://glassfish.java.net/.

[3]http://www.jboss.org/jbossas.

RQ3. What are the implications of the (lack of) conformance observed?

Rationale. The extent to which a given application server conforms to the specification may have impact on the portability of applications developed for that application server. Furthermore, the (lack of) conformance may lead to confusion for developers who are only familiar with application servers that behave differently for certain requirements.

Results. Our results suggest that EJB applications that do not conform to the EJB specification may not be portable between different application servers. Hence, the lack of conformance results in a deviation from the portability philosophy of Java. Furthermore, anecdotal evidence indicates that lack of conformance results in confusion amongst EJB developers, and affects validity of the analysis tools.

The remainder of the paper is organized as follows. After introducing EJB 3 in Section 2, we discuss **RQ1** and present our approach in Section 3. In Section 4 we present a motivating example. The approach is evaluated and **RQ2** and **RQ3** are addressed in Section 5. Finally, related work is reviewed in Section 6, while conclusions and directions for future work are sketched in Section 7.

2 OVERVIEW OF EJB 3

This section presents a brief overview of Enterprise JavaBeans and the client view interfaces of the EJB 3.1 session bean (EJB 3.1 Expert Group, 2009).

2.1 EJB 3 Alternatives

In addition to EJB 3 additional architectural approaches have been proposed for the development of (business) Web applications. Similarly to EJB 3, Windows Communication Foundation (.NET) and Jini/-JavaSpaces attempt to decouple interface and implementation, and support service lookup and discovery, as well as transaction management. However, Windows Communication Foundation does not distinguish between local and remote applications, while Jini/JavaSpaces has been reported to have failed to attract a significant market.[4] Tomcat is sometimes mentioned as an alternative to EJB as it used to offer a lightweight version of JavaEE. However, since 2011 next to the lightweight version a JavaEE certified version of Tomcat is offered as well, deeming the Tomcat

[4]http://goo.gl/gwZVd0.

vs. JavaEE discussion "tired and old".[5]

2.2 Enterprise JavaBeans

Enterprise JavaBeans (EJB) is a server-side component architecture for development and deployment of business applications. The business logic of the application is implemented in Java classes. Optionally, a set of EJBs is packaged into larger deployable application. Finally, the EJBs are deployed into an EJB container provided by an application server.

The EJB container is a run-time environment for EJBs within the application server (Figure 1). The container can contain one or more EJB modules which each can contain one or more EJBs. It manages the life cycles of bean objects, provides remote access, and coordinates distributed transactions.

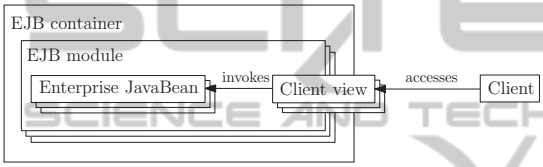

Figure 1: Overview of an EJB container.

The EJB architecture defines three types of EJBs: the session bean, the message-driven bean, and the deprecated entity bean. We focus on the first type, the session bean, because it has more interesting requirements: the specification (EJB 3.1 Expert Group, 2009) lists only a few requirements for the message-driven bean and entity bean is deprecated.

The session bean implements business methods of a module. The business methods are exposed through the client view interfaces of an EJB module. Client access to an instance of the session bean's class is mediated by the container via the session bean's client view. A client view is provided by the session bean and made available when deployed in the EJB container, the container itself is transparent to the client.

2.3 Client Views

Client views provide access to the business methods of a session bean. There are a total of six client views: Local, Remote, No-Interface, Web Service, 2.x Local Home and 2.x Remote Home.

The *Local* client view is location dependent. The EJB 3.1 specification only requires the support of Local clients that are packaged within the same module as the session bean that provides the Local client view. However, EJB application servers may also support access to the Local client view of session beans that are located in a different module.

The *Remote* client view is location independent. The Remote client may be located in the same Java Virtual Machine (JVM) or in a different JVM. Arguments and return values of methods of the Remote client interface are passed by value. Communication occurs through Java Remote Method Invocation (RMI) over the Internet Inter-Orb Protocol (IIOP).

The *No-Interface* client view is a variation of the Local client view. All public methods of the session bean's class are exposed to clients.

The *Web Service* client view provides web services access to the exposing session bean. This client view is specified in a WSDL document describing the client view interface as a set of endpoints operating on messages. Communication to this client view occurs through SOAP or plain XML over HTTP or HTTPS.

Finally, the *2.x Local Home and 2.x Remote Home* client views are used to provide Local and Remote client views in EJB 2.1 and earlier releases. These client views ensure backwards-compatibility for applications written for EJB 2.1 or earlier.

3 APPROACH

Discrepancies between the EJB specification and EJB implementations have been observed in the past (Dorda et al., 1999; Serebrenik et al., 2009). Such discrepancies may put portability at risk or lead to unexpected behaviour. It is therefore important that implementations adhere to the specification. Furthermore, testing requirements for multiple application servers is essential to the EJB component market.

To address **RQ1** we present an application of the traditional software testing methodology (International Software Testing Qualifications Board, 2011) to the client-server architecture of EJB. As opposed to traditional software testing, the testing requirements on an application server requires designing an input program and a client of the input program. A test case consists of the server-side application (the input program), a client application, and a specification of the expected behaviour. The input program adheres to a given requirement iff it exhibits the expected behaviour. To indicate an application server's lack of conformance to the EJB specification it is sufficient to present input programs that fail to adhere to the specification's requirements.

Our approach consists of the following steps. *First*, a server-side test application is built. When a requirement demands a certain behaviour to be supported, the server-side test application implementing

[5] http://goo.gl/3N5jpt.

Table 1: Requirements from Section 4.9.7 (EJB 3.1 Expert Group, 2009) and conformance results for GlassFish 3.1.2. and JBoss AS 7.1.1

Requirement	*GlassFish*	*JBoss*	*Remarks*
1 The interface must not extend the `javax.ejb.EJBObject` or `javax.ejb.EJBLocalObject` interface.			
Local	fail	fail	The presence of `EJBLocalObject` is ignored.
Remote	success	fail	In GlassFish the application fails to deployed. In JBoss the presence of `EJBObject` is ignored.
2 If the business interface is a remote business interface, the argument and return values must be of valid types for RMI/IIOP. The remote business interface is not required or expected to be a `java.rmi.Remote` interface. The throws clause should not include the `java.rmi.RemoteException`. The methods of the business interface may only throw the `java.rmi.RemoteException` if the interface extends `java.rmi.Remote`.			
Local	N/A	N/A	The requirement is only applicable to remote applications.
Remote	success	success	An exception is thrown *after* invocation of the test method in both servers.
3 The interface is allowed to have superinterfaces.			
Local	success	success	
Remote	success	success	
4 If the interface is a remote business interface, its methods must not expose local interface types, timers or timer handles, or the managed collection classes that are used for EJB 2.1 entity beans with container-managed persistence as arguments or results.			
Local	N/A	N/A	The requirement is only applicable to remote applications.
Remote	success	success	*After* invocation of the test method an error is raised both client-side and server-side.
5 The bean class must implement the interface or the interface must be designated as a local or remote business interface of the bean by means of the Local or Remote annotation or in the deployment descriptor. The following rules apply:			
5a If the bean does not expose any other business interfaces (Local, Remote) or No-Interface view, and the bean class implements a single interface, that interface is assumed to be the business interface of the bean. This business interface will be a local interface unless the interface is designated as a remote business interface by use of the `Remote` annotation on the bean class or interface or by means of the deployment descriptor.			
Local	success	success	
Remote	N/A	N/A	The requirement is only applicable to local applications.
5b A bean class is permitted to have more than one interface. If a bean class has more than one interface — excluding the interfaces listed below — any business interface of the bean class must be explicitly designated as a business interface of the bean by means of the `Local` or `Remote` annotation on the bean class or interface or in the deployment descriptor.			
Local	fail	success	In GlassFish all interfaces are exposed. JBoss does not deploy the bean.
Remote	fail	success	In GlassFish all interfaces are exposed. JBoss does not deploy the bean.
5c The following interfaces are excluded when determining whether the bean class has more than one interface: `java.io.Serializable`; `java.io.Externalizable`; any of the interfaces defined by the `javax.ejb` package.			
Local	success	success	
Remote	success	success	
5d The same business interface cannot be both a local and a remote business interface of the bean. It is also an error if the `Local` and/or `Remote` annotations are specified both on the bean class and on the referenced interface and the values differ.			
Local			
`@Local` and `@Remote` on the BC	fail	success	GlassFish ignores these inconsistent annotations.
`@Local` on the BC, `@Remote` on the BI	fail	success	GlassFish ignores these inconsistent annotations.
`@Local` on the BI, `@Remote` on BC	fail	success	GlassFish ignores these inconsistent annotations.
`@Local` and `@Remote` on the BI	fail	success	GlassFish ignores these inconsistent annotations.
Remote			
`@Local` and `@Remote` on the BC	success	success	
`@Local` on the BC, `@Remote` on the BI	success	success	
`@Local` on the BI, `@Remote` on BC	fail	success	GlassFish ignores these inconsistent annotations.
`@Local` and `@Remote` on the BI	success	success	
5e While it is expected that the bean class will typically implement its business interface(s), if the bean class uses annotations or the deployment descriptor to designate its business interface(s), it is not required that the bean class also be specified as implementing the interface(s).			
Local	success	success	
Remote	success	success	

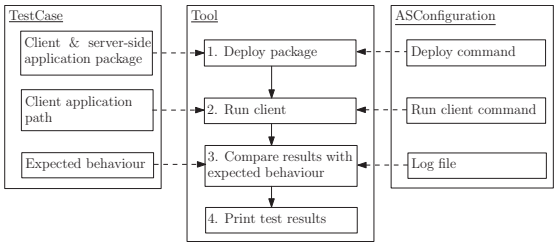

Figure 2: The tool supporting conformance testing.

this behaviour should succeed to indicate requirement compliance. Similarly, when a requirement prohibits certain behaviour, the server-side test application implementing this behaviour should fail to indicate requirement compliance. In any case, the server-side test application should have a test method logging a message upon invocation. *Second*, a client application for the server-side application is designed. The client application invokes the server-side test method which logs a message upon invocation. *Then*, the client application and server-side test application are packaged in a portable archive which is deployed on an application server, and the client application is run. *Finally*, the test results are compared with the expected behaviour. Test results include the messages logged by the server-side application, as well as error messages reported by the application server itself.

We extend this approach by treating test cases and application servers independently of each other. To facilitate application of the approach we have implemented a tool (Figure 2) which automatically deploys, runs, and analyses results for a set of test cases and a set of application servers. A *TestCase* consists of the packed client and server-side applications, a path to the client application and the expected behaviour. An *ASConfiguration* (application server configuration) consists of a deployment command, a command which extracts and runs the client application from the package and a path to the server's log file. For all *TestCases* and all *ASConfigurations* steps 1 to 4 are performed.

Since test cases do not rely on any application server they must be portable. In EJB, the portable way of distributing applications is by packaging the application in a portable archive such as a Web Archive (WAR) or an Enterprise Archive (EAR).

4 MOTIVATING EXAMPLE

To illustrate the approach we show its application for requirement **5d** in Table 1:
The same business interface cannot be both a local and a remote business interface of the bean. It is also an error if the `Local` *and/or* `Remote` *annotations are specified both on the bean class and on the referenced interface and the values differ.*

Corresponding to the first step of the approach, we first develop the server-side application which does not adhere to the given requirement. For the discussion of the example we focus on violating the requirement by means of annotations. The requirement distinguishes between the locations where a business interface can be declared: on the bean class or on the business interface. This gives rise to four distinct ways to construct a server-side application by means of annotations such that the first part of the requirement is violated: both the `Local` annotation and the `Remote` annotation are declared on the bean class; the `Local` annotation is declared on the bean class while the `Remote` annotation—on the business interface; the `Local` annotation is declared on the business interface while the `Remote` annotation—on the bean class; both the `Local` annotation and the `Remote` annotation are declared on the business interface.

For the sake of an example, we continue with the case where the `Local` annotation is on the bean class and the `Remote` annotation is on the business interface. The business interface `BI_5d` is both a local and a remote interface of the session bean `SB_5d` bean: the `Local` annotation on the bean class exposing the referenced as a local business interface, while the interface itself is annotated with `Remote` exposing the interface as a remote business interface. This clearly violates requirement **5d**.

```
@Remote
public interface BI_5d{
  public void testMethod();
}

@Stateless
@Local(BI_5d.class)
public class SB_5d implements BI_5d{
  public void testMethod(){
  System.out.println("testM invoked");
  }
}
```

Next, we develop the client applications for this session bean according to the second step of the approach. Requirement **5d** mentions both the local and the remote business interfaces. Hence, we develop both a local client and a remote client obtaining eight distinct test cases in total. We obtain a reference to the business interface by means of dependency injection through the `@EJB` annotation.

The *local client* `LC_5d` is a web application extending `javax.servlet.http.HttpServlet` and residing in the same jar file as the bean application. The `doGet` method is invoked when the server re-

ceives a HTTP GET request. The *remote client* RC_5d is a plain Java class:

```
public class LC_5d extends HttpServlet{
  @EJB private BI_5d bi;

  protected void doGet(...)
      throws ServletException{
     bi.testMethod();
  }
}

public class RC_5d{
  @EJB static private BI_5d bi;
  public static void main(...){
    bi.testMethod();
  }
}
```

We package the client application together with the server-side application and deploy the package onto the GlassFish 3.1.2 application server. The local client application and server-side application are packaged in a Web Archive (WAR). The remote client application and server-side application are packaged together in a Enterprise Archive (EAR).

Next, we run the client applications corresponding to the fourth step of the approach. The local client is run by submitting a HTTP GET request to the server. The remote client is run through the *appclient* program which is required to be present for any EJB 3.1 application server.

Finally, we compare the test results with the expected behaviour as described in the final step of the approach. Since the requirement **5d** prohibits certain behaviour ("It is also an error…") if the server-side application conforms to requirement **5d**, the test method should not be invoked. It turns out that deployment fails in the remote client case, therefore, the test method is not invoked. However, in the local client case, we find evidence of the invocation of the test method, *i.e.*, "testM invoked" is present in the server log. Hence, GlassFish passes the test for the remote client, but fails the test for the local client.

5 CASE STUDY

To answer **RQ2** and **RQ3** we apply our approach on a larger scale.

5.1 Setup

We apply the approach to two "Java EE 6 Full Profile"-compatible application servers: GlassFish 3.1.2, and JBoss AS 7.1.1. Both servers are tested for conformance to the requirements for the Session bean's business interface. The exposure of client view interfaces through the deployment descriptor is not considered. Hence, only the exposures of client view interfaces through annotations is studied. Additional client views and exposure of client views through the deployment descriptor will be considered in a follow-up study.

A session bean is either *stateful*, *stateless* or *singleton*. In our experiments we consider stateless session beans only, and the local and remote client view interfaces of these beans. The requirements for the session bean's business interfaces are shown in Table 1. We test every requirement with both a Remote and a Local client whenever the requirement does not explicitly mention the client view it applies to.

5.2 Results

Answering **RQ2** Table 1 presents the results of our requirements tests. The *Requirement* column refers to Table 1 with the client type used between parentheses. In the Setup variation column variations of a test case's setup are shown. We abbreviate *session bean's class* to *BC* and *session bean's business interface* to *BI*. Indication *fail* means that the test application provides evidence that the application server does not adhere to the specific requirement, *success*—that the test application adheres to the specific requirement with respect to the application server.

Of the 21 test GlassFish 3.1.2 fails at eight tests corresponding to requirements **1**, **5b**, and **5d**. The JBoss AS 7.1.1 application server fails at two tests corresponding to requirement **1**. In the case of requirements **1** and **5d** GlassFish 3.1.2 is inconsistent: its conformance depends on whether a local or a remote client is used. In the remote case of requirement **5d** the results even show inconsistencies for different combinations of the Local and Remote annotations on bean class and/or business interface. In JBoss AS 7.1.1 no such inconsistencies are found.

5.3 Discussion

We proceed with the discussion of **RQ3**. We discuss the implications of the lack of conformance to requirements **1**, **5b**, and **5d**. To illustrate the effects of the lack of conformance on developers we quote StackOverflow questions as indications of the developers' confusion[6] and discuss a small use case for each of the requirements.

Requirement 1. The lack of conformance to requirement **1** makes developers doubt whether a busi-

[6]None of the authors has been involved in asking or answering these questions.

ness interface must or must not extend `EJBObject` or `EJBLocalObject`.[7] Prohibition of `EJBObject` and `EJBLocalObject` extension is most crucial for applications ported from EJB 2.x, as EJB 2.x beans are required to extend `EJBObject` or `EJBLocalObject`.

Use Case. Alice wants to migrate a set of remote EJB applications from JBoss AS to GlassFish. As both JBoss AS and GlassFish are "Java EE 6 Full Profile"-compatible application servers she attempts to deploy the applications on GlassFish. To her surprise the deployment on GlassFish fails while the applications deploy and run correctly on JBoss AS.

Requirement 5b. GlassFish does not conform to requirement **5b**: when a bean class has multiple interfaces it allows implicit designation of all implemented interfaces as Local or Remote. As a results, confused developers ask *Can an EJB bean implement multiple interfaces?*[8] Moreover, the portability of the application is lost.

Use Case:. Bob is a novice EJB developer who recently became proficient with GlassFish. Working with GlassFish Bob has learned that it is possible to expose all interfaces of a bean as local business interfaces by adding the `Local` annotation with no arguments, and used this to his advantage. Charlotte, Bob's senior, wants to gain more understanding of the application Bob is developing by calculating business-method-based metrics (Roubtsov et al., 2013). The metric tool she uses presents alarming results when analysing Bob's application. After a while, they find out that the metric tool, designed based on the EJB specification, does not detect the expected number of business methods. As GlassFish exposes more business interfaces than the requirements prescribes, the calculated metrics are invalid.

Requirement 5d. Requirement **5d** states that business interface should not be annotated with both the `Local` and `Remote` annotations. However, this inconsistent annotation is erroneously accepted by GlassFish. Access via a local client seems to completely disrespect the requirement. In the remote case adherence even depends on where the `Local` and `Remote` annotations are specified. Additionally, developers may feel that a single business interface for both local and remote access is more convenient [9]. Nevertheless, local interfaces pass objects by reference and remote interface pass objects by value. Evaluation of objects becomes ambiguous when an interface is both local and remote. Hence, it is important to separate local and remote interfaces.

Use Case. Derek has developed a local EJB application which is to be deployed on GlassFish. The business interface of this application is annotated as `Local`. Derek receives a request for remote access to the functionality of a bean implementing the business interface. Thus, Derek decides to expose the functionality of the bean by annotating the bean with the the `Remote` annotation. After successfully testing the application Derek re-factors his code such that both annotations are on the bean class. Testing the application again reveals that the remote clients can no longer access the bean for no apparent reason.

The three StackOverflow questions referenced above have been answered after 2254, 166 and 80 hours, *i.e.*, significantly *longer* than the median time of 11 minutes reported for StackOverflow (Mamykina et al., 2011) or even the median time of 47 minutes reported for a subcommunity of StackOverflow dedicated to R, a popular statistics software (Vasilescu et al., 2014). This suggests that questions pertaining to conformance (or lack thereof) of the EJB 3 application servers to the EJB specification are more complex than the "average" StackOverflow questions.

6 RELATED WORK

Portability problems in EJB have been identified already for EJB 1.0 (Dorda et al., 1999): the portability problems have been caused by the vagueness of the EJB 1.0 specification. Portability issues in EJB 2.0 are discussed in (Roeser, 2005), and are attributed to different deployment descriptors and container/server configurations amongst application servers (Krastev and Galletly, 2003). Unfortunately, in both cases discussion stays at the anecdotal level: evidence of portability problems is provided, but the evidence is obtained by manual inspection. Therefore, completeness of the approach or its automation remain outside the discussion. As opposed to these results we present a structured approach based on the EJB-specification for testing portability of application servers. While in the case study we focus on the most recent EJB 3.1 specification and two application servers (GlassFish and JBoss AS), the methodology can be applied to other versions of the specification or other application servers.

Outside of the portability domain several *discrepancies between the EJB 3.1 specification and the application server GlassFish*, the reference implementation, were found in (Serebrenik et al., 2009). However, this research focused on sequence diagram reconstruction for EJB-based applications with interceptors and also merely reported on the discrepancies rather than presenting an approach for the systematic

[7] http://goo.gl/awxc3G.
[8] http://goo.gl/R8iDKT.
[9] http://goo.gl/A8FbTJ.

discovery of those. As opposed to reconstructing a sequence diagram for one business method (Serebrenik et al., 2009), subsequent research on reverse engineering sequence diagrams for EJB-based applications with interceptors (Roubtsov et al., 2011; Roubtsov et al., 2013) advocated interceptor-based metric (scenario depth) and large-scale empirical studies of the interceptor use. Specifically, for large-scale empirical studies of the use of business method interceptors one first has to identify all business methods within a given project as only invocations of business methods can be intercepted by business method interceptors. Identification of business methods, however, requires checking whether a business interface adheres to the business interface requirements shown on Table 1. As shown in Table 1 adherence to requirements, and hence, the distribution of scenarios' depth is implementation dependent. Hence, when comparing the way interceptors are used in a given system with benchmark systems one should take the deployment application server into account.

7 CONCLUSION

In this paper we presented a testing approach for verifying requirements for EJB applications and EJB application servers with focus on portability. Testing portability required a novel approach to recovering information from EJB application servers. Using the approach developed we have established discrepancies between the application server behaviour as prescribed by the specification and as implemented in two popular "Java EE 6 Full Profile"-compatible EJB application servers, GlassFish and JBoss. Lack of conformance to the EJB specification compromises the portability of the EJB applications, deviates from the portability philosophy of Java, leads to unexpected behaviour. Furthermore, lack of conformance may cause semantic differences between application servers, limiting program understanding and hindering the learning process of novice EJB developers.

A number of directions can be considered as *future work*. The first direction is related to application of the approach developed to additional requirements and application servers, and using the information obtained. Application of the approach to additional requirements and application servers will allow us to obtain a more comprehensive picture of portability in EJB 3.1. Analysis tools and techniques based on the EJB specification such as (Sutii et al., 2013) can be adapted to specifics of different application servers.

Second, the approach itself can be fully automated by generating test cases based on formally specified requirements.

REFERENCES

Dorda, S. C., Robert, J., and Seacord, R. (1999). Theory and Practice of Enterprise JavaBean™: Portability. Technical Report CMU/SEI-99-TN-005, CMU.

EJB 3.1 Expert Group (2009). *EJB 3.1 Expert Group. JSR-318 Enterprise JavaBeans, Version 3.1*.

Hamilton, M. A. (1996). Java and the shift to net-centric computing. *Computer*, 29(8):31–39.

International Software Testing Qualifications Board (2011). *Certified Tester Foundation Level Syllabus*. International Software Testing Qualifications Board. version 2011.

Kounev, S., Weis, B., and Buchmann, A. (2004). Performance tuning and optimization of J2EE applications on the JBoss platform. *Journal of Computer Resource Management*, 113:40–49.

Krastev, A. and Galletly, J. (2003). Do we really need EJB? In *CompSysTech*, pages 190–195. ACM.

Mamykina, L., Manoim, B., Mittal, M., Hripcsak, G., and Hartmann, B. (2011). Design lessons from the fastest Q&A site in the West. In *CHI*, pages 2857–2866. ACM.

Matena, V. and Harper, M. (1998). *Enterprise JavaBeans*. version 1.

Roeser, T. (2005). Portierbarkeit von J2EE-basierten Applikationen am Beispiel des mobilen Gedächtnishilfesystems MEMOS.

Roubtsov, S. A., Serebrenik, A., Mazoyer, A., and van den Brand, M. G. J. (2011). I2SD: Reverse Engineering Sequence Diagrams from Enterprise Java Beans with Interceptors. In *SCAM*, pages 155–164. IEEE.

Roubtsov, S. A., Serebrenik, A., Mazoyer, A., van den Brand, M. G. J., and Roubtsova, E. (2013). I2SD: Reverse Engineering Sequence Diagrams from Enterprise Java Beans with Interceptors. *IET Sofware*, 7(3):1–17.

Serebrenik, A., Roubtsov, S. A., Roubtsova, E. E., and van den Brand, M. G. J. (2009). Reverse engineering sequence diagrams for Enterprise JavaBeans with business method interceptors. In *WCRE*, pages 269–273. IEEE.

Sutii, A., Roubtsov, S. A., and Serebrenik, A. (2013). Detecting dependencies in enterprise javabeans with squavisit. In *WCRE*, pages 485–486. IEEE.

Vasilescu, B., Serebrenik, A., Devanbu, P. T., and Filkov, V. (2014). How social Q&A sites are changing knowledge sharing in open source software communities. In *CSCW*. ACM.

Xian, F., Srisa-an, W., and Jiang, H. (2006). Investigating throughput degradation behavior of java application servers: a view from inside a virtual machine. In *PPPJ*, pages 40–49. ACM.