

# An Investigation of the Transactional Scope of a Persistence Context

Kevin Kavanagh

Thesis presented for the degree of MSc in Software Engineering to the  
Department of Software Engineering, Athlone Institute of Technology

Supervisor: Enda Fallon

# Abstract

Transactions are used in almost every application in every industry today whether it's e-commerce, telecommunications, cloud computing etc. Java Enterprise Applications (JEAs), built using J2EE and or newer JEE technology provide an n-tiered distributed environment which are widely used in the industry. In order to exploit enhanced technology features, it is common for organisations to routinely update their existing applications. Then aim for system architects is to gain new features while maintaining backwards compatibility.

This work illustrates a number of backward compatibility issues in the Enterprise Java Bean (EJB) architecture. In particular it highlights that a transactional scope problem of a persistence context can occur in a complex transaction pattern when using the new EJB3.x technology. This interoperability issue occurs when migrating applications from EJB2.x to EJB3.x. The issue occurs when the transactional context spans multiple JEE application servers. An experimental evaluation highlights that EJB3.x is not completely backwards compatible. A corrective system architecture is developed and evaluated as part of this work.

# Table of Contents

## Contents

1. Project Introduction .....	5
1.1 Problem Statement .....	5
1.2 Overview of Proposed Solution .....	5
1.3 Contributions Arising from the Work.....	7
1.4 Thesis Outline.....	7
2 Literature Review .....	9
2.1 N-tier/distributed application.....	9
2.2 J2EE .....	10
2.3 JEE5 .....	11
2.4 JEE6 .....	16
2.5 JSF.....	16
2.6 Transactions .....	16
2.7 ORB.....	24
2.8 CORBA .....	24
2.9 JPA.....	25
2.10 Related studies.....	27
3 System Design .....	33
3.1 Requirements Analysis.....	33
3.2 System Architecture.....	34
3.2.1 System Configuration.....	34
3.2.2 EAR files.....	47
3.2.3 Database .....	54
3.2.4 JSF Web Client.....	55

3.3 High Level Design .....	60
3.2.1 Use Case Descriptions .....	60
3.4 Low Level Design .....	80
3.4.1 Class Diagram .....	80
3.4.2 Sequence Diagram .....	89
4 System Verification .....	101
4.1 Unit Testing .....	101
4.1.1 Unit test results for ejb-module-a .....	102
4.1.2 Unit test results for ejb-module-b .....	103
4.1.3 Unit test results for remote-api .....	105
4.2 Integration Testing using Arquillian .....	106
4.2.1 Test environment .....	108
4.2.2 Test steps .....	115
4.2.3 Test Cases .....	119
4.2.4 Test Results .....	122
5 Summary of Results and Conclusions .....	123
6 References .....	127
6.1 GitHub - Source Code .....	131

# 1. Project Introduction

## 1.1 Problem Statement

This work identifies backward compatibility issues between EJB3.x and EJB2.x issues. Specifically the problem occurs when an EJB3.x bean A is deployed on an application server A which is the only bean in the system that has access to the underlying database. This bean EJB A exposes two interfaces a local interface which is accessed by the client to trigger the use case and a remote interface that is accessed by EJB3.x bean B which is deployed in a different application server B. On the clients request EJB bean A will start a new transaction, then modify an attribute of an existing entity and then proceed to invoke EJB bean B on server B. EJB bean B will then join the same transaction and then remotely call back to EJB bean A to read the modified attribute by accessing the same persistence context. The attribute value is read and returned to server B which returns the value back to EJB bean A where it is verified to see if it is the same modified attribute value, before the transaction is finally committed. It can be seen that the EJB bean A modified attribute value in the transaction is not retrieved by EJB bean B so does not "see" the same persistence context used by EJB bean A.

## 1.2 Overview of Proposed Solution

The proposed solution encompasses a number of configuration and code changes. The solution uses configuration changes to the standalone xml of the JBOSS application servers to enable JTS over IIOP transport. The EJB 3.x beans must be configured to tell the application server that these EJB3.x beans will be using the EJB2.x client views not the EJB3.x client view by declaring so using the jboss-ejb3.xml. The EJB business method interfaces need to be changed to extend the EJBHome and EJBObject interfaces and finally these EJBObject interfaces need to be exposed via IIOP so these interface end points can be accessed by Corba.

The solution as proposed by this work will:-

- Enable JTS over IIOP so that transactions can be propagated between application servers on both JBOSS application servers. This is done by editing the application server profile “standalone-full.xml”
- Start both JBOSS application servers (one with a port offset) specifying the jvm arg *-Djboss.tx.node.id*. This will set the core environment *node-identifier* in the transaction subsystem.
- Extend the EJB 2.0 *EJBHome* interface - the *EJBHome* interface must be extended by all enterprise beans' remote home interfaces (prior to EJB 3.0 API that is), this allows us to create an EJB object. This is done by *StatelessRemoteHomeA* and *StatelessRemoteHomeB*.
- Expose the remote *EJBObject* interfaces i.e *StatelessRemoteObjectA*, *StatelessRemoteObjectB*, *StatefulRemoteObjectA* and *StatefulRemoteObjectB* via IIOP – this is done in the “jboss-ejb3.xml” in the META-INF folder of the respective *ejb* jar modules
- Edit the “jboss-ejb3.xml” in the META-INF folder of *ejb-module-a* and *ejb-module-b* to tell the application servers that the beans declared in the file are going to implement the EJB 2.1 client view not the standard EJB 3.1 client view.

Once the steps above are done then the *Ejb2x\_StatelessA*, *Ejb2x\_StatelessB*, *Ejb2x\_StatefulA*, *Ejb2x\_StatefulB* remote home interfaces will be exposed and accessible via the following corba's endpoint name:

Ejb2x\_StatelessA corba endpoint lookup:

[corbaname:iiop:localhost:<port>#jts/Ejb2x\\_StatelessA](#)

Ejb2x\_StatefullA corba endpoint lookup:

[corbaname:iiop:localhost:<port>#jts/Ejb2x\\_StatefulA](#)

Ejb2x\_StatelessB corba endpoint lookup:

corbaname:iiop:localhost:<port>#jts/Ejb2x\_StatelessB

Ejb2x\_StatefulB corba endpoint lookup:

corbaname:iiop:localhost:<port>#jts/Ejb2x\_StatefulB

where <port> is the default port value (3528 + port offset) for that server

### 1.3 Contributions Arising from the Work

The main contribution arising from this work is to highlight issues migrating from EJB 2.x to EJB3.x technology and to provide a solution to the problem statement in 1.1 using EJB3.1 technology. The specific contributions involve:

- how to implement EJBs using the EJB2.x and EJB3.x APIs
- how to tell the JBOSS application server that an EJB3.x will be using the EJB2.x client view implementation
- how to enable the JTS in JBoss application servers
- how to expose remote EJB interfaces via IIOP
- how to test the deployment and JEE semantics of the EJBs mentioned above using the Arquillian test framework
- how to implement a JSF web client

### 1.4 Thesis Outline

The aim of this study is to achieve a solution to the problem statement mentioned in section 1.1.

Chapter 1: Project Introduction – describes information relating to the background of the problem.

Chapter 2: Literature Review – Describes some of the concepts and technologies used in this thesis and investigate other papers that have looked at issues relating to EJB and transaction problems and issues.

Chapter 3: System Design - Describes the design of the whole application including a web based test client with information relating to the system architecture and use-case diagrams and descriptions.

Chapter 4: System Verification – This is the functional testing of the application that describe the different tests and their results.

Chapter 5: Summary of Results and Conclusions – A summary of the results is presented and conclusions are drawn.



## 2 Literature Review

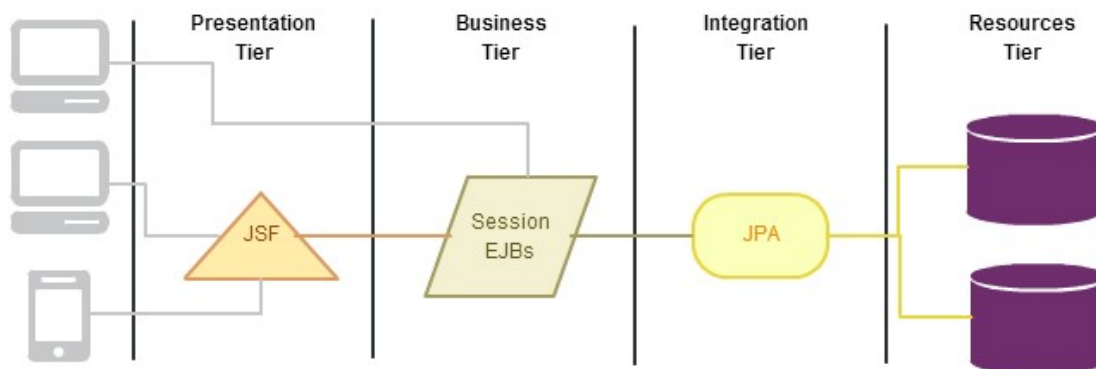
This chapter will review JEE technologies relevant to this work. This includes reviewing n-tier/distributed applications and the evolution of JEE technology from J2EE to JEE6 with an emphasis on the changes of enterprise java beans (EJBs) from EJB2.x to EJB3.x including their remote interfaces which is an important aspect of this thesis. Transaction concepts are reviewed with a focus on EJB transactions, persistence contexts, entity managers and JBOSS application server technologies. Then finally the chapter concludes with a review of a number of different related papers and articles in the area of transactions and JEE technology changes. Here is a list of some of the topics this review chapter will cover:

- N-tier (distributed) applications
- J2EE (Java 2 Platform Enterprise Edition)
- JEE (Java Enterprise Edition)
- EJB's (Enterprise Java Beans)
- Transaction Concepts
- Distributed Transactions
- JTS (Java Transaction Service)
- JTA (Java Transaction API)
- IIOP (Internet Inter Orb Protocol)
- RMI (Remote Method Invocation)
- JBOSS (application server)

### 2.1 N-tier/distributed application

An n-tier application is an application that is spread over multiple logical tiers, also known as a distributed application. Typically these logical tiers are physical tiers that are spread across a network or a JVM boundary. Applications would typically distribute functionality to achieve scalability or to place functionality close to the data that it accesses. In this study the test application uses logical tiers on two separate application servers all running in the same machine but there is no reason why it could not be executed from different physical

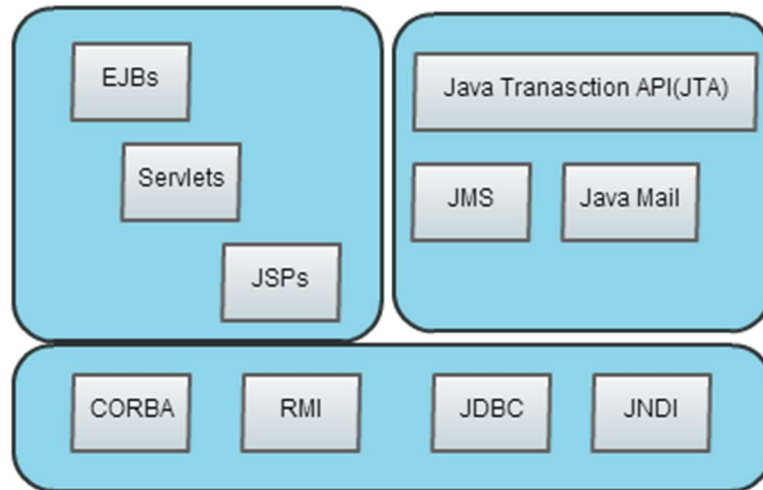
machines located in different parts of a network like the internet. See Fig.1 for an example of an n-tier or distributed application.



**Fig 1.** Logical N-tier example showing some of the APIs that could be used

## 2.2 J2EE

The popular Java 2 Enterprise Edition 1.4 (J2EE) [1] technology proposed by Sun Technologies [2] in 2003 is a collection of APIs and is designed for n-tier or distributed application development, see Fig. 2 below. These J2EE applications are hugely popular as the J2EE application server provides the facilities for scalability, reliability, distribution, transactions, security, and concurrency thus enabling developers to concentrate on business logic. Even though its regarded as a success there are some criticisms as applications can be overly complex, require a large amount of design effort as APIs are low level and require excessive amounts of “plumbing” code, uses a complex component model that is object based not object oriented based for example EJBs (EJB 2.1) are distributed components and require an application server to execute. Java EE 5 (JEE5) was the first edition designed to ease some of these criticisms of J2EE.



**Fig 2.** Summary of J2SE Architecture

One of the J2EE 1.4 APIs is JSR-153 [3] which defines the specifications for EJB2.1 which will be discussed in more detail later on in this section. J2EE provides different options for building distributed object applications

- Session Beans
- Web Services

This study will focus on session beans only and more specifically EJB2.1 beans will be used by the application developed later on.

## 2.3 JEE5

After J2EE 1.4 the next version of the technology in 2006 is called JEE5 which is specified by JSR-244 [4] and is the first edition of Java Enterprise Edition which brought with it big changes and improvements over J2EE with the primary focus being the ease of development. It reduces the amount of code to write for developers by making large use of annotations and uses the concept of “convention over configuration” where developers provide information only if required over default settings, this paradigm is popular by many frameworks like Maven [5] and the Spring Framework [6]. JEE5 contained the first EJB3 API namely EJB 3.0 which is specified by JSR 220 [7]. This new API completely restructured the

EJB model and is the main basis for the study in this paper. Here I will look at the main session and entity bean changes between EJB2.1 and EJB 3.0.

### *2.3.1 Session Bean Changes*

In EJB2.1 and earlier specifications, two interfaces – the home and local or remote business interfaces and the bean implementation class were required for each session bean. The home interface was required to extend the EJBHome or the EJBLocalHome interface and declare the lifecycle method create(). The local or remote business interface was required to extend the EJBObject or the EJBLocalObject interface and declare the business methods. The bean implementation class itself was an EnterpriseBean type and in the case of session beans, extended the SessionBean sub-interface. Callback method implementations in the bean class had to be provided so that the container could trigger them on occurrence of the appropriate lifecycle events. In addition, critical elements of the bean, including its transaction and security definition, and whether it was stateful or stateless, were defined in the associated deployment descriptors.

#### **Example of a stateful session bean using EJB2.1 specification**

```
public interface Ejb1RemoteHome extends EJBHome{
    Ejb1RemoteObject create() throws RemoteException, CreateException;
}

public interface Ejb1RemoteObject extends EJBObject{
    String getCountryOfOrigin(long id) throws RemoteException;

    void addCastToFilm() throws RemoteException;
}

public class Ejb2_V2 implements SessionBean {

    String getCountryOfOrigin(long id) throws RemoteException{ // business logic here }
```

```
void addCastToFilm() throws RemoteException{// business logic here }
```

```
public void ejbCreate(){ // bean life cycle logic here }
```

```
public void ejbActivate(){ // bean life cycle logic here }
```

```
public void ejbPassivate(){// bean life cycle logic here }
```

```
public void ejbRemove(){// bean life cycle logic here }
```

```
public void setSessionContext(SessionContext context){ (){// bean life cycle logic here }  
}
```

In the EJB3.0 specification, a session bean only has to define a business interface and a bean implementation as the home interface is removed. The business interface is a regular pojo (plain old java object) interface and it does not need to extend the EJBObject or the EJBLocalObject interface. The bean implementation class is also a pojo class and it does not implement an enterprise bean type. The development is further simplified by the fact that the declaration and the configuration in the deployment descriptor can now be defined within the java code using annotations as a metadata facility. Also default values are provided for most configurations and so it minimizes the bean specific configuration requirements. In the EJB3.0 specification it's possible to deploy a session bean without any ejb-jar.xml deployment descriptor but the deployment descriptor facility still exists if the developer prefers to use it over annotations. As the EJBs are pojo's it also means that they can be easily unit tested using testing frameworks like Junit and TestNG independently of an application server. In 2005 Oracle wrote a white paper entitled "EJB 3.0 Migration" [8] that goes through the EJB 3.0 migration process.

### **Example of a stateful session bean using EJB3.0 specification**

@Remote

```
public interface Ejb1StatefulRemote {  
    String getCountryOfOrigin(long id) throws RemoteException;  
    void addCastToFilm() throws RemoteException;  
}
```

@Stateful

```
public class Ejb1_V3_Stateful implements Ejb1StatefulRemote {  
    String getCountryOfOrigin(long id) throws RemoteException;  
    void addCastToFilm() throws RemoteException;  
}
```

It's worth knowing that EJB components which need to work with EJB2.1 and earlier specifications do not need to be written using old specifications any more as EJB3.0 beans can use metadata annotations to make them work with older clients that expect them to use the home and remote interface. This is important to point this as this is used later on in the final solution. The methods required by the older specifications are mapped to corresponding methods in the enterprise bean using EJB3.0 specification and example would be the create() method could map to a method that initializes the bean.

### ***2.3.2 Entity Bean Changes***

In EJB2.1 entity beans implemented the EntityBean interface but in EJB3.0 they are just plain pojos with an Entity annotation. In EJB3.0, the Persistence API [9] replaces using the EntityBean interface. The Persistence API defines metadata annotations to define persistence and relationship criteria for object-relational mapping concepts. The home and local interfaces are again not required in EJB3.0. In EJB3.0 the identifier or primary key for a persistent entity is defined using the Id annotation and the Persistence API defines the

EntityManager where the lifecycle management, like creation and removal, and searching of a persistent entity is done using the `persist()`, `remove()` and `find()` method calls on the EntityManager class. The Java Persistence API is used by the test application in this paper to model entities that are persisted in a database.

#### **Example of an entity bean using EJB2.1 specification**

```
public class SomeEntityBean implements EntityBean {
    public void setEntityContext(EntityContext ctx) { context = ctx; }
    public void unsetEntityContext() { context = null; }
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void ejbLoad() {}
    public void ejbStore() {}
    public void ejbRemove() {}
    // plus other getter/setter methods
}
```

#### **Example of an entity bean using EJB3.0 specification**

```
@Entity
@Table(name="Cast")
public class Cast implements Serializable{
    @Id
    private long id;

    String leadActor;
    public String getLeadActor() {
        return leadActor;
    }

    public void setLeadActor(String leadActor) {
        this.leadActor = leadActor;
    }
}
```

```
}  
public long getId() {  
    return id;  
}  
public void setId(long id){  
    this.id = id;  
}  
}
```

## 2.4 JEE6

The second release of JEE is known as JEE6 in 2009 and is specified by JSR-316 [10] and aimed to further simplify application development over JEE5 and by introducing some newer APIs like EJB 3.1 [11], JSF 2.0 [12] and JPA 2.0 [13]. EJB3.1 had some further improvements like a simplified local view for session beans, singleton session beans, automatically created EJB timers, asynchronous session bean invocations and a portable global JNDI name syntax for looking up EJB components. The changes from JEE5 to JEE6 are not as dramatic as from J2EE to JEE especially in the area of sessions beans as this thesis will show. The test application developed as part of this thesis uses JEE6.

## 2.5 JSF

Java Server Faces 2.0 (JSF) [12] is a technology for building server side user interfaces for java based web applications. JSF is used by the test application in this paper to create a test client that is accessed by a web browser.

## 2.6 Transactions

As this thesis is based on a complex transaction use case it's important to understand some transaction concepts.



### *2.6.1 The ACID Properties of transactions*

Transactions are characterized by four properties known as the ACID properties:

**Atomicity** – for a transaction to be atomic all transactions participants must make the same decision, meaning they either all commit or they all roll back.

**Consistency** – this means that written to a database is guaranteed to be valid data. The database must always be in a consistent state. One example of an inconsistent state would be a field in which half of the data is written before an operation aborts. A consistent state would be if all the data were written, or the write were rolled back when it could not be completed.

**Isolation** - Isolation means that data being operated on by a transaction must be locked before modification, to prevent processes outside the scope of the transaction from modifying the data.

**Durability** - Durability means that in the event of an external failure after transaction members have been instructed to commit, all members will be able to continue committing the transaction when the failure is resolved. This failure may be related to hardware, software, network, or any other involved system.

This means that a transaction is an indivisible unit of work that usually contains a number of operations that either must all be completed together or not at all. A transaction can only end in two ways, either it is committed or rolled back. When a transaction commits, the data modifications made by its operations are saved. If a statement within a transaction fails, the transaction rolls back, undoing the effects of all statements in the transaction. Transactions can be local or global.

### *2.6.2 Transaction Participant*

A Transaction Participant (TP) is any process within a transaction which has the ability to commit or rollback state. This could be a database, another application or another EJB for

example. In the context of this thesis the database and the EJBs on both JBoss 7 applications servers [14] are transaction participants in the same transaction context.

### ***2.6.3 Transaction Scope***

The scope of a transaction is defined by a transaction context that is shared by the participating objects or to put it more plainly it means those objects that share the same transaction context are all part of the same transaction scope. As it is an important aspect of this thesis as we focus on the transactional scope of the persistence context (or entity manager) in particular.

### ***2.6.4 Java Transaction API***

The Java Transaction API (JTA) is specified by JSR-907 [16] and is a specification for using transactions that allows access to resource managers in a uniform manner. The JTA is an API that allows you to demarcate transactions in a manner that is independent of the transaction manager (TM) implementation. A JTA transaction is controlled by the JEE transaction manager and it can span updates to multiple databases. However the JEE transaction manager does not support nested transactions meaning that it cannot start a new transaction for an instance until the previous transaction has ended (Java EE 6 Tutorial, Oracle 2013)[17].

This is not to be confused with the Java Transaction Service (JTS) [18].

### ***2.6.5 Controlling EJB transactions***

With EJBs transactions can be controlled in two ways, either by the EJB container known as container managed transactions (CMT) or it's controlled by the EJB itself, known as bean managed transactions (BMT). The test application in this thesis uses CMTs as it involves less code and is less complex and the use of BMTs would make no difference to the outcome of this thesis.

### 2.6.6 Container Managed Transactions

In an EJB with Container Managed Transactions (CMT), the container sets the boundaries of the transaction. The code does not include statements that begin and end the transaction. If no transaction demarcation is specified then CMT is the default. The transaction attribute controls the scope of the transaction and can have the following values:

REQUIRED	If a client has started a TX and calls an EJB method annotated with <code>@TransactionAttribute(REQUIRED)</code> the method executes in the clients TX, otherwise the container starts a new TX before running the method.
REQUIRES_NEW	If a client has started a TX and calls an EJB method annotated with <code>@TransactionAttribute(REQUIRES_NEW)</code> , then the container suspends the client's TX, starts a new TX, delegates the call to the method and then resumes the client's TX after the method completes. If the client does not have a TX started the container starts a new TX before running the method.
MANDATORY	If a client has started a TX and calls an EJB method annotated with <code>@TransactionAttribute(MANDATORY)</code> , then the method executes within the client's TX. If the client does not have a TX a started then the container throws a <code>TransactionRequiredException</code> .
NOT_SUPPORTED	If a client has started a TX and calls an EJB method annotated with <code>@TransactionAttribute(NOT_SUPPORTED)</code> , then the container suspends the client's TX before executing the method. After the method has completed the container resumes the client's TX. If the client does not have a TX started the container does not start a new TX before running the method.
SUPPORTS	If a client has started a TX and calls an EJB method annotated with <code>@TransactionAttribute(SUPPORTS)</code> , the method executes within

	the clients TX. If the client does not have a TX started then the container does not start a new TX before calling the method.
NEVER	If a client has started a TX and calls an EJB method annotated with <code>@TransactionAttribute(SUPPORTS)</code> , the container throws <code>RemoteException</code> . If the client does not have a TX started then the container does not start a TX before calling the method.

### *2.6.7 Bean Managed Transactions*

The other way to manage transactions in session beans is to use Bean Managed Transactions (BMT) where it is up to the code in the session bean to mark the boundaries of the transaction using JTA [16] from Sun Micro Systems.

#### **Example of a Stateless Session Bean using BMT**

```

@Stateless
@TransactionManagement( TransactionManagementType.BEAN )
public class MyMusicBean implements MyLocalMusicBean {

    @Resource
    Private UserTransation userTransaction;

    public void updateMusicCatalog(int percentage){
    ....
    try{
        userTransaction.begin();
        // do some update
        userTransaction.commit();
    }catch(DataAccessException dae){

```

```
        User.transaction.rollback();
    }
}
```

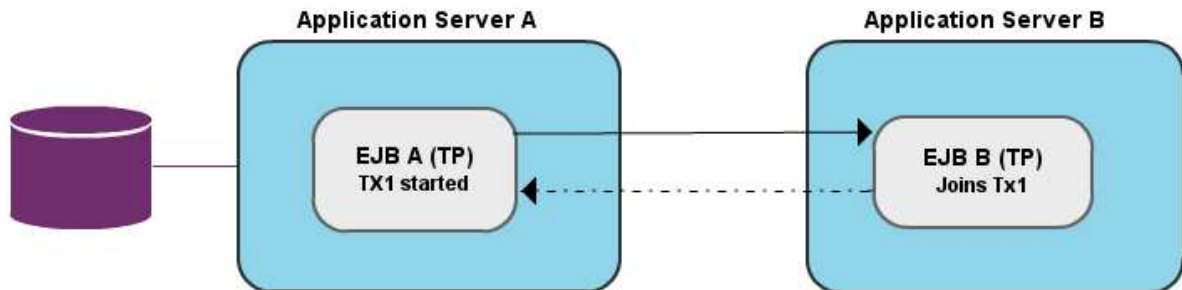
### ***2.6.8 Java Transaction Service***

The Java Transaction Service (JTS) [18], was developed by Sun Micro Systems, and is a specification for building a transaction manager that adheres to the Object Management Group (OMG) Object Transaction Service (OTS) [19] used in the Common Object Request Broker Architecture (CORBA) [20] architecture. It uses Internet Inter-ORB Protocol (IIOP) [21] to propagate JTA transactions between multiple application servers. To put it plainly, JTS is a mechanism for supporting JTA transactions when transaction participants reside in multiple JEE application servers. JTS is used by the test application developed as part of this thesis.

### ***2.6.9 Distributed Transactions***

A distributed transaction or a distributed JTA transaction is where the transaction participants are in separate applications on different application servers. If a participant joins a transaction that already exists rather than creating a new transaction, the two (or more) participants share the same transaction context and are participating in a distributed transaction. The JBoss application server supports distributed JTA transactions. The complex transaction scenario that is used in this thesis is one that I have not encountered in any paper or article to date, in that respect it could be deemed quite unique as the EJB B participant on the second applications server makes an invocation call back to EJB A on the first application server which is already a transaction participant in the same transaction context.

In the diagram below, both EJBs are transaction participants as they both have the ability to affect the transaction state. EJB B joins the transaction started by EJB A and therefore EJB B becomes part of or shares the same transaction context as EJB A.



**Fig 3.** Example of a distributed transaction where the transaction participants are EJBs

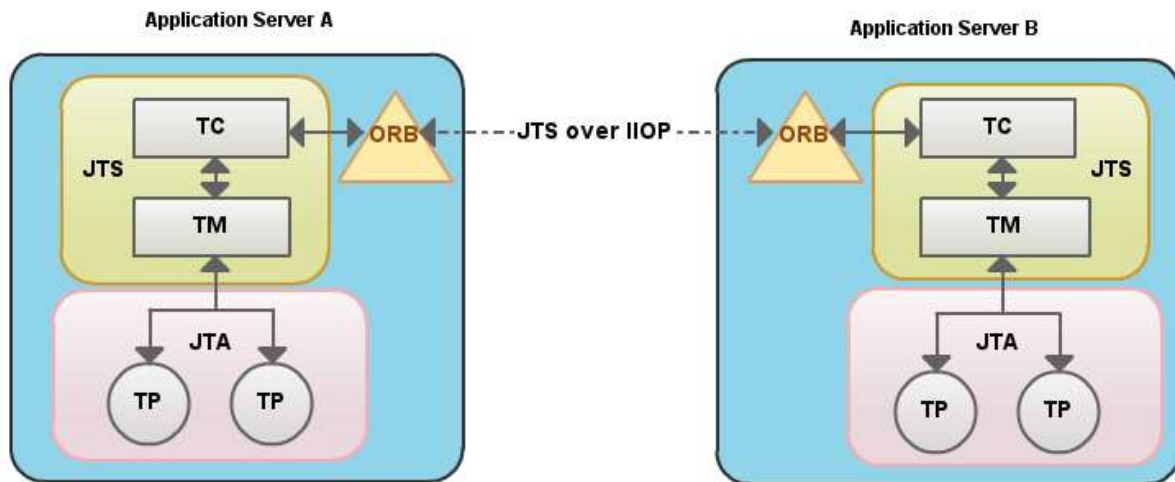
It is important to note that nested transactions are not studied as part of this thesis and are not used in the test application that is developed. Nested transactions are transactions where some participants are also transactions. This means that if a transaction is already associated with a client call when a new transaction begins the new transaction is nested within it. Nested Transactions are only supported when distributed transactions are used with the JTS API, and not part of the JTA API.

#### ***2.6.10 Transaction Manager /Transaction Coordinator***

The terms Transaction Manager (TM) and Transaction Coordinator (TC) are mostly interchangeable in terms of transactions with the JBoss application servers. The term transaction coordinator is usually used in the context of distributed transactions.

In JTA transactions, the Transaction Manager runs within each JBoss application server. The Transaction Manager tells transaction participants during the two-phase commit protocol whether to commit or roll back their data, depending on the outcome of other transaction participants. In this way, it ensures that transactions adhere to the ACID standard.

In JTS transactions, the Transaction Coordinator manages interactions between transaction managers on different servers. This communication is done so using a process called an Object Request Broker (ORB), using the CORBA communication standard.



**Fig 4.** Example of JTS transactions over IIOP between Transaction Coordinators on different servers

From an application standpoint, a JTS transaction behaves in the same way as a JTA transaction. The difference is that transaction participants and data sources reside in different containers. The application server implements the transaction manager with the Java Transaction Service (JTS) but the code doesn't call the JTS methods directly but instead it invokes the JTA methods which then invoke the lower-level JTS methods. The test application in this study is deployed on JBoss AS 7.3.4 servers which implements JTS and supports distributed JTA applications with other JBoss AS containers only.

### **2.6.11 Two Phase Commit Protocol**

As a distributed transaction is more complicated than a local transaction there needs to be a mechanism to decide when all the transaction participants should all commit or all roll back – this is where the two-phase commit protocol (2PC) is used and it involves two phases.

In Phase 1 the participants notify the transaction coordinator whether they are able to commit the transaction or must roll back.

In Phase 2, the transaction coordinator makes the decision about whether the overall transaction should commit or rollback. If any one of the participants cannot commit, the transaction must roll back otherwise the transaction can commit. The transaction coordinator directs the transactions about what to do and they notify the coordinator when they have done it. It is at this point that the transaction is finished.

## 2.7 ORB

The Object Request Broker (ORB) is a process which sends and receives messages to transaction participants, coordinators, resources, and other services distributed across multiple application servers. The main use of the ORB is a system of distributed java transactions using JTS. The ORB Portability API provides mechanisms to interact with an ORB. This API provides methods for obtaining a reference to the ORB, as well as placing an application into a mode where it listens for incoming connections from an ORB. An ORB uses a standardized Interface Description Language (IDL) to communicate and interpret messages. Common Object Request Broker Architecture (CORBA) is the IDL used by the ORB in JBoss AS 7. The ORB can be seen in Fig. 4.

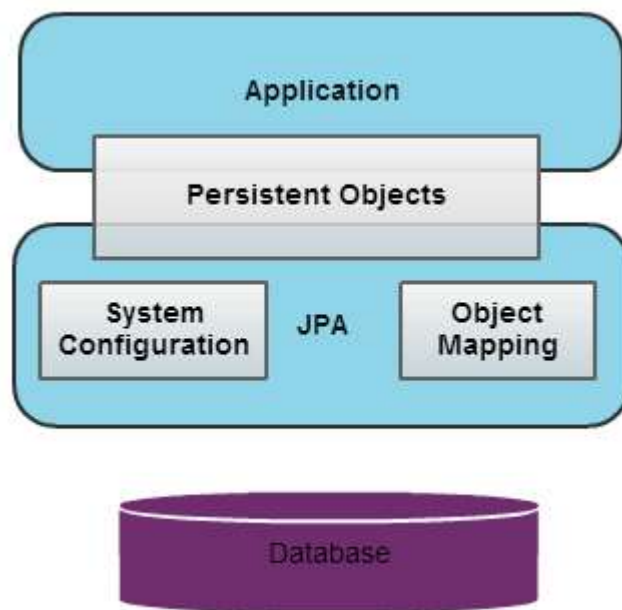
## 2.8 CORBA

Common Object Request Broker Architecture is a standard that enables applications and services to work together even when they are written in multiple, otherwise-incompatible, languages or hosted on separate platforms. CORBA requests are brokered by a server-side Object Request Broker component. The JBoss AS 7 provides an ORB instance, by means of the JacORB component subsystem. The Object Transaction Service is a service that's part of CORBA and is a set of standards maintained by the Object Management Group in order to help with cross-platform processes. In general, OTS helps to standardize the routine communications between various network components



## 2.9 JPA

The Java Persistence Architecture (JPA) , which is specified by JSR-317 [13] provides a java centric view of persistent information and uses a standardised approach to persistence. It works with POJOs which makes things easy and supports object oriented concepts like inheritance and polymorphism. JPA implements the data access layer as it provides domain objects as POJOs in response to queries and persists domain objects. JPA is a specification that is implemented by vendors and does not require an application server to execute.



**Fig. 5** The JPA architecture

### 2.9.1 Entity Manager and Persistence Context

The system level configuration is defined in the persistence.xml file. This defines one or more persistence units and when it's used in an application is known as the persistence

context. The persistence unit specifies the meta-data or schema for a database. JEE6 uses the Java Persistence 2.0 specification.

In JEE6 an EntityManager object is retrieved by injection by using the PersistenceContext annotation.

### **Example of a getting an EntityManager using PersistenceContext annotation**

```
@PersistenceContext(unitName = "FilmDatabase")  
private EntityManager em;
```

It's the EntityManager instance that maintains a persistence context. The persistence context is the collection of entity beans attached (managed) by the entity manager. The persistence context can have two scopes, either transaction-scoped persistence context (default) or extended persistence context. The persistence context scope used in the test application uses the default transaction-scoped persistence context.

#### ***2.9.2 Transaction-scoped persistence context***

In this scope the persistence context lives as long as the transaction. When the transaction completes any modifications to the attached entity bean are saved, the persistence context is destroyed and the attached entity beans become detached. Any subsequent changes to the detached entity beans are not tracked and will never be saved to the database. A typical example would be a client invokes a method on a session bean which starts a new transaction. The persistence context is associated with the transaction. At the end of the method, the transaction commits and any pending changes to attached beans are saved (flushed) to the database.

### 2.9.3 Extended persistence context

In this scope the context lives beyond the end of the transaction. Entity beans attached to an extended persistence context remain managed even after the transaction is complete. The extended persistence context is mainly used in client code or in stateful session beans which can retain the persistence context and its attached beans across multiple calls.

This table shows the typical operations of an EntityManager:

EntityManager method	Description
persist()	persist or insert a bean into the database
find()	find an entity by its primary key
clear()	detach all managed entities from a persistence context
refresh()	refresh the state of an entity from the database
remove()	remove an entity from the database

### 2.10 Related studies

There have been many studies in the area of distributed applications in J2EE and JEE technologies however few compare EJB2.x with EJB3.x. [28] illustrates how having a dual container to support existing EJB2.x and the newer EJB3.x technology was more beneficial in reducing the maintenance costs brought about by functional redundancy than the route many open source application servers were taking whereby it implemented a new EJB3.x container that is independent of the existing EJB2.x container. The proposed dual container architecture can be seen below in Fig 6.

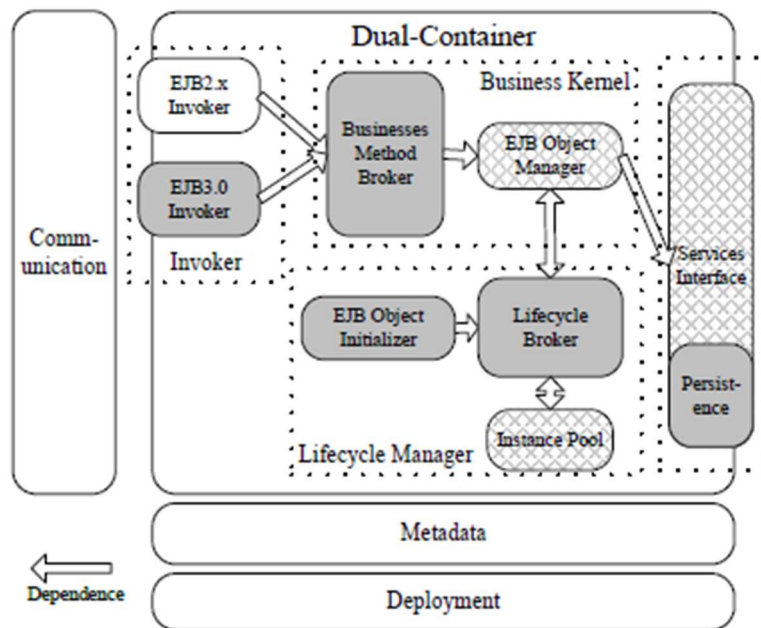


Fig 6. The dual container architecture

In [28] it recognises the deep overhaul and simplification of the EJB3.x specification and changes in contracts between the both versions. It mentions that containers must evolve to provide backwards compatibility for EJB2.x and on the other hand it must evolve to achieve compliance with the EJB3.x specification and that the JBoss AS 5 [22] chose to implement a new container to support EJB3.x which is completely independent of the existing one for EJB2.x. It proposes three principles on how to introduce the features presented by EJB3.x into the existing architecture.

The first is called the greatest-common-divisor (GCD). It means all possible overlap code between EJB2.x and EJB3.x should be identified, analysed, extracted and reused. EJB3.x inherits a number of contracts from EJB2.x e.g. runtime environment, lifecycle management and standard services. GCD reduces the possibility of introducing redundant modules.

The second principle is the least-common-multiple (LCM), this represents the abstraction of the difference between EJB2.x and EJB3.x. Abstract means that a common interface is built for the different implementations. In brief, different internal designs have the same external interface.

The final principle is transparent intrusion, an enhancement of the former principles. By inserting new modules into the logic processes transparently, the modification to the existing architecture can be reduced to the minimum as much as possible.

However since the days of JBoss AS 5 the application server has evolved and now in JBOSS AS 7 there are no longer independent containers as mentioned in [27]. The specification for the JBOSS AS 7.1 release notes [15] says that it supports *“legacy compatibility with EJB 2”* as defined in JEE 6. In JEE 7 [26] the EJB2.x specification is made optional meaning that vendors are not required to support it but currently RedHat indicate in the JSR-366 specification [27] for JBoss AS 8 that they do plan to support it. A quick look at the plans for JEE8 says *“In accordance with the pruning process defined by the Java EE 6 specification, we will consider designating the following as Proposed Optional in this release: the EJB 2.x client view APIs (EJBObject, EJBHome, EJBLocalObject, EJBLocalHome) and support for CORBA IIOP interoperability.”* Here it clearly states that it will consider proposing it as an optional feature again but what if RedHat or other vendors decide to drop support for the “optional” spec? Well that would mean that the transactional scope of a persistence scope solution as proposed by this thesis will no longer work and it will be up to the company/developer to re-design their application resulting in increased maintenance costs on top of the maintenance costs arising from the proposed solution changes as part of this thesis. It could even be worse in that the support for EJB2.x client view and CORBA IIOP interoperability may not make it into the final released version!

In [29] it mentions that J2EE has excelled at standardizing many important middleware concepts. For example, J2EE provides a standard interface for distributed transaction management, directory services, and messaging. In addition, Java 2 Standard Edition (J2SE), which underpins J2EE, provides a largely successful standard for Java interaction with relational databases. However, the platform has failed to deliver a satisfactory application programming model. It talks about EJB3.0 in that it seems to be attempting to standardize dependency injection, even though it’s unclear what benefits this will bring -especially if it results in the loss of important features, which seems inevitable. Clearly this is true as this thesis clearly shows from its results of using EJB3.x session beans the transactional scoped persistence context example shows.

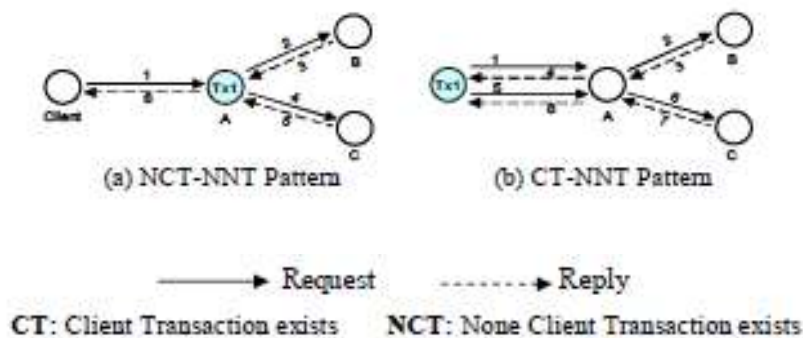
In [30] it mentions that it's widely viewed that OTS is good model in most situations for heterogeneous fault tolerant computing over networks where the concept of transactions is not only indispensable in database applications, but also useful in building robust software for mission critical applications. The paper presents an implementation of the Object Transaction Service (OTS) based on CORBA 2.0 specification. Transactional applications developed with the support of our OTS implementation are able to assure the ACID properties even in the presence of node crashes, software system failures and process hangs. The preliminary results obtained from the experiments on Sun workstations with Orbix 1.3 show that the overhead due to the OTS service is satisfactory for most applications. But this thesis asks is it good for more unusual or less common transaction problems.

OTS has shown to have some disadvantages [31] though when it comes to scaling as it relies on synchronous communication and it has been shown that for synchronous single threaded clients as the number of databases grows the response time for the workload increases. However, multi-threading adds complexity to the programming effort and failure handling and increases the cost of ownership of the applications. Transaction management always adds performance overheads to distributed applications.

There have been many papers that describe solutions that focus on fault tolerant schemes [32] [33] [34] where the assumption is made that a single client request generates exactly one transaction at the middle tier server (e.g. ejb server or web server). There are some papers [33] [35] that identify that it is quite common for multiple client requests to be encapsulated within one server transaction or that a single client request can initiate several server transactions but there has been no paper that describes a transaction pattern where by a distributed transaction re-enters a middle tier server to gain access to an object that is already part of the same transaction scope.

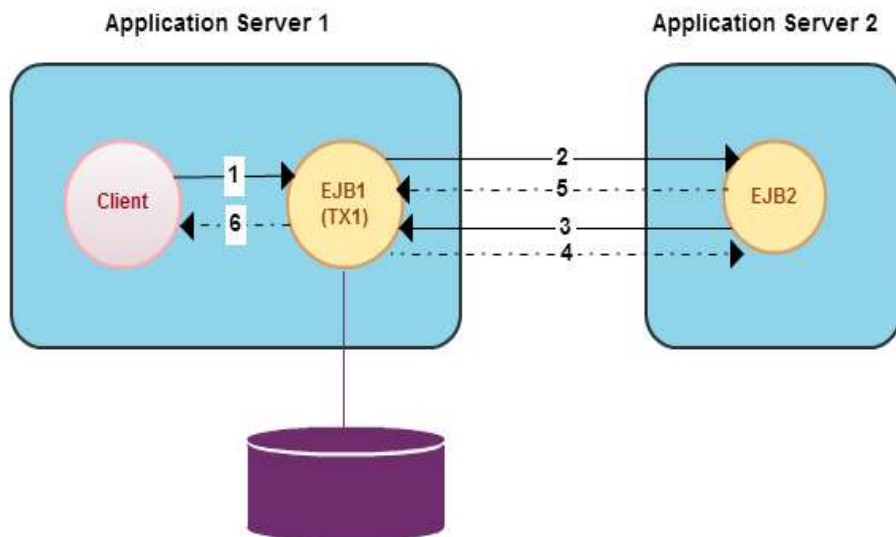
In [35] the paper points out that current transaction mechanisms assume simple transaction patterns and don't often think about complex patterns. It goes through some transaction patterns examples including nested transactions but as this paper is not concerned with nested transactions then these will not be examined here. It describes that an initiator of a transaction can be a client (web server or standalone application) or an application server

container. The patterns examples are described using stateful session beans (SFSB) that maintains conversational state and EntityBean (EB) which represents data of an application. The client can start a transaction using the UserTransaction interface [13] and this is regarded as a client transaction (CT). The container can initiate a transaction through the transaction manager interface [13]. In Fig.7 (a) is the NCT-NNT pattern where there only one transaction exists started by the container so there is no CT. After SFSB A at application server receives a request issued by the client, the container of A initiates a transaction TX1 for it. During the execution of A, it invokes EB D and EB C in the context of TX1. Both B and C will access different back-end databases before the response is returned to the client and TX1 is committed by the container of A. In the CT-NNT pattern of Fig.7 (b), there is only one CT. A client first sends a begin request to transaction manager for starting a transaction TX1 and issues a couple of requests to A in TX1, after all the request have finished the client sends a commit or abort request to the transaction manager.



**Fig. 7** Transaction Patterns

The paper describes ‘complex’ transaction patterns such as in Fig. 7 (a) and (b) above but in my opinion these scenarios are not that complex at all but are in fact quite common nowadays. In this thesis I tackle a more ‘complex’ transaction example that will show deficiencies in transaction scope handling of objects when using stateful and stateless session beans using EJB3.1 such as shown in diagram Fig 8.



**Fig. 8** A more complex cross container NCT-NNT transaction pattern

The diagram in Fig.8 also adheres to the NCT-NNT pattern but goes a step further in introducing more complexity. The client invokes step 1 on EJB1 where the container starts transaction TX1. A remote call step 2 is made to application server 2 to invoke EJB2 and TX1 is propagated as part of the call – no new transaction is started by EJB as it has the TransactionAttribute(SUPPORTS)annotation and so joins TX1 and becomes a TP, it then makes a remote invocation back to EJB1 in step 3 where it tries to use the same persistence context that is part of the same transaction context, it changes some data in the EB and then returns in step 4 and EJB2 returns back to EJB1 in step 5 at which point the transaction is committed by the container. As we will see later on this yields some interesting results when it's tested using different EJB implementations.

No articles were identified which investigated the transactional scopes of objects in any related context and I certainly did not come across any papers that looked at a complex transaction scenario as show in Fig. 8. This paper hopes to address the lack of research in this area in some way.



## 3 System Design

The complete system architecture is shown in fig 8 below. The system design illustrates the backwards compatibility issues of using pure EJB3.x beans, which uses the RMI protocol to remotely access other EJB3.x beans so does not allow an EJB3.x beans on server B to access the persistence context of server A within the same transaction. It is possible to do this in previous ejb technology such as EJB2.x, as described in the problem statement section.

### 3.1 Requirements Analysis

At a high level the main requirement can be stated like this: *“Using EJB3.x technology, show that EJB B can become a transaction participant of a distributed transaction started by EJB A from a different application server while getting access to the same transaction participants as those involved in the transaction started by EJB A before the transaction is committed to the database”*.

In a more detailed level it can be described by this sentence *“using EJB3.x EJBs, create an object within a transaction TX1 on application server A in EJB A, make a remote call to EJB B on a separate application server B which joins TX1, where EJB B then makes a remote call back to EJB A to create or modify the same object and so becomes a transaction participant in the same transaction using the same persistence context used by EJB A, before EJB A commits TX1”*. This can be seen in Fig 8.

## 3.2 System Architecture

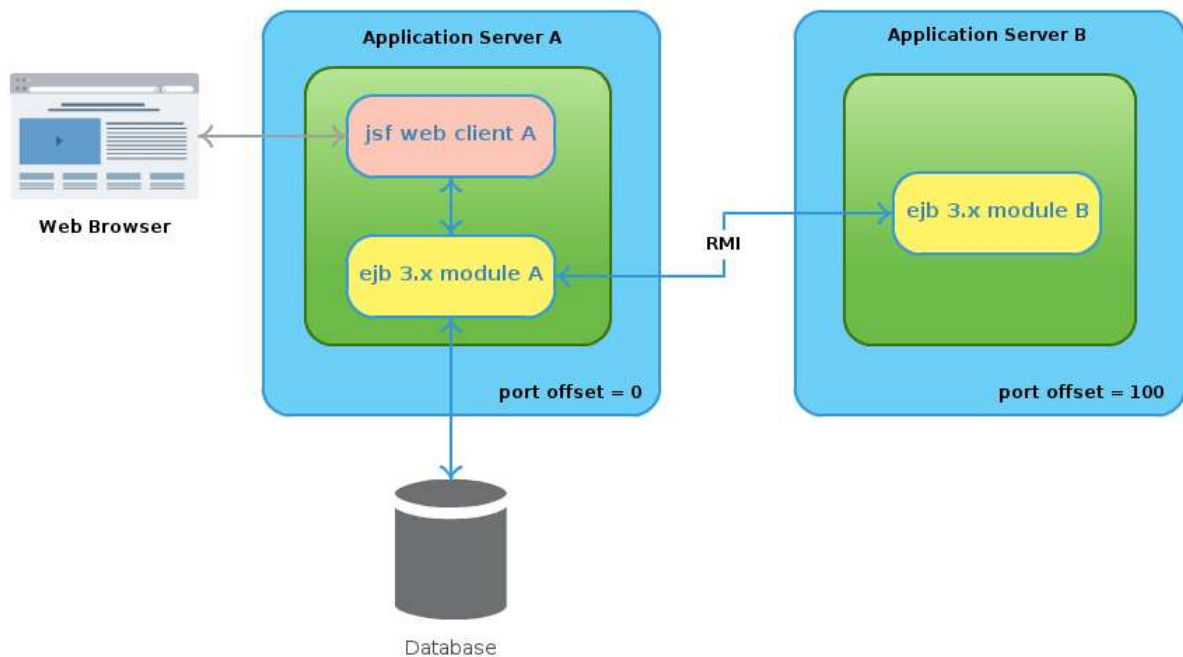


Fig. 9 The System Architecture diagram

This system solution requires two JBoss AS 7 servers A and B to be running concurrently. The JBoss servers used by this system are version "AS 7.3.4.Final-redhat-1". Because both servers are started on the same local host 127.0.0.1 then server B was started with a specified additional port offset so that the ports used by each server does not clash. The launch configurations used by both servers is as follows:

### 3.2.1 System Configuration

#### 3.2.1.1 Server A

##### Program arguments

```
-mp "/home/eeikkah/Tools/jboss-as-dist-jboss-eap-6.0.1/modules" -jaxpmodule  
javax.xml.jaxp-provider org.jboss.as.standalone -b localhost --server-config=standalone-full-  
jts-node1.xml -Djboss.server.base.dir=/home/eeikkah/Tools/jboss-as-dist-jboss-eap-  
6.0.1/standalone
```

## VM arguments

```
-server -Xms64m -Xmx1024m -XX:MaxPermSize=256m -Dorg.jboss.resolver.warning=true -
Djava.net.preferIPv4Stack=true -Dsun.rmi.dgc.client.gcInterval=3600000 -
Dsun.rmi.dgc.server.gcInterval=3600000 -Djboss.modules.system.pkgs=org.jboss.byteman -
Djava.awt.headless=true "-Dorg.jboss.boot.log.file=/home/eeikkah/Tools/jboss-as-dist-
jboss-eap-6.0.1/standalone/log/boot.log" "-
Dlogging.configuration=file:/home/eeikkah/Tools/jboss-as-dist-jboss-eap-
6.0.1/standalone/configuration/logging.properties" "-
Djboss.home.dir=/home/eeikkah/Tools/jboss-as-dist-jboss-eap-6.0.1" "-
Djboss.bind.address.management=localhost" "-Djboss.tx.node.id=1111" "-
Djboss.node.name=node1" "-Djboss.server.name=node1"
```

### 3.2.1.2 Server B

## Program arguments

```
-mp "/home/eeikkah/Tools/jboss-as-dist-jboss-eap-6.0.1-1/modules" -jaxpmodule
javax.xml.jaxp-provider org.jboss.as.standalone -b localhost --server-config=standalone-full-
jts-node2.xml -Djboss.server.base.dir=/home/eeikkah/Tools/jboss-as-dist-jboss-eap-6.0.1-
1/standalone
```

## VM arguments

```
-server -Xms64m -Xmx1024m -XX:MaxPermSize=256m -Dorg.jboss.resolver.warning=true -
Djava.net.preferIPv4Stack=true -Dsun.rmi.dgc.client.gcInterval=3600000 -
Dsun.rmi.dgc.server.gcInterval=3600000 -Djboss.modules.system.pkgs=org.jboss.byteman -
Djava.awt.headless=true "-Dorg.jboss.boot.log.file=/home/eeikkah/Tools/jboss-as-dist-
jboss-eap-6.0.1-1/standalone/log/boot.log" "-
Dlogging.configuration=file:/home/eeikkah/Tools/jboss-as-dist-jboss-eap-6.0.1-
1/standalone/configuration/logging.properties" "-
Djboss.home.dir=/home/eeikkah/Tools/jboss-as-dist-jboss-eap-6.0.1-1" "-
```

```
Djboss.bind.address.management=localhost" "-Djboss.tx.node.id=1111" "-  
Djboss.node.name=node2" "-Djboss.server.name=node2"
```

### 3.2.1.3 Setting up EJB3.x remote connections between application servers

As the EJBs of server A will invoke the EJBs of server B and the EJBs of server B will invoke those of A then the remote connections between the servers needs to be configured for this communication. This is done by adding remote connections to the relevant standalone xmls by manually editing the xmls or the *jboss-cli.sh* script can be used to make the configuration changes. The *jboss-cli.sh* script is located in the bin directory of the jboss runtime environment.

In JEE6 application servers have security turned on by default. So before the EJB remote connections can be set up the security credentials need to be configured between the application servers, then the socket bindings are created and finally the remote connections are defined. Here are the steps that were carried out to configure the EJB remote connections:

#### Setting up the users and security realm

1. First add a user was added to both application servers called "ejbuser" with password "EJBUser" . This was done using the *add-user.sh* script in the runtime bin directory.

#### **Server A and B:**

What type of user do you wish to add?

- a) Management User (mgmt-users.properties)
- b) Application User (application-users.properties)

(a): b

Enter the details of the new user to add.

Using realm 'ApplicationRealm' as discovered from the existing property files.

Username : ejbuser

Password : EJBUser

Re-enter Password : EJBUser

What groups do you want this user to belong to? (Please enter a comma separated list, or leave blank for none)[ ]:

About to add user 'ejbuser' for realm 'ApplicationRealm'

Is this correct yes/no? yes

Is this new user going to be used for one AS process to connect to another AS process?

e.g. for a slave host controller connecting to the master or for a Remoting connection for server to server EJB calls.

yes/no? yes

To represent the user add the following to the server-identities definition <secret value="RUpCVXNlcg==" />

2. Add a new security realm called "ejb-security-realm" to each application server by running the *jboss-cli.sh* script and add the password output from the command above to represent the user ejbuser:

**Server A:**

```
[standalone@localhost:9999 /] /core-service=management/security-realm=test-realm:add()
```

```
[standalone@localhost:9999 /] /core-service=management/security-realm=test-realm/server-identity=secret:add(value="RUpCVXNlcg==")
```

### Server B:

```
[standalone@localhost:10099 /] /core-service=management/security-realm=test-realm:add()
```

```
[standalone@localhost:10099 /] /core-service=management/security-realm=test-realm/server-identity=secret:add(value="RUpCVXNlcg==")
```

The security configuration is now setup for the applications servers to communicate with each other.

### Setting up the socket bindings

Next the outbound socket binding connections needed to be created for each application server. To do this the following steps were carried out:

1. Using the *jboss-cli.sh* script for server A this command was executed to create an outbound socket binding to be used to communicate with server B. Note the port offset of 100 was added to 4447 as this is the port that application server B will be running on.

### Server A:

```
[standalone@localhost:9999 /] /socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=remote-ejb:add(host=localhost, port=4547)
```

The same command was executed for server B but with no port offset as application server A is not running with an offset.

### Server B:

```
[standalone@localhost:10099 /] /socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=remote-ejb:add(host=localhost, port=4447)
```

2. Next, the remote outbound connections need to be created for both applications servers that use the socket binding created above. This was done by executing the following steps:

**Server A:**

```
[standalone@localhost:9999 /] /subsystem=remoting/remote-outbound-connection=remote-ejb-connection:add(outbound-socket-binding-ref=remote-ejb, security-realm=ejb-security-realm,username=ejbuser)
```

**Server B:**

```
[standalone@localhost:10099 /] /subsystem=remoting/remote-outbound-connection=remote-ejb-connection:add(outbound-socket-binding-ref=remote-ejb2, security-realm=ejb-security-realm,username=ejbuser)
```

**Creating the outbound connections for the socket bindings**

1. On server A a remote outbound connection was created that uses the newly created socket binding using this command:

**Server A:**

```
[standalone@localhost:9999 /] /subsystem=remoting/remote-outbound-connection=remote-ejb-connection:add(outbound-socket-binding-ref=remote-ejb, security-realm=ejb-security-realm,username=ejbuser)
```

**Server B:**

On server B a remote outbound connection was also created that use the newly created socket binding using this command:

```
[standalone@localhost:10099 /] /subsystem=remoting/remote-outbound-connection=remote-ejb-connection2:add(outbound-socket-binding-ref=remote-ejb2, security-realm=ejb-security-realm,username=ejbuser)
```

2. Then some SASL and SSL properties were set as recommended by the JBoss EJB remoting documentation:

### Server A:

```
[standalone@localhost:9999 /] /subsystem=remoting/remote-outbound-connection=remote-ejb-connection/property=SASL_POLICY_NOANONYMOUS:add(value=false)
```

```
[standalone@localhost:9999 /] /subsystem=remoting/remote-outbound-connection=remote-ejb-connection/property=SSL_ENABLED:add(value=false)
```

### Server B:

```
[standalone@localhost:10099 /] /subsystem=remoting/remote-outbound-connection=remote-ejb-connection/property=SASL_POLICY_NOANONYMOUS:add(value=false)
```

```
[standalone@localhost:10099 /] /subsystem=remoting/remote-outbound-connection=remote-ejb-connection/property=SSL_ENABLED:add(value=false)
```

This is the configuration required to enable EJB3.x remoting between different application servers. After the steps above are carried out the following output can be seen...

### Summary of the configuration changes for EJB3.x remote connections

User “ejbuser” is added to the **application-users.properties** file for both servers:

```
ejbuser=2d01290ce92f7b34b90d364a9d93415b
```

Security realm “ejb-security-realm” is added to the **standalone.xmls** for both server’s:

```
<management>
```



.....

```
<security-realms>
    <security-realm name="ejb-security-realm">
        <server-identities>
            <secret value="RUpCVXNlcg==" />
        </server-identities>
    </security-realm>
</security-realms>
</management>
```

The following socket binding configuration for “remote-ejb” is added to server A standalone.xml:

```
<socket-binding-group>
    ....
    <outbound-socket-binding name="remote-ejb">
        <remote-destination host="localhost" port="4547" />
    </outbound-socket-binding>
</socket-binding-group>
```

The following remote outbound connection “remote-ejb-connection” is added to server A standalone.xml:

```

<subsystem xmlns="urn:jboss:domain:remoting:1.1">

    <connector name="remoting-connector" socket-binding="remoting" security-
realm="ApplicationRealm"/>

    <outbound-connections>

        <remote-outbound-connection name="remote-ejb-connection" outbound-socket-
binding-ref="remote-ejb" username="ejbuser" security-realm="ejb-security-realm">

            <properties>

                <property name="SASL_POLICY_NOANONYMOUS" value="false"/>

                <property name="SSL_ENABLED" value="false"/>

            </properties>

        </remote-outbound-connection>

    </outbound-connections>

</subsystem>

```

The following socket binding configuration for “remote-ejb2” is added to server B standalone.xml:

```

<socket-binding-group>

.....

<outbound-socket-binding name="remote-ejb2">

    <remote-destination host="localhost" port="4447"/>

</outbound-socket-binding>

```

```
</socket-binding-group>
```

The following remote outbound connection “remote-ejb-connection2” is added to server B standalone.xml:

```
<subsystem xmlns="urn:jboss:domain:remoting:1.1">
    <connector name="remoting-connector" socket-binding="remoting" security-
realm="ApplicationRealm"/>
    <outbound-connections>
        <remote-outbound-connection name="remote-ejb-connection2" outbound-socket-
binding-ref="remote-ejb2" username="ejbuser" security-realm="ejb-security-realm">
            <properties>
                <property name="SASL_POLICY_NOANONYMOUS" value="false"/>
                <property name="SSL_ENABLED" value="false"/>
            </properties>
        </remote-outbound-connection>
    </outbound-connections>
</subsystem>
```

Note for the EJBs to be able to use EJB remote connections defined above the EAR files need to have a **jboss-ejb-client.properties** file added to the **META-INF** folder of their respective ears with the name of the outbound connection specified. This is covered in more detail in the next EAR files section below.

### 3.2.1.4 Enabling IIOP

JBoss comes prepackaged with different standalone xml files depending on what features you would like to use and this is the starting point for any JBoss administrator. This standalone xmls used by this system are copies of the standalone-full.xml which then were edited as needed. The Jacorb (Corba) subsystem is part of the standalone-full.xml. To enable IIOP in the application servers the Jacorb subsystem must be running and the <iiop/> element present in the ejb3 subsystem configuration. This iiop is important to the solution as this is the element that enables IIOP (corba) invocation of the EJBs. This element has two properties (1) **enable-by-default** - If this is true then all EJB's with EJB2.x home interfaces are exposed via IIOP, otherwise they must be explicitly enabled via jboss-ejb3.xml and (2) **use-qualified-name** - if this is true then EJB's are bound to the corba naming context with a binding name that contains the application and modules name of the deployment if this is false the default binding name is simply the bean name.

```
<subsystem xmlns="urn:jboss:domain:ejb3:1.3">
    .....
    <iiop enable-by-default="false" use-qualified-name="false"/>
</subsystem>
```

### 3.2.1.5 Enabling JTS

To enable the java transaction service the </jts> element needs to be added to the transactions subsystem.

```
<subsystem xmlns="urn:jboss:domain:transactions:1.2">
    .....
    <jts/>
```

```
</subsystem>
```

It is also necessary to enable the Jacorb transactions interceptor by setting it to 'on'.

```
<subsystem xmlns="urn:jboss:domain:jacorb:1.2">  
  <orb socket-binding="jacorb" ssl-socket-binding="jacorb-ssl">  
    <initializers security="off" transactions="on"/>  
  </orb>  
</subsystem>
```

### *3.2.1.6 Exposing EJB2.x home interfaces via IIOP*

With IIOP and JTS activated along with the Jacorb subsystem running it is possible then to declare using the jboss-ejb3.xml which specific EJB2.x home interfaces are exposed via IIOP. This is crucial in the outcome of this solution. The EJB2.x home interfaces are exposed by having these jboss-ejb3.xml files in the EJB modules META-INF folders.

**jboss-ejb3.xml of used by both EJB modules:**

```
<?xml version="1.0" encoding="UTF-8"?>  
<jboss:ejb-jar xmlns:jboss="http://www.jboss.com/xml/ns/javaee"  
  xmlns="http://java.sun.com/xml/ns/javaee"  
  xmlns:iiop="urn:iiop"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee
http://www.jboss.org/j2ee/schema/jboss-ejb3-2_0.xsd
```

```
http://java.sun.com/xml/ns/javaee
http://www.jboss.org/j2ee/schema/jboss-ejb3-spec-2_0.xsd
```

```
urn:iiop:jboss-ejb-iiop_1_0.xsd"
```

```
version="3.1"
```

```
impl-version="2.0">
```

```
<assembly-descriptor>
```

```
<iiop:iiop>
```

```
<ejb-name>Ejb2x_StatelessA</ejb-name>
```

```
<iiop:binding-name>jts/Ejb2x_StatelessA</iiop:binding-name>
```

```
</iiop:iiop>
```

```
<iiop:iiop>
```

```
<ejb-name>Ejb2x_StatelessB</ejb-name>
```

```
<iiop:binding-name>jts/Ejb2x_StatelessB</iiop:binding-name>
```

```
</iiop:iiop>
```

```
</assembly-descriptor>
```

```
</jboss:ejb-jar>
```

**Note** an observation that was made was that it declares that the version of these EJBs is 3.1 but that it is going to use the EJB 2.1 client view implementation. In fact looking in the jboss schema documentation “jboss-ejb3-2\_0.xsd” for this it specifies that the impl-version is fixed as 2.0 so it will in fact ignore what is entered.

```
<xs:attribute name="impl-version" type="javaee:dewey-versionType" fixed="2.0"
use="required"/>
```

### 3.2.2 EAR files

The deployment of the whole system is deployed in 2 EAR (enterprise archive) files. The deployment structure of the files ear-module-a-1.0-SNAPSHOT.ear and ear-module-b-1.0-SNAPSHOT.ear is as follows:

#### 3.2.2.1 ear-module-a-1.0-SNAPSHOT.ear content

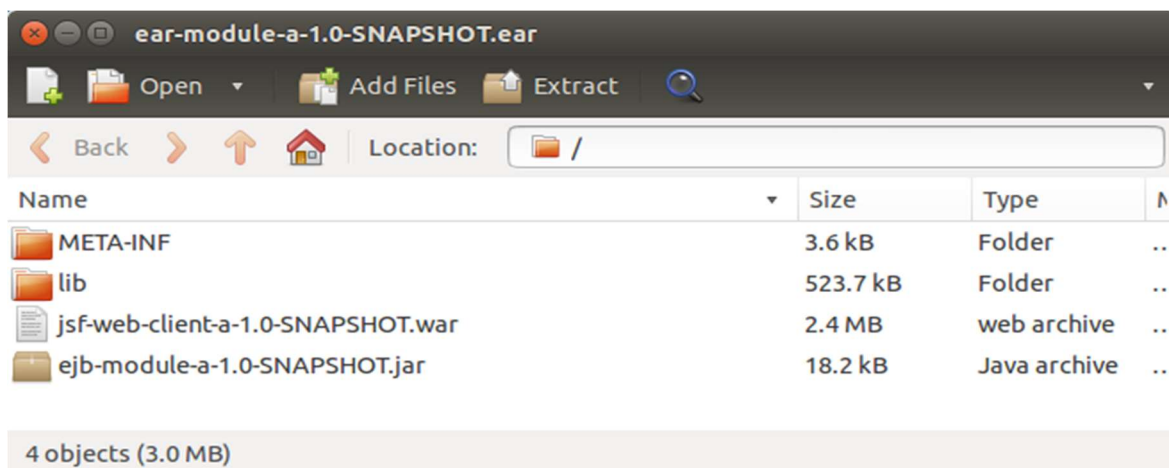


Fig. 10 ear-module-a content

- **ejb-module-a-1.0.SNAPSHOT.jar** - this contains all the ejbs and the entity classes
- **jsf-web-client-a-1.0-SNAPSHOT.war** - this contains the web application client that can be accessed from web browser

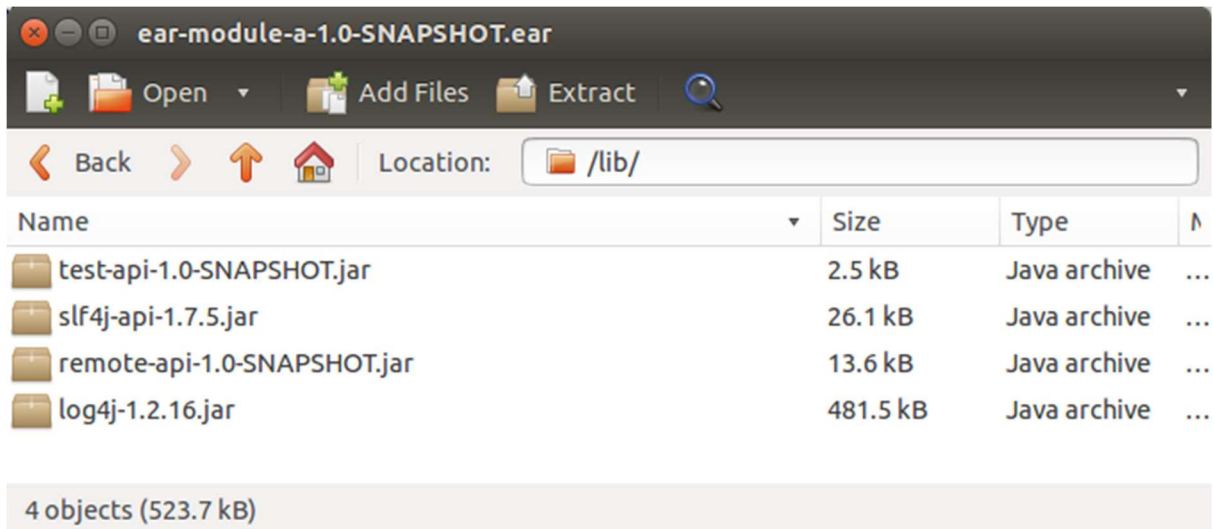


Fig. 11 ear-module-a /lib content

- **lib/remote-api-1.0-SNAPSHOT.jar** - this contains the library of remote interfaces in the system and goes in the /lib directory of the EAR file.
- **lib/test-api-1.0-SNAPSHOT.jar** - this contains the library of test interfaces in the system and goes in the /lib directory of the EAR file.

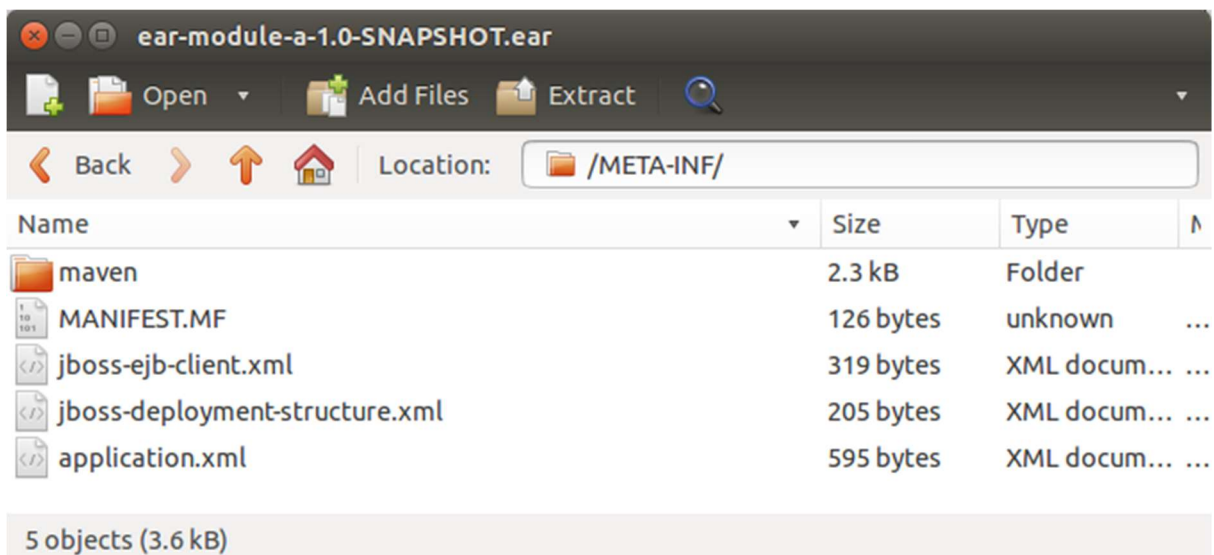


Fig. 12 ear-module-a META-INF content

- **META-INF/application.xml** - here the display name of the EAR is set to *ear-module-a*. This file also tells the server that it contains a web module with a web uri called *jsf-web-*



*client-a-1.0-SNAPSHOT.war* and that its root context is *jsf-web-client-a*. This is important when accessing the client from the browser. This file also says it contains an ejb module called *ejb-module-a-1.0-SNAPSHOT.jar*.

```
<?xml version="1.0" encoding="UTF-8"?>

<application xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/application_6.xsd" version="6">

  <display-name>ear-module-a</display-name>

  <module>

    <ejb>ejb-module-a-1.0-SNAPSHOT.jar</ejb>

  </module>

  <module>

    <web>

      <web-uri>jsf-web-client-a-1.0-SNAPSHOT.war</web-uri>

      <context-root>/jsf-web-client-a</context-root>

    </web>

  </module>

  <library-directory>lib</library-directory>

</application>
```

- **META-INF/jboss-deployment-structure.xml** - here a dependency on the SLF4J jboss module is declared to enable logging to the console.

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2">
```

```
<deployment>
    <dependencies>
        <module name="org.slf4j"/>
    </dependencies>
</deployment>
</jboss-deployment-structure>
```

- **META-INF/jboss-ejb-client.xml** - this file is used to tell the application server that there are EJB clients in this EAR that will be making remote outbound connections to other servers and it declares the name of the remote outbound connection to use as **remote-ejb-connection** as this matches with the name of the connection configured in the standalone xml file for server A. If the name of the outbound connection ref does not match with what is configured in the standalone xml then the EJB to EJB communication will fail.

```
<jboss-ejb-client xmlns:xsi="urn:jboss:ejb-client:1.0"
xsi:noNamespaceSchemaLocation="jboss-ejb-client_1_2.xsd">
    <client-context>
        <ejb-receivers>
            <remoting-ejb-receiver outbound-connection-ref="remote-ejb-connection"/>
        </ejb-receivers>
    </client-context>
</jboss-ejb-client>
```

### 3.2.2.2 ear-module-b-1.0-SNAPSHOT.ear content

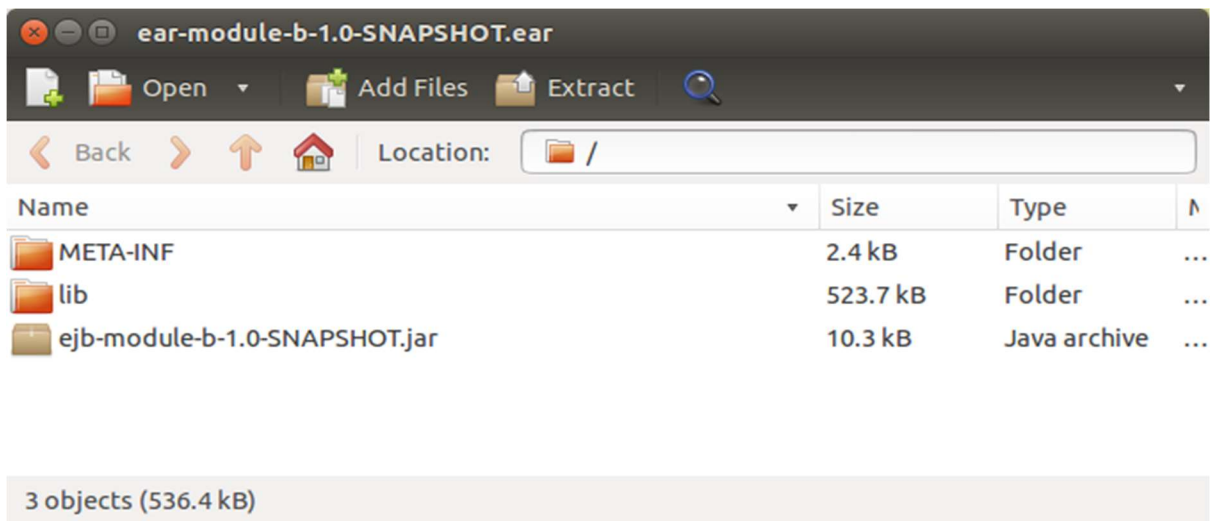


Fig. 13 ear-module-b content

- **ejb-module-b-1.0-SNAPSHOT.jar** - this contains all the ejbs and the entity classes

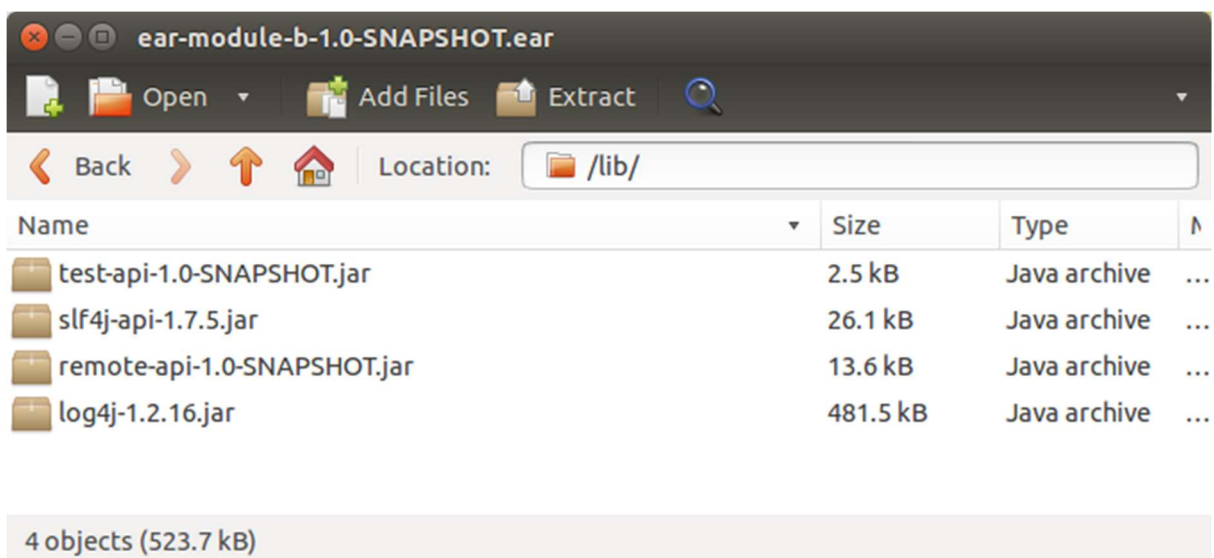
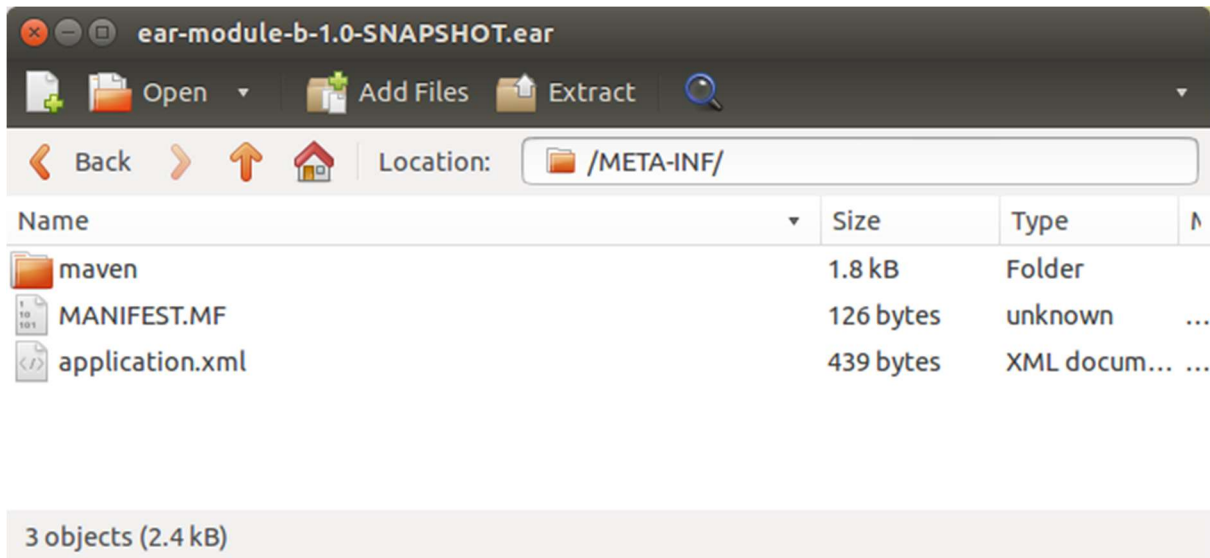


Fig. 14 ear-module-b /lib content

- **/lib/remote-api-1.0-SNAPSHOT.jar** - this contains the library of remote interfaces in the EAR file.
- **/lib/test-api-1.0-SNAPSHOT.jar** - this is only here because it contains an interface that is used in the arquillian testing



**Fig. 15** ear-module-b META-INF content

- **/META-INF/application.xml** - here the display name of the EAR is set to *ear-module-b*. This file also says it contains an ejb module called *ejb-module-b-1.0-SNAPSHOT.jar*.

```
<?xml version="1.0" encoding="UTF-8"?>

<application xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/application_6.xsd" version="6">

  <display-name>ear-module-b</display-name>

  <module>

    <ejb>ejb-module-b-1.0-SNAPSHOT.jar</ejb>

  </module>
```

```
<library-directory>lib</library-directory>
```

```
</application>
```

- **/META-INF/jboss-deployment-structure.xml** - here a dependency on the SLF4J jboss module is declared to enable logging to the console.

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2">
```

```
<deployment>
```

```
<dependencies>
```

```
<module name="org.slf4j"/>
```

```
</dependencies>
```

```
</deployment>
```

```
</jboss-deployment-structure>
```

- **/META-INF/jboss-ejb-client.xml** - similar to above this declares the name of the remote outbound connection to use as **remote-ejb-connection2** as this matches with the name of the connection configured in the standalone xml file for server B. If the name of the outbound connection ref does not match with what is configured in the standalone xml then the EJB to EJB communication will fail.

```
<jboss-ejb-client xmlns:xsi="urn:jboss:ejb-client:1.0"
```

```
xsi:noNamespaceSchemaLocation="jboss-ejb-client_1_2.xsd">
```

```
<client-context>
```

```
<ejb-receivers>
```

```
        <remoting-ejb-receiver outbound-connection-ref="remote-ejb-
connection2"/>
    </ejb-receivers>
</client-context>
</jboss-ejb-client>
```

### 3.2.3 Database

JBOSS AS 7 (EAP 6) comes shipped with an open source lightweight database called *H2* that can be very useful for development/test activities such as this thesis and which is easy to install and use. The H2 database can be configured to run in an embedded java application, run in client-server mode or configured to run as an in-memory database as I have chosen, as I have no need to persistently store objects.

To configure it as an in-memory database [36] edit the standalone.xml to add this:

```
<subsystem xmlns="urn:jboss:domain:datasources:1.1">
    <datasources>
        <datasource jndi-name="java:jboss/datasources/ExampleDS" pool-
name="ExampleDS" enabled="true" use-java-context="true">
            <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-1</connection-url>
            <driver>h2</driver>
            <security>
                <user-name>sa</user-name>
                <password>sa</password>
```

```
</security>

</datasource>

<drivers>

  <driver name="h2" module="com.h2database.h2">

    <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>

  </driver>

</drivers>

</datasources>

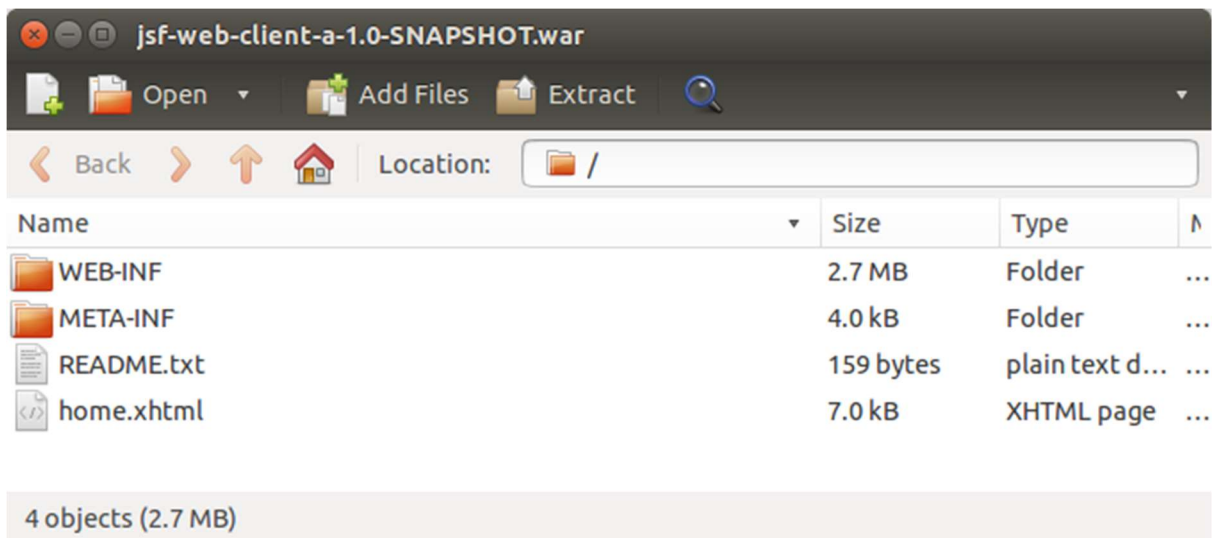
</subsystem>
```

This amount of configuration is enough for testing with JPA applications but it does not allow persisting data to the hard disk and you cannot access the H2 from the web console.

### 3.2.4 JSF Web Client

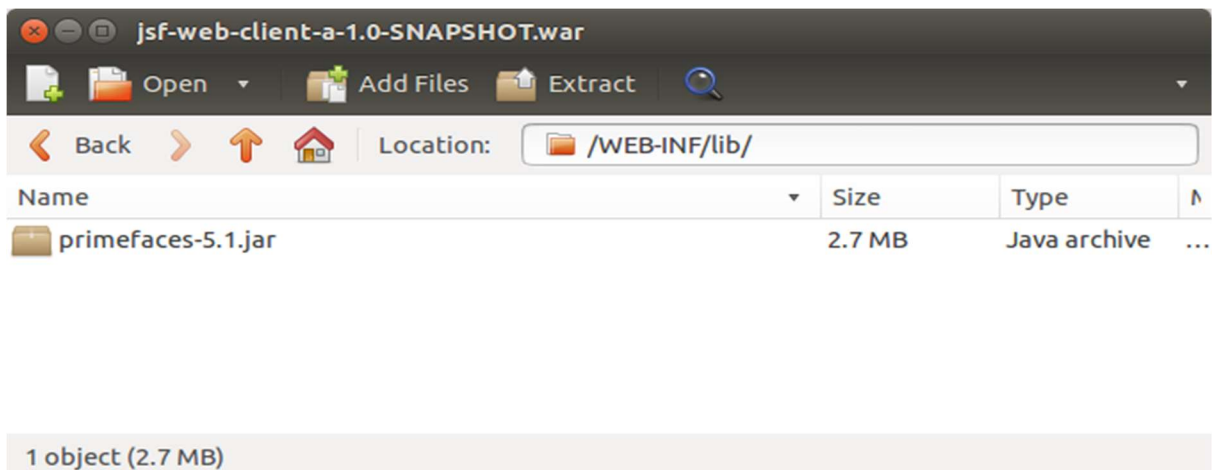
The JSF client *jsf-web-client-a-1.0-SNAPSHOT.war* is the web application that is deployed in server A and provides the client for the user to interact with the system using a web browser backed by a managed bean. JSF provides a simple mechanism that requires only a small amount of code and configuration before it can be used and that is why JSF was chosen for the client in this system.

#### 3.2.4.1 *jsf-web-client-a-1.0-SNAPSHOT.war* content



**Fig. 16** jsf-web-client content

- **home.xhtml** - this is the file that is loaded by JSF and used for the presentation part of the client as it specifies the margins, headings, layouts, buttons and the action listeners to be executed when events occur such as button presses. This page is backed by the managed bean called “tc” which is the TestCaseManager class and invokes the bean methods in this class when the buttons are pressed by the user.



**Fig. 17** jsf-web-client /WEB-INF/ content

- **primefaces.jar** - this jar is required for JSF to execute



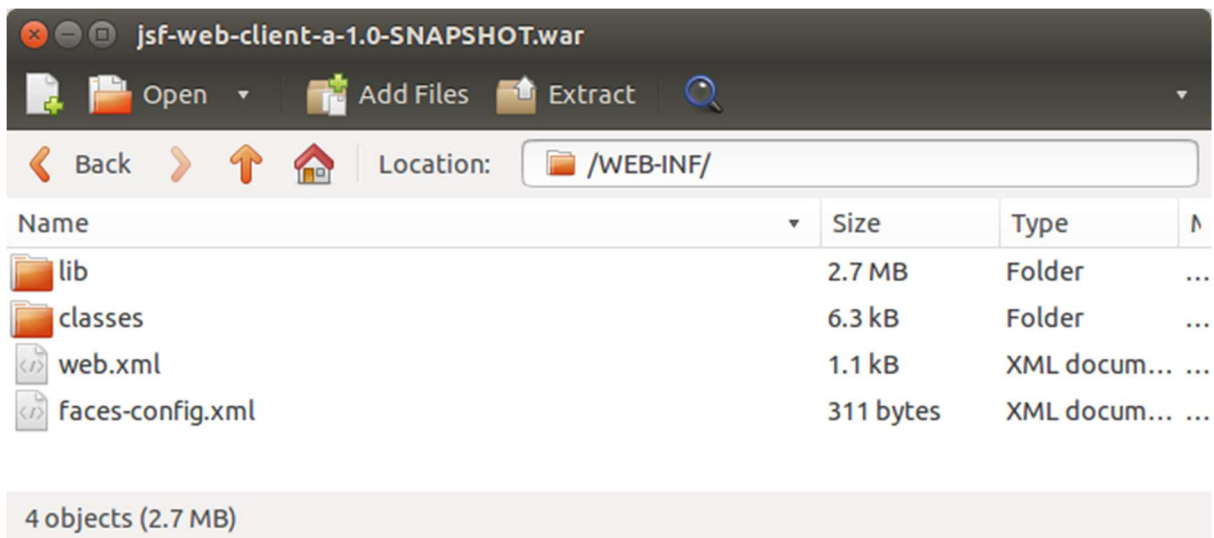


Fig. 18 jsf-web-client /WEB-INF/ content

- **faces-config.xml** - this is for JSF customizations but is not used by this application
- **web.xml** - this is used to specify the faces servlet is to be used and also what the mapping url pattern to look for is on deployment of the war file

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID"
version="3.0">
```

```
<display-name>JSFClient</display-name>
```

```
<servlet>
```

```
<servlet-name>Faces Servlet</servlet-name>
```

```
<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
```

```
<load-on-startup>1</load-on-startup>
```

```
</servlet>

<servlet-mapping>

    <servlet-name>Faces Servlet</servlet-name>

    <url-pattern>*.xhtml</url-pattern>

</servlet-mapping>

</web-app>
```

#### *3.2.4.2 Test Client*

Once deployed in the application server the test client contains a selection of buttons that the user can press to execute the different test cases that test the transactional scope of a persistence context. The format for the tests is the same whereby the user clicks the setup button followed by the execute and teardown buttons in that order. A message is displayed in the client to provide the user with the result of each button press. Below is an image of the application which can be accessed using the following url: **<http://localhost:8080/jsf-web-client-a/home.xhtml>**

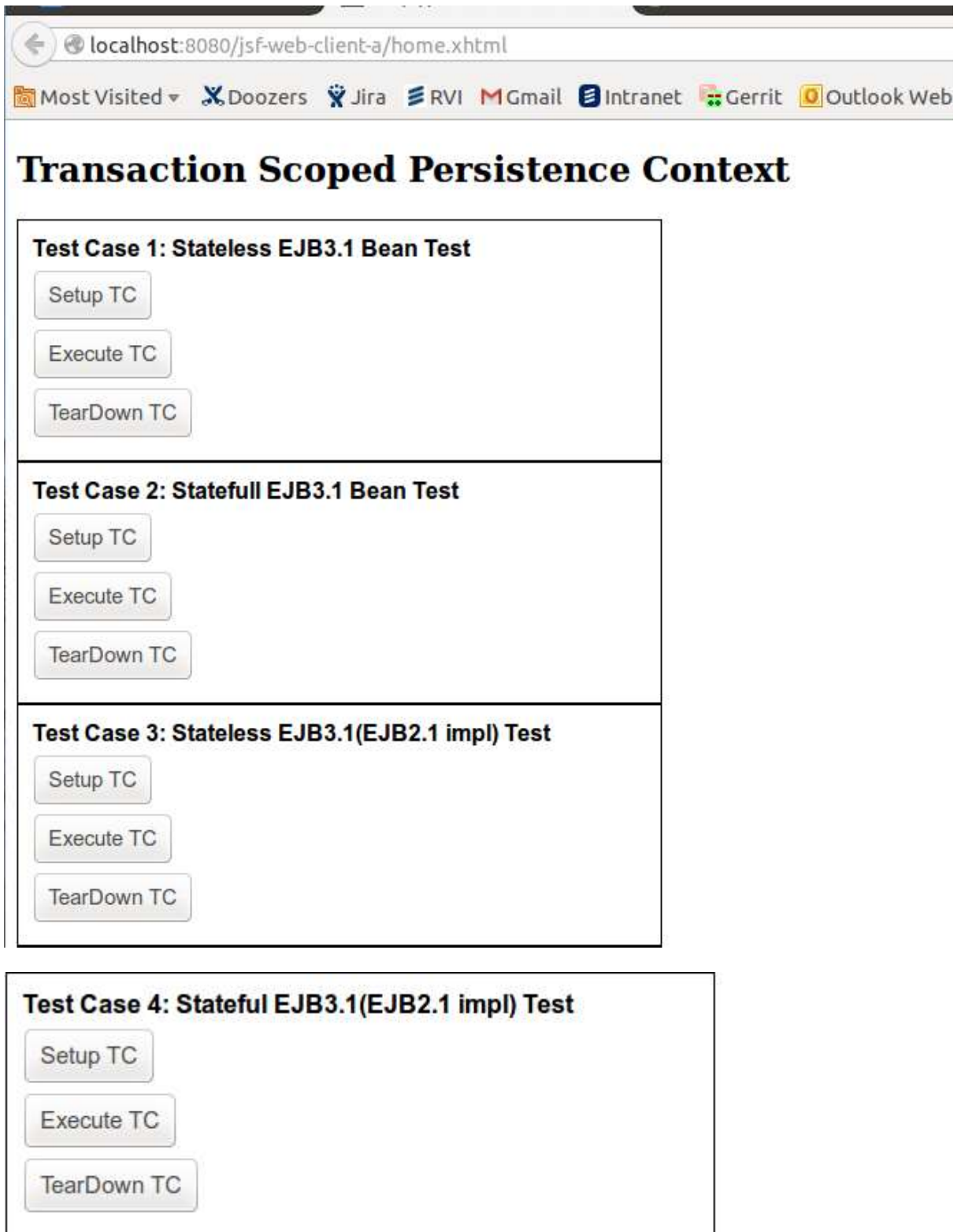
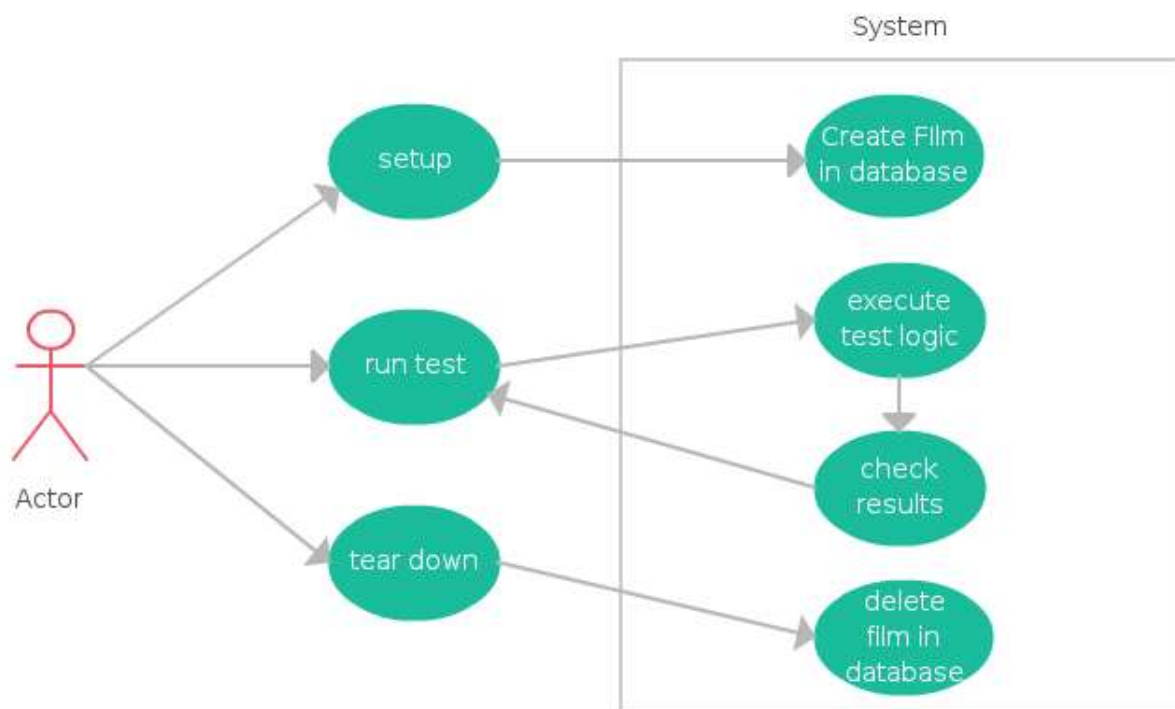


Fig. 19 Screenshot of the JSF web client

### 3.3 High Level Design

The use cases in this system follow the same pattern and so are repeated for different test cases using different jboss server configuration settings and different types of enterprise java beans.

#### 3.2.1 Use Case Descriptions



**Fig. 20** The Use Case Descriptons

<b>Use Case 1</b>	Stateless EJB3.1in server A and B with JTA only, no JTS and Jacorb TXs off
<b>Goal in Context</b>	Ejb3x_StatelessA in server A creates new TX and modifies attribute in Film, invokes remotely Ejb3x_StatelessB in server B which invokes remotely back to Ejb3x_StatelessA in server A to create new Cast

	object and read the modified attribute from the same TX persistence context as used by Ejb3x_StatelessA in server A	
<b>Scope and level</b>	Single remote method invocation from server A to server B, two remote invocations from server B to server A. JTA only	
<b>Preconditions</b>	Both application servers are started. ear-module-a-1.0-SNAPSHOT.ear is deployed in server A and ear-module-b-1.0-SNAPSHOT.ear in server B. Both server A and B are not configured to use JTS. Web browser has loaded the page "http://http://localhost:8080/jsf-web-client-a/home.xhtml"	
<b>Success post condition</b>	Ejb3x_StatelessA in server A successfully retrieves the modified attribute via Ejb3x_StatelessB in server B and the Cast object is successfully committed to the database and the result "passed" is displayed in the web browser	
<b>Failed post condition</b>	Ejb3x_StatelessA in server A does not successfully retrieve the modified attribute via Ejb3x_StatelessB in server B and the Cast object is not successfully committed to the database and the TX is rolled back	
<b>Primary - Secondary actors</b>	User (primary) System (secondary)	
<b>Trigger</b>	The user clicks the Execute TC button in the web browser	
<b>Description</b>	<b>Step</b>	<b>Action</b>
	1	User clicks the "Setup TC" button to create a Film object in the database
	2	User clicks the "Execute TC" to run the test
	3	The system starts a new TX if one does not exist already and invokes stateless Ejb3x_StatelessA runTest method

	4	The system retrieves the <i>Film</i> object from the database
	5	The system modifies the <i>CountryOfOrigin</i> attribute value to <i>IRELAND</i>
	6	The system invokes Ejb3x_StatelessB in server B within the same TX
	7	The system invokes Ejb3x_StatelessA twice in server A to get the <i>CountryOfOrigin</i> attribute and create a new <i>Cast</i> object within the <i>Film</i> object
	8	The system returns the <i>CountryOfOrigin</i> attribute from Ejb3x_StatelessB in server B to Ejb3x_StatelessA in server A
	9	The system verifies that the <i>CountryOfOrigin</i> returned has the value <i>IRELAND</i> and that there exists a new <i>Cast</i> object within the <i>Film</i> object
	10	The system commits the Ejb3x_StatelessA TX in server A
<b>Extensions</b>		<b>Branching Actions</b>
	1a	The user fails to create a <i>Film</i> object in the database
	2a	The user fails to execute the test
	4a	The system fails to retrieve the <i>Film</i> object from the database
	6a	The system fails to remotely invoke Ejb3x_StatelessB in server B
	7a	The system fails to remotely invoke Ejb3x_StatelessA in server A
	9a	The system fails to verify the expected results

<b>Use Case 2</b>	Stateless EJB3.1 in server A and B with JTA, JTS on and Jacorb TXs on	
<b>Goal in Context</b>	Ejb3x_StatelessA in server A creates new TX and modifies attribute in Film, invokes remotely Ejb3x_StatelessB in server B which invokes remotely back to same Ejb3x_StatelessA in server A to create new Cast object and read the modified attribute from the same TX persistence context as used Ejb3x_StatelessA in server A	
<b>Scope and level</b>	Single remote method invocation from server A to server B, two remote invocations from server B to server A with JTA, JTS and Jacorb TXs on	
<b>Preconditions</b>	Both application servers are started. ear-module-a-1.0-SNAPSHOT.ear is deployed in server A and ear-module-b-1.0-SNAPSHOT.ear in server B. Both server A and B are configured to use JTS and Jacorb TXs. Web browser has loaded the page "http://localhost:8080/jsf-web-client-a/home.xhtml"	
<b>Success post condition</b>	Ejb3x_StatelessA in server A successfully retrieves the modified attribute via Ejb3x_StatelessB in server B and the Cast object is successfully committed to the database and the result "passed" is displayed in the web browser	
<b>Failed post condition</b>	Ejb3x_StatelessA in server A does not successfully retrieve the modified attribute via Ejb3x_StatelessB in server B and the Cast object is not successfully committed to the database and the TX is rolled back	
<b>Primary - Secondary actors</b>	User (primary) System (secondary)	
<b>Trigger</b>	The user clicks the Execute TC button in the web browser	
<b>Description</b>	<b>Ste</b>	<b>Action</b>

	<b>p</b>	
	1	User clicks the "Setup TC" button to create a Film object in the database
	2	User clicks the "Execute TC" to run the test
	3	The system starts a new TX if one does not exist already and invokes stateless Ejb3x_StatelessA runTest method
	4	The system retrieves the <i>Film</i> object from the database
	5	The system modifies the <i>CountryOfOrigin</i> attribute value to <i>IRELAND</i>
	6	The system invokes Ejb3x_StatelessB in server B within the same TX
	7	The system invokes Ejb3x_StatelessA twice in server A to get the <i>CountryOfOrigin</i> attribute and create a new Cast object within the <i>Film</i> object
	8	The system returns the <i>CountryOfOrigin</i> attribute from Ejb3x_StatelessB in server B to Ejb3x_StatelessA in server A
	9	The system verifies that the <i>CountryOfOrigin</i> returned has the value <i>IRELAND</i> and that there exists a new <i>Cast</i> object within the <i>Film</i> object
	10	The system commits the Ejb3x_StatelessA TX in server A
<b>Extensions</b>		<b>Branching Actions</b>
	1a	The user fails to create a <i>Film</i> object in the database
	2a	The user fails to execute the test
	4a	The system fails to retrieve the <i>Film</i> object from the database



	6a	The system fails to remotely invoke Ejb3x_StatelessB in server B
	7a	The system fails to remotely invoke Ejb3x_StatelessA in server A
	9a	The system fails to verify the expected results

<b>Use Case 3</b>	Statefull EJB3.1in server A and B with JTA only, no JTS and Jacorb TXs off
<b>Goal in Context</b>	Ejb3x_StatefulA in server A creates new TX and modifies attribute in Film, invokes remotely Ejb3x_StatefulB in server B which invokes remotely back to same Ejb3x_StatefulA in server A to create new Cast object and read the modified attribute from the same TX persistence context as used Ejb3x_StatefulA in server A
<b>Scope and level</b>	Single remote method invocation from server A to server B, two remote invocations from server B to server A with JTA, JTS and Jacorb TXs on
<b>Preconditions</b>	Both application servers are started. ear-module-a-1.0-SNAPSHOT.ear is deployed in server A and ear-module-b-1.0-SNAPSHOT.ear in server B. Both server A and B are configured to use JTS and Jacorb TXs. Web browser has loaded the page “http://localhost:8080/jsf-web-client-a/home.xhtml”
<b>Success post condition</b>	Ejb3x_StatefulA in server A successfully retrieves the modified attribute via Ejb3x_StatefulB in server B and the Cast object is successfully committed to the database and the result “passed” is

	displayed in the web browser	
<b>Failed post condition</b>	Ejb3x_StatefulA in server A does not successfully retrieve the modified attribute via Ejb3x_StatefulB in server B and the Cast object is not successfully committed to the database and the TX is rolled back	
<b>Primary - Secondary actors</b>	User (primary) System (secondary)	
<b>Trigger</b>	The user clicks the Execute TC button in the web browser	
<b>Description</b>	<b>Step</b>	<b>Action</b>
	1	User clicks the "Setup TC" button to create a Film object in the database
	2	User clicks the "Execute TC" to run the test
	3	The system starts a new TX if one does not exist already and invokes stateless Ejb3x_StatefulA runTest method
	4	The system retrieves the <i>Film</i> object from the database
	5	The system modifies the <i>CountryOfOrigin</i> attribute value to <i>IRELAND</i>
	6	The system invokes Ejb3x_StatefulB in server B within the same TX
	7	The system invokes Ejb3x_StatefulA twice in server A to get the <i>CountryOfOrigin</i> attribute and create a new Cast object within the <i>Film</i> object
	8	The system returns the <i>CountryOfOrigin</i> attribute from Ejb3x_StatefulB in server B to Ejb3x_StatefulA in server A

	9	The system verifies that the <i>CountryOfOrigin</i> returned has the value <i>IRELAND</i> and that there exists a new <i>Cast</i> object within the <i>Film</i> object
	10	The system commits the Ejb3x_StatefulA TX in server A
<b>Extensions</b>		<b>Branching Actions</b>
	1a	The user fails to create a <i>Film</i> object in the database
	2a	The user fails to execute the test
	4a	The system fails to retrieve the <i>Film</i> object from the database
	6a	The system fails to remotely invoke Ejb3x_StatefulB in server B
	7a	The system fails to remotely invoke Ejb3x_StatefulA in server A
	9a	The system fails to verify the expected results

<b>Use Case 4</b>	Stateful EJB3.1 in server A and B with JTA, JTS on and Jacorb TXs on
<b>Goal in Context</b>	Ejb3x_StatefulA in server A creates new TX and modifies attribute in Film, invokes remotely Ejb3x_StatefulB in server B which invokes remotely back to same Ejb3x_StatefulA in server A to create new Cast object and read the modified attribute from the same TX persistence context as used Ejb3x_StatefulA in server A
<b>Scope and level</b>	Single remote method invocation from server A to server B, two remote invocations from server B to server A with JTA, JTS and

	Jacorb TXs on	
<b>Preconditions</b>	<p>Both application servers are started.</p> <p>ear-module-a-1.0-SNAPSHOT.ear is deployed in server A and ear-module-b-1.0-SNAPSHOT.ear in server B.</p> <p>Both server A and B are configured to use JTS and Jacorb TXs.</p> <p>Web browser has loaded the page “http://localhost:8080/jsf-web-client-a/home.xhtml”</p>	
<b>Success post condition</b>	Ejb3x_StatefulA in server A successfully retrieves the modified attribute via Ejb3x_StatefulB in server B and the Cast object is successfully committed to the database and the result “passed” is displayed in the web browser	
<b>Failed post condition</b>	Ejb3x_StatefulA in server A does not successfully retrieve the modified attribute via Ejb3x_StatefulB in server B and the Cast object is not successfully committed to the database and the TX is rolled back	
<b>Primary - Secondary actors</b>	<p>User (primary)</p> <p>System (secondary)</p>	
<b>Trigger</b>	The user clicks the Execute TC button in the web browser	
<b>Description</b>	<b>Step</b>	<b>Action</b>
	1	User clicks the “Setup TC” button to create a Film object in the database
	2	User clicks the “Execute TC” to run the test
	3	The system starts a new TX if one does not exist already and invokes stateless Ejb3x_StatefulA runTest method
	4	The system retrieves the <i>Film</i> object from the database

	5	The system modifies the <i>CountryOfOrigin</i> attribute value to <i>IRELAND</i>
	6	The system invokes Ejb3x_StatefulB in server B within the same TX
	7	The system invokes Ejb3x_StatefulA twice in server A to get the <i>CountryOfOrigin</i> attribute and create a new <i>Cast</i> object within the <i>Film</i> object
	8	The system returns the <i>CountryOfOrigin</i> attribute from Ejb3x_StatefulB in server B to Ejb3x_StatefulA in server A
	9	The system verifies that the <i>CountryOfOrigin</i> returned has the value <i>IRELAND</i> and that there exists a new <i>Cast</i> object within the <i>Film</i> object
	10	The system commits the Ejb3x_StatefulA TX in server A
<b>Extensions</b>		<b>Branching Actions</b>
	1a	The user fails to create a <i>Film</i> object in the database
	2a	The user fails to execute the test
	4a	The system fails to retrieve the <i>Film</i> object from the database
	6a	The system fails to remotely invoke Ejb3x_StatefulB in server B
	7a	The system fails to remotely invoke Ejb3x_StatefulA in server A
	9a	The system fails to verify the expected results

<b>Use Case 5</b>	Stateless EJB2.1 in server A and B with JTA only, no JTS and Jacorb TXs off
<b>Goal in Context</b>	Ejb2x_StatelessA in server A creates new TX and modifies attribute in Film, invokes remotely Ejb2x_StatelessB in server B which invokes remotely back to same Ejb2x_StatelessA in server A to create new Cast object and read the modified attribute from the same TX persistence context as used Ejb2x_StatelessA in server A
<b>Scope and level</b>	Single remote method invocation from server A to server B, two remote invocations from server B to server A. JTA only
<b>Preconditions</b>	Both application servers are started. ear-module-a-1.0-SNAPSHOT.ear is deployed in server A and ear-module-b-1.0-SNAPSHOT.ear in server B. Both server A and B are not configured to use JTS. Web browser has loaded the page "http://localhost:8080/jsf-web-client-a/home.xhtml"
<b>Success post condition</b>	Ejb2x_StatelessA in server A successfully retrieves the modified attribute via Ejb2x_StatelessB in server B and the Cast object is successfully committed to the database and the result "passed" is displayed in the web browser
<b>Failed post condition</b>	Ejb2x_StatelessA in server A does not successfully retrieve the modified attribute via Ejb2x_StatelessB in server B and the Cast object is not successfully committed to the database and the TX is rolled back
<b>Primary - Secondary actors</b>	User (primary) System (secondary)
<b>Trigger</b>	The user clicks the Execute TC button in the web browser

Description	Step	Action
	1	User clicks the "Setup TC" button to create a Film object in the database
	2	User clicks the "Execute TC" to run the test
	3	The system starts a new TX if one does not exist already and invokes stateless Ejb2x_StatelessA runTest method
	4	The system retrieves the <i>Film</i> object from the database
	5	The system modifies the <i>CountryOfOrigin</i> attribute value to <i>IRELAND</i>
	6	The system invokes Ejb2x_StatelessB in server B within the same TX
	7	The system invokes Ejb2x_StatelessA twice in server A to get the <i>CountryOfOrigin</i> attribute and create a new Cast object within the <i>Film</i> object
	8	The system returns the <i>CountryOfOrigin</i> attribute from Ejb2x_StatelessB in server B to Ejb2x_StatelessA in server A
	9	The system verifies that the <i>CountryOfOrigin</i> returned has the value <i>IRELAND</i> and that there exists a new <i>Cast</i> object within the <i>Film</i> object
	10	The system commits the Ejb2x_StatelessA TX in server A
<b>Extensions</b>		<b>Branching Actions</b>
	1a	The user fails to create a <i>Film</i> object in the database
	2a	The user fails to execute the test

	4a	The system fails to retrieve the <i>Film</i> object from the database
	6a	The system fails to remotely invoke Ejb2x_StatelessB in server B
	7a	The system fails to remotely invoke Ejb2x_StatelessA in server A
	9a	The system fails to verify the expected results

<b>Use Case 6</b>	Stateless EJB2.1 in server A and B with JTA, JTS on and Jacorb TXs on
<b>Goal in Context</b>	Ejb2x_StatelessA in server A creates new TX and modifies attribute in Film, invokes remotely Ejb2x_StatelessB in server B which invokes remotely back to same Ejb2x_StatelessA in server A to create new Cast object and read the modified attribute from the same TX persistence context as used Ejb2x_StatelessA in server A
<b>Scope and level</b>	Single remote method invocation from server A to server B, two remote invocations from server B to server A with JTA, JTS and Jacorb TXs on
<b>Preconditions</b>	Both application servers are started. ear-module-a-1.0-SNAPSHOT.ear is deployed in server A and ear-module-b-1.0-SNAPSHOT.ear in server B. Both server A and B are configured to use JTS and Jacorb TXs. Web browser has loaded the page "http://localhost:8080/jsf-web-client-a/home.xhtml"
<b>Success post condition</b>	Ejb2x_StatelessA in server A successfully retrieves the modified attribute via Ejb2x_StatelessB in server B and the Cast object is successfully committed to the database and the result "passed" is



	displayed in the web browser	
<b>Failed post condition</b>	Ejb2x_StatelessA in server A does not successfully retrieve the modified attribute via Ejb2x_StatelessB in server B and the Cast object is not successfully committed to the database and the TX is rolled back	
<b>Primary - Secondary actors</b>	User (primary) System (secondary)	
<b>Trigger</b>	The user clicks the Execute TC button in the web browser	
<b>Description</b>	<b>Step</b>	<b>Action</b>
	1	User clicks the "Setup TC" button to create a Film object in the database
	2	User clicks the "Execute TC" to run the test
	3	The system starts a new TX if one does not exist already and invokes stateless Ejb2x_StatelessA runTest method
	4	The system retrieves the <i>Film</i> object from the database
	5	The system modifies the <i>CountryOfOrigin</i> attribute value to <i>IRELAND</i>
	6	The system invokes Ejb2x_StatelessB in server B within the same TX
	7	The system invokes Ejb2x_StatelessA twice in server A to get the <i>CountryOfOrigin</i> attribute and create a new Cast object within the <i>Film</i> object
	8	The system returns the <i>CountryOfOrigin</i> attribute from Ejb2x_StatelessB in server B to Ejb2x_StatelessA in server A

	9	The system verifies that the <i>CountryOfOrigin</i> returned has the value <i>IRELAND</i> and that there exists a new <i>Cast</i> object within the <i>Film</i> object
	10	The system commits the Ejb2x_StatelessA TX in server A
<b>Extensions</b>		<b>Branching Actions</b>
	1a	The user fails to create a <i>Film</i> object in the database
	2a	The user fails to execute the test
	4a	The system fails to retrieve the <i>Film</i> object from the database
	6a	The system fails to remotely invoke Ejb2x_StatelessB in server B
	7a	The system fails to remotely invoke Ejb2x_StatelessA in server A
	9a	The system fails to verify the expected results

<b>Use Case 7</b>	Stateless EJB2.1 in server A and B with JTA only, no JTS and Jacorb TXs off
<b>Goal in Context</b>	Ejb2x_StatefulA in server A creates new TX and modifies attribute in Film, invokes remotely Ejb2x_StatefulB in server B which invokes remotely back to same Ejb2x_StatefulA in server A to create new Cast object and read the modified attribute from the same TX persistence context as used Ejb2x_StatefulA in server A
<b>Scope and level</b>	Single remote method invocation from server A to server B, two

	remote invocations from server B to server A. JTA only	
<b>Preconditions</b>	<p>Both application servers are started.</p> <p>ear-module-a-1.0-SNAPSHOT.ear is deployed in server A and ear-module-b-1.0-SNAPSHOT.ear in server B.</p> <p>Both server A and B are not configured to use JTS.</p> <p>Web browser has loaded the page “http://localhost:8080/jsf-web-client-a/home.xhtml”</p>	
<b>Success post condition</b>	Ejb2x_StatefulA in server A successfully retrieves the modified attribute via Ejb2x_StatefulB in server B and the Cast object is successfully committed to the database and the result “passed” is displayed in the web browser	
<b>Failed post condition</b>	Ejb2x_StatefulA in server A does not successfully retrieve the modified attribute via Ejb2x_StatefulB in server B and the Cast object is not successfully committed to the database and the TX is rolled back	
<b>Primary - Secondary actors</b>	<p>User (primary)</p> <p>System (secondary)</p>	
<b>Trigger</b>	The user clicks the Execute TC button in the web browser	
<b>Description</b>	<b>Step</b>	<b>Action</b>
	1	User clicks the “Setup TC” button to create a Film object in the database

	2	User clicks the "Execute TC" to run the test
	3	The system starts a new TX if one does not exist already and invokes stateless Ejb2x_StatefulA runTest method
	4	The system retrieves the <i>Film</i> object from the database
	5	The system modifies the <i>CountryOfOrigin</i> attribute value to <i>IRELAND</i>
	6	The system invokes Ejb2x_StatefulB in server B within the same TX
	7	The system invokes Ejb2x_StatefulA twice in server A to get the <i>CountryOfOrigin</i> attribute and create a new Cast object within the <i>Film</i> object
	8	The system returns the <i>CountryOfOrigin</i> attribute from Ejb2x_StatefulB in server B to Ejb2x_StatefulA in server A
	9	The system verifies that the <i>CountryOfOrigin</i> returned has the value <i>IRELAND</i> and that there exists a new <i>Cast</i> object within the <i>Film</i> object
	10	The system commits the Ejb2x_StatefulA TX in server A
<b>Extensions</b>		<b>Branching Actions</b>
	1a	The user fails to create a <i>Film</i> object in the database
	2a	The user fails to execute the test

	4a	The system fails to retrieve the <i>Film</i> object from the database
	6a	The system fails to remotely invoke Ejb2x_StatefulB in server B
	7a	The system fails to remotely invoke Ejb2x_StatefulA in server A
	9a	The system fails to verify the expected results

<b>Use Case 8</b>	Stateless EJB2.1 in server A and B with JTA, JTS on and Jacorb TXs on
<b>Goal in Context</b>	Ejb2x_StatefulA in server A creates new TX and modifies attribute in Film, invokes remotely Ejb2x_StatefulB in server B which invokes remotely back to same Ejb2x_StatefulA in server A to create new Cast object and read the modified attribute from the same TX persistence context as used Ejb2x_StatefulA in server A
<b>Scope and level</b>	Single remote method invocation from server A to server B, two remote invocations from server B to server A with JTA, JTS and Jacorb TXs on
<b>Preconditions</b>	Both application servers are started.  ear-module-a-1.0-SNAPSHOT.ear is deployed in server A and ear-module-b-1.0-SNAPSHOT.ear in server B.  Both server A and B are configured to use JTS and Jacorb TXs.  Web browser has loaded the page "http://localhost:8080/jsf-web-

	client-a/home.xhtml”	
<b>Success post condition</b>	Ejb2x_StatefulA in server A successfully retrieves the modified attribute via Ejb2x_StatefulB in server B and the Cast object is successfully committed to the database and the result “passed” is displayed in the web browser	
<b>Failed post condition</b>	Ejb2x_StatefulA in server A does not successfully retrieve the modified attribute via Ejb2x_StatefulB in server B and the Cast object is not successfully committed to the database and the TX is rolled back	
<b>Primary - Secondary actors</b>	User (primary) System (secondary)	
<b>Trigger</b>	The user clicks the Execute TC button in the web browser	
<b>Description</b>	<b>Step</b>	<b>Action</b>
	1	User clicks the “Setup TC” button to create a Film object in the database
	2	User clicks the “Execute TC” to run the test
	3	The system starts a new TX if one does not exist already and invokes stateless Ejb2x_StatefulA runTest method
	4	The system retrieves the <i>Film</i> object from the database

	5	The system modifies the <i>CountryOfOrigin</i> attribute value to <i>IRELAND</i>
	6	The system invokes Ejb2x_StatefulB in server B within the same TX
	7	The system invokes Ejb2x_StatefulA twice in server A to get the <i>CountryOfOrigin</i> attribute and create a new Cast object within the <i>Film</i> object
	8	The system returns the <i>CountryOfOrigin</i> attribute from Ejb2x_StatefulB in server B to Ejb2x_StatefulA in server A
	9	The system verifies that the <i>CountryOfOrigin</i> returned has the value <i>IRELAND</i> and that there exists a new <i>Cast</i> object within the <i>Film</i> object
	10	The system commits the Ejb2x_StatefulA TX in server A
<b>Extensions</b>		<b>Branching Actions</b>
	1a	The user fails to create a <i>Film</i> object in the database
	2a	The user fails to execute the test
	4a	The system fails to retrieve the <i>Film</i> object from the database
	6a	The system fails to remotely invoke Ejb2x_StatefulB in server B
	7a	The system fails to remotely invoke Ejb2x_StatefulA in server

		A
	9a	The system fails to verify the expected results

### 3.4 Low Level Design

The source code of the complete system is spread over 5 different modules. As it is not feasible to show all the classes and their relationships with each other in a single diagram instead a break down of the class diagrams per module is shown. The next section will show each of these class diagrams in each module and a description of each module. The source code modules of the system are:

- **remote-api** - library module that contains the remote interfaces used by the system
- **ejb-module-a** - an EJB module for server A
- **ejb-module-b** - an EJB module for server B
- **jsf-web-client-a** - WEB module that contains the test client code
- **test-api** - library module that contains the test case interfaces used by the ejb module A and also contains an interface that is used for testing purposes only

#### 3.4.1 Class Diagram

##### 3.4.1.1 remote-api module

Fig 21. and Fig 22. shows the classes of the *remote-api* library module that contains common code used by both the *ejb-module-a* and *ejb-module-b*. Most importantly it contains the different remote interfaces used by the system and it also contains some helper classes and constants that are used by both ejb modules to locate different EJB bean versions. The *remote-api.jar* is contained in the */lib* directory of each EAR file.



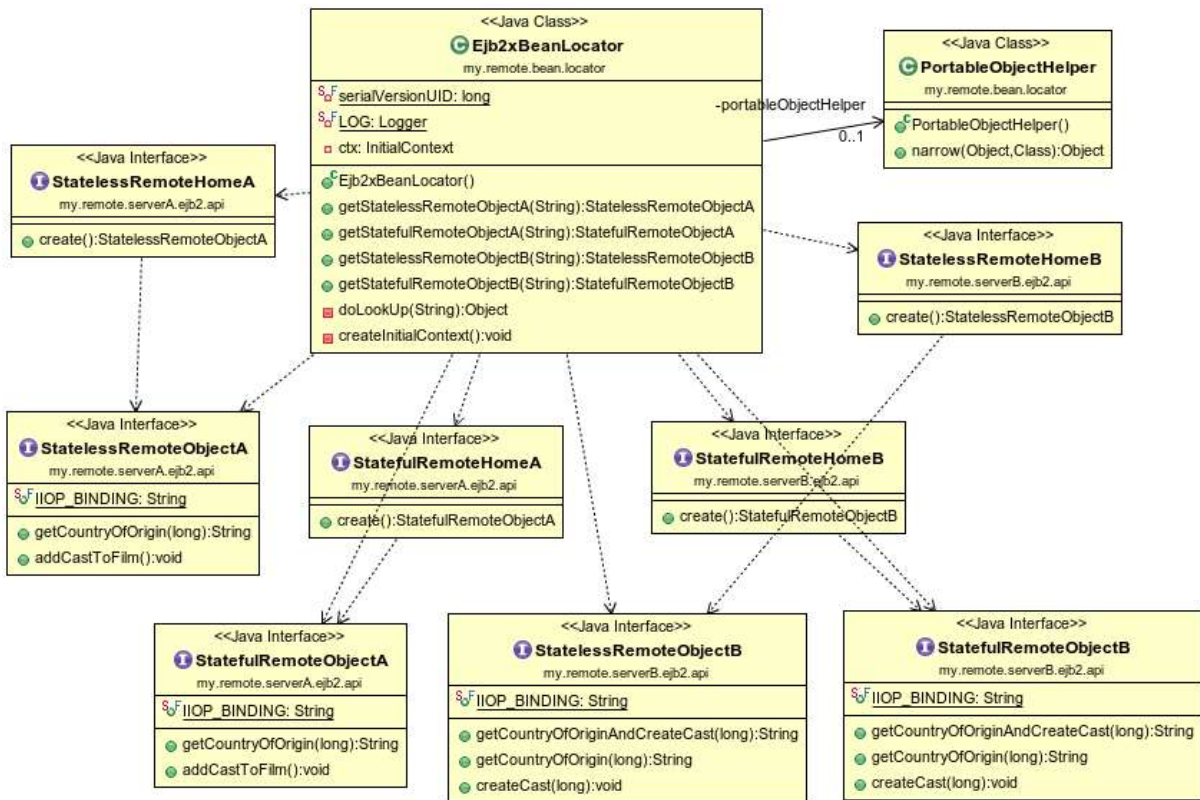


Fig. 21 remote-api class diagram

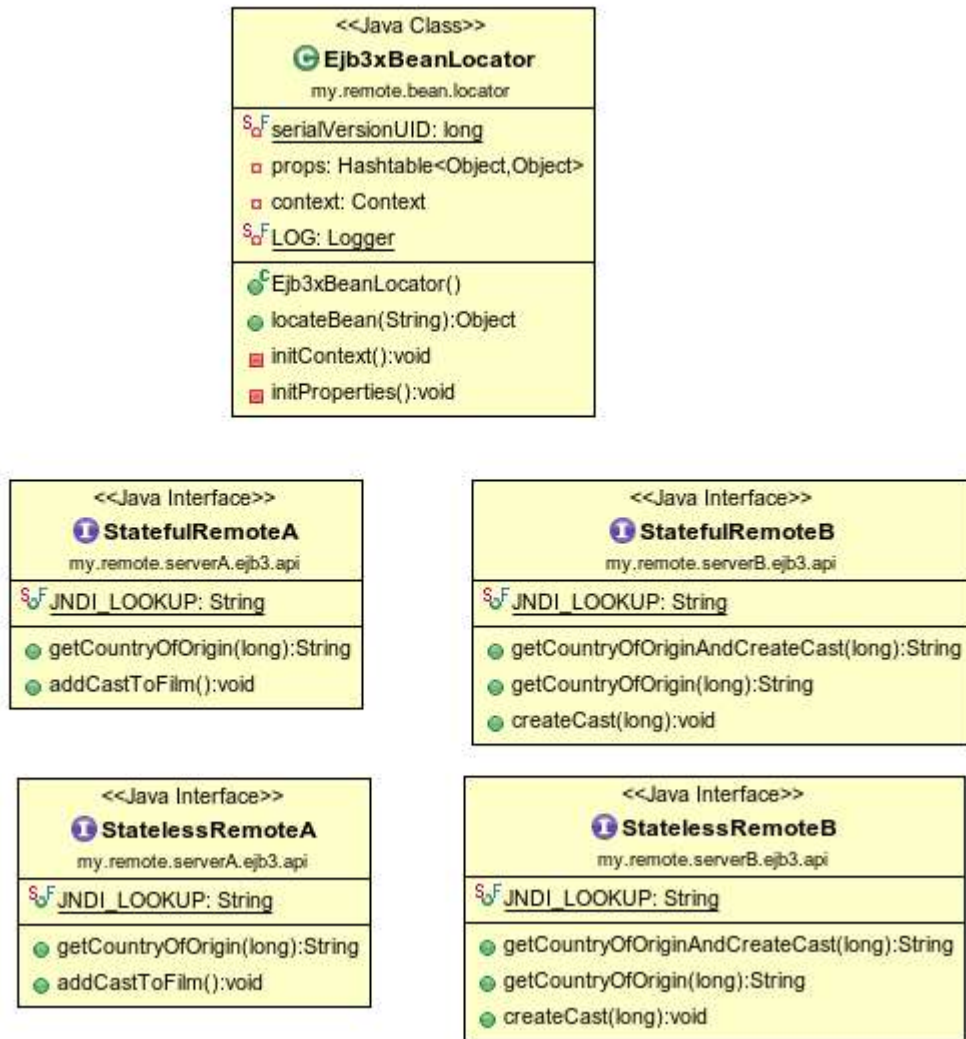


Fig. 22 remote-api class diagram

- StatefulRemoteHomeA - the EJB2.x stateful remote home interface which must extend the EJBHOME interface used by server A to create StatefulRemoteObjectA objects
- StatefulRemoteObjectA - the EJB2.x stateful EJB remote object interface which provides the remote client business view
- StatelessRemoteHomeA - the EJB2.x stateful EJB remote home interface which must extend the EJBHOME interface
- StatelessRemoteObjectA - the EJB2.x stateless EJB remote object interface which provides the remote client business view
- StatefulRemoteA - the EJB3.x stateful remote business interface
- StatelessRemoteA - the EJB3.x stateless remote business interface

- StatefulRemoteHomeB - the EJB2.x stateful remote home interface which must extend the EJBHOME interface used by server B to create StatefulRemoteObjectB objects
- StatefulRemoteObjectB - the EJB2.x stateful EJB remote object interface which provides the remote client business view
- StatelessRemoteHomeB - the EJB2.x stateful EJB remote home interface which must extend the EJBHOME interface
- StatelessRemoteObjectB - the EJB2.x stateless EJB remote object interface which provides the remote client business view
- StatefulRemoteB - the EJB3.x stateful remote business interface
- StatelessRemoteB - the EJB3.x stateless remote business interface
- Ejb2xBeanLocator - the helper bean for locating EJB2.x beans
- PortableObjectHelper - used by Ejb2xBeanLocator to narrow objects
- Ejb3xBeanLocator - the helper bean for locating EJB3.x beans
- Constants - contains common constants used through the system

### 3.4.1.2 *ejb-module-a*

Fig 23. shows the different combinations of EJB implementation classes that are deployed as part of *ear-module-a-1.0-SNAPSHOT.ear* on server A that implement the test case interface. It is through this *TestCase* interface that the test setup, execution and teardown is done similar to the junit philosophy. It is also these EJB classes that interact with the WEB module that contains the *TestManager* test client which injects these EJBs into its managed bean that is controlled by a Java Server Faces (JSF) client. This client can be viewed in a web browser and it allows the user to click buttons to invoke the varioud methods on the *TestCase* interface. The EJB3x and EJB2x bean locator helper classes from the remote api library ase used by the different test EJBs to locate the respective EJBs implementation versions in server B. The persistent entity classes *Film* and *Cast* are also shown.

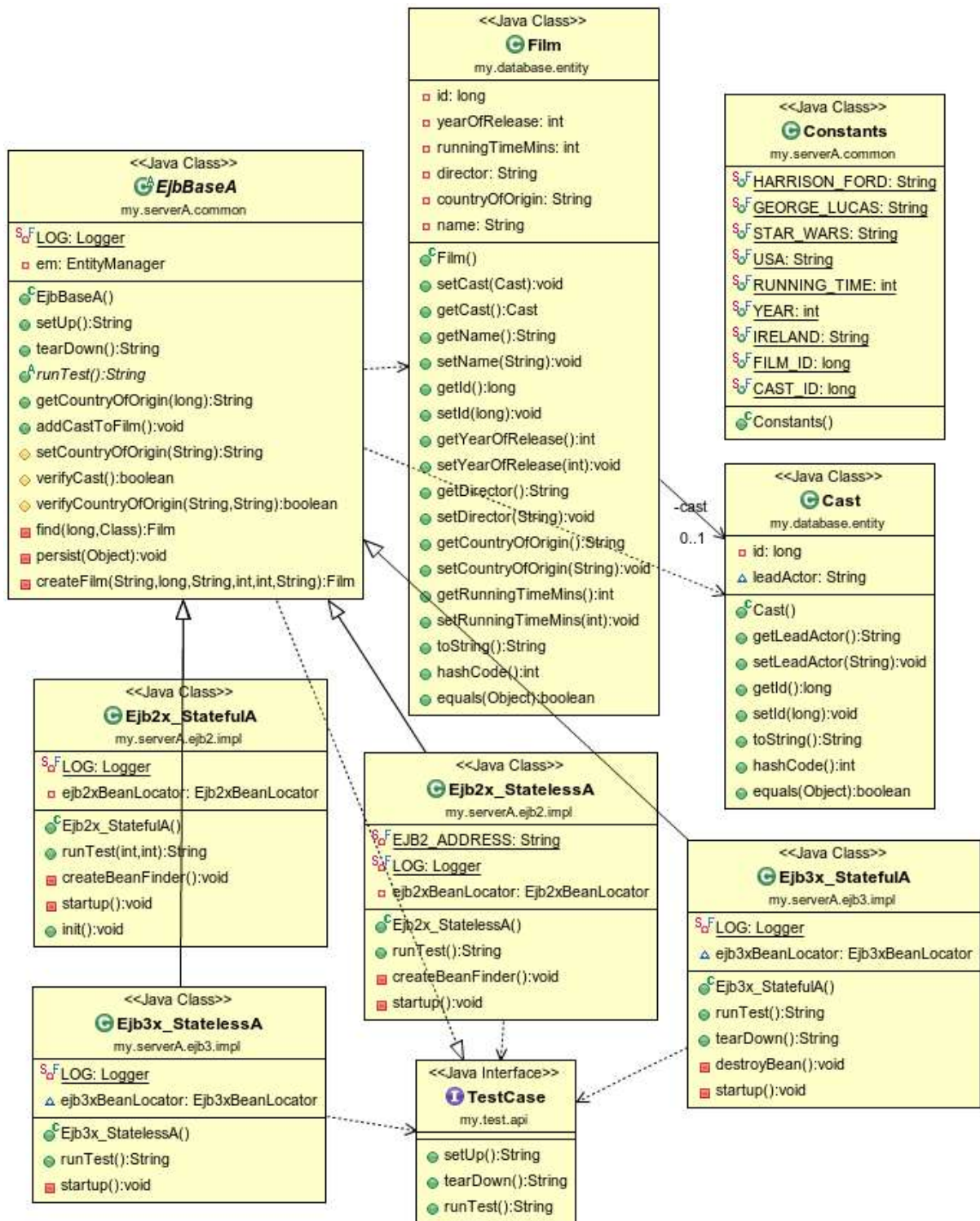


Fig. 23 ejb-module-a class diagram

- EjbBaseA - this is an abstract class that implements the *TestCase* local interface. It contains common method implementations that are used by all the subclasses, for

example it contains the *getCountryOfOrigin* method that is used by the EJBs in server B to try and retrieve the modified entity attribute.

- Ejb2x\_StatelessA - the EJB2.x stateless EJB implementation that extends EjbBaseA and provides a specific implementation of the *runTest* method. This bean designates the *RemoteHome* interface to be adapted to be *StatelessRemoteHomeA* and implements the *StatelessRemoteObjectA* interface business method.
- Ejb2x\_StatefulA - the EJB2.x stateful EJB implementation that extends EjbBaseA and provides a specific implementation of the *runTest* method. This bean designates the *RemoteHome* interface to be adapted to be *StatefulRemoteHomeA* and implements the *StatefulRemoteObjectA* interface business method.
- Ejb3x\_StatefulA - the EJB3.x stateful EJB implementation that extends EjbBaseA and provides a specific implementation of the *runTest* method. This bean implements the *StatefulRemoteA* remote interface.
- Ejb3x\_StatelessA - the EJB3.x stateless EJB implementation that extends EjbBaseA and provides a specific implementation of the *runTest* method. This bean implements the *StatelessRemoteA* remote interface.
- Film - the entity class that has some attributes related to a film. It also contains a *Cast* field object.
- Cast - the entity class that has some attributes related to a cast.
- Constants - a class that stores some constant values related to the Film and Cast entities.

### 3.4.1.3 ejb-module-b

Fig 24. shows the different combinations of EJB implementation classes that are deployed as part of *ear-module-b-1.0-SNAPSHOT.ear* on server B. These EJBs are invoked from server A and all of them participate (join) in the transaction that is started by the EJB in container A. These EJBs then invoke back to server A to try and read the modified *CountryOfOrigin* attribute and to also create a new *Cast* object before returning from their method call back to server A.





Fig. 24 ejb-module-b class diagram

- Ejb3x\_StatefulB - the EJB3.x stateful EJB implementation bean that implements the *StatefulRemoteB* remote interface. This bean contains methods to lookup the StatefulRemoteA EJB to read the *CountryOfOriginAttribute* and create a *Cast* object. The business methods of this bean join the transaction context of the calling EJB.
- Ejb3x\_StatelessB - the EJB3.x stateful EJB implementation bean that implements the *StatelessRemoteB* remote interface. This bean contains methods to lookup the StatelessRemoteA EJB to read the *CountryOfOriginAttribute* and create a *Cast* object. The business methods of this bean join the transaction context of the calling EJB.
- Ejb2x\_StatelessB - the EJB2.x stateless EJB implementation that implements the *RemoteHome* interface to be adapted to be *StatelessRemoteHomeB* and implements the *StatelessRemoteObjectB* interface business method. This bean contains methods to lookup the StatelessRemoteObjectA EJB to read the *CountryOfOriginAttribute* and create

a *Cast* object. The business methods of this bean join the transaction context of the calling EJB.

- *Ejb2x\_StatefulB* - the EJB2.x stateful EJB implementation that implements the *RemoteHome* interface to be adapted to be *StatefulRemoteHomeB* and implements the *StatefulRemoteObjectB* interface business method. This bean contains methods to lookup the *StatefulRemoteObjectA* EJB to read the *CountryOfOriginAttribute* and create a *Cast* object. The business methods of this bean join the transaction context of the calling EJB.

#### 3.4.1.4 jsf-web-client-a module

Fig 25. shows the class belonging to the web application that is deployed as *jsf-web-client-a-1.0-SNAPSHOT.war* on server A. This is a Java Server Faces client that requires the *primefaces-3.5.jar* runtime library on its classpath. The client is implemented by the *TestCaseManager* class. The structure, layout and presentation of the web client in the browser is controlled by the web content file *home.xhtml*.

- *TestCaseManager* - this is the main managed bean client class. Each different EJB implementation is injected into the managed bean by using its default class name as a reference. Each test case has a similar set of variables for storing the results of each operation of the *TestCase* interface (which each EJB implements) for displaying on the web browser to give feedback to the user of the results of each operation (button press).

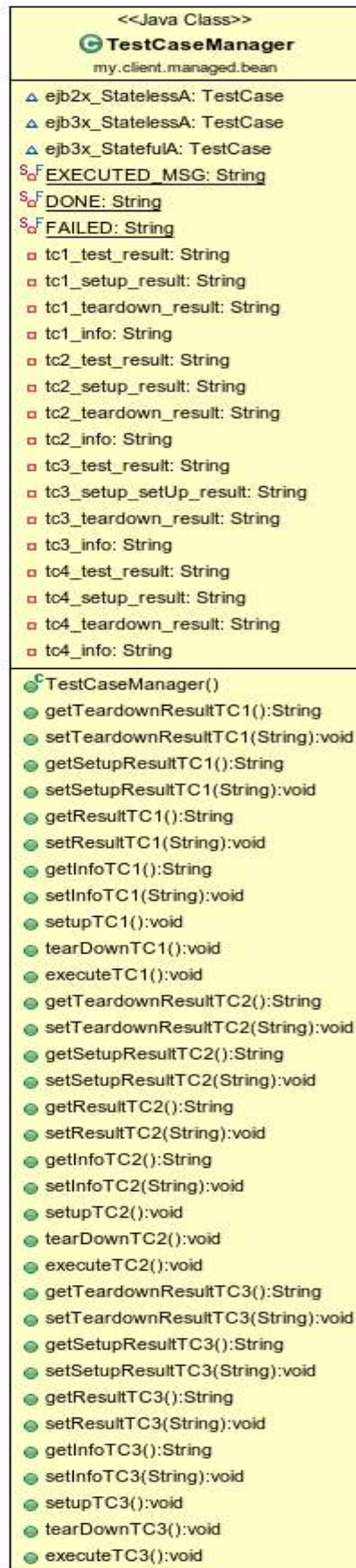


Fig. 25 jsf-web-client-a class diagram



### 3.4.1.5 Test Client API module

Fig 26. shows the test client API interface that is implemented by the different EJB implementations that are deployed on server A. This API module is the link between the web client and the different EJB implementations deployed on server A and enables the user to interact with the system.



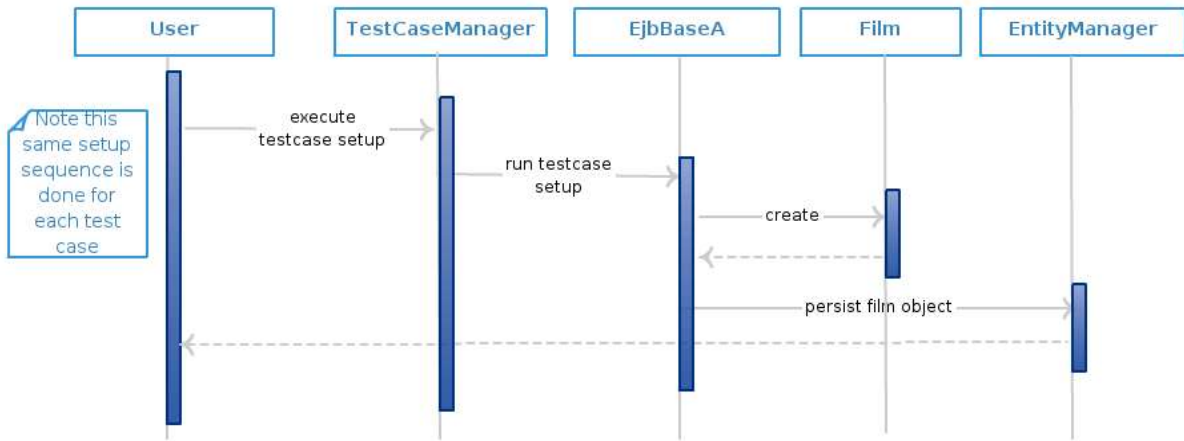
**Fig. 26** test client api class diagram

- TestCase - interface that allows the user to interact with the different EJBs deployed on server A through the web client. This interface contains a *setUp* method that creates a new *Film* object. The *runTest* method executes the actual test case and *tearDown* deletes the *Film* object and *Cast* object created after each test run is executed.

## 3.4.2 Sequence Diagram

### 3.4.2.1 Setup Sequence

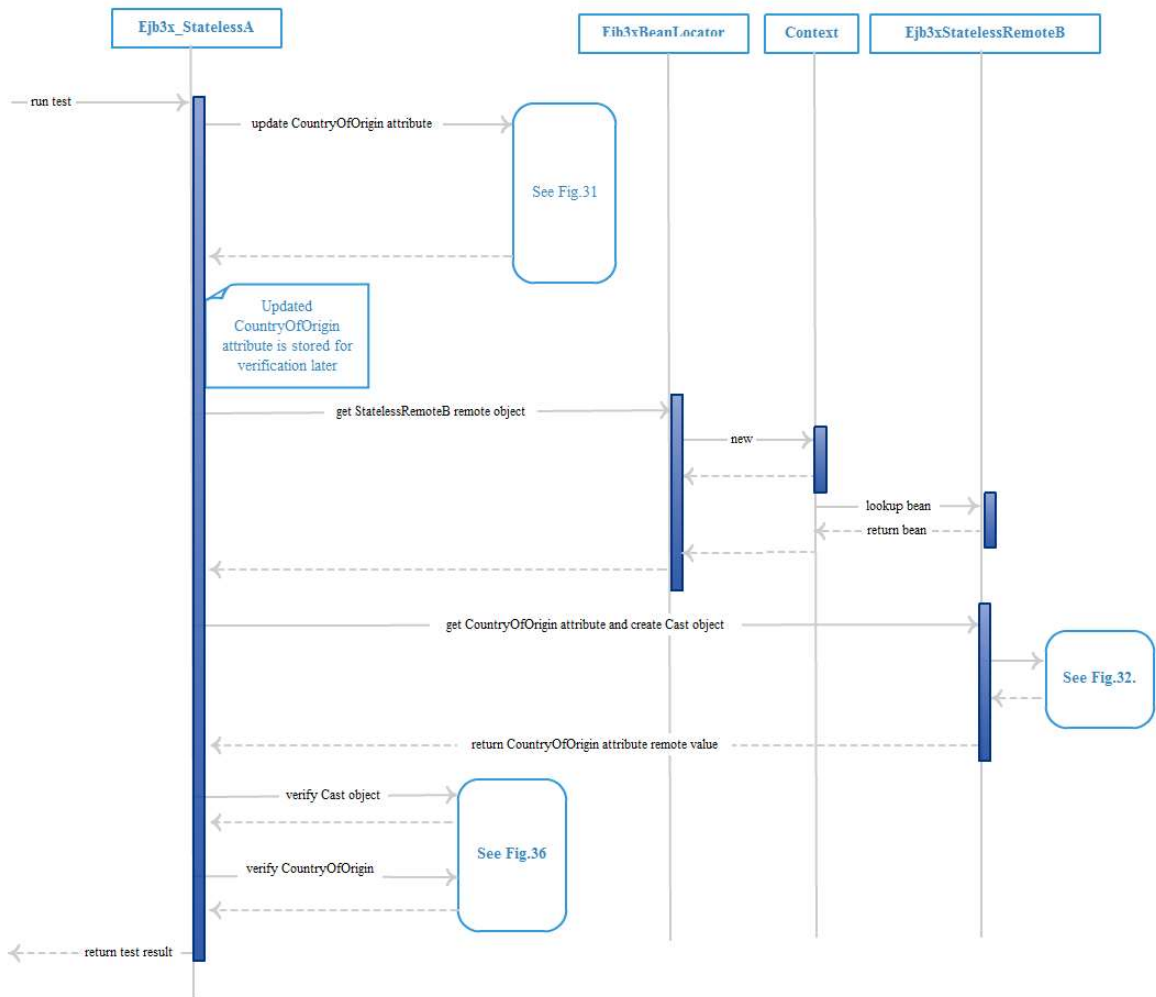
Fig.27 shows the sequence that is executed when the user clicks any of the *Setup TC* buttons in the test client.



**Fig. 27** Setup sequence

### 3.4.2.2 Execute Ejb3x\_StatelessA Test Case Sequence

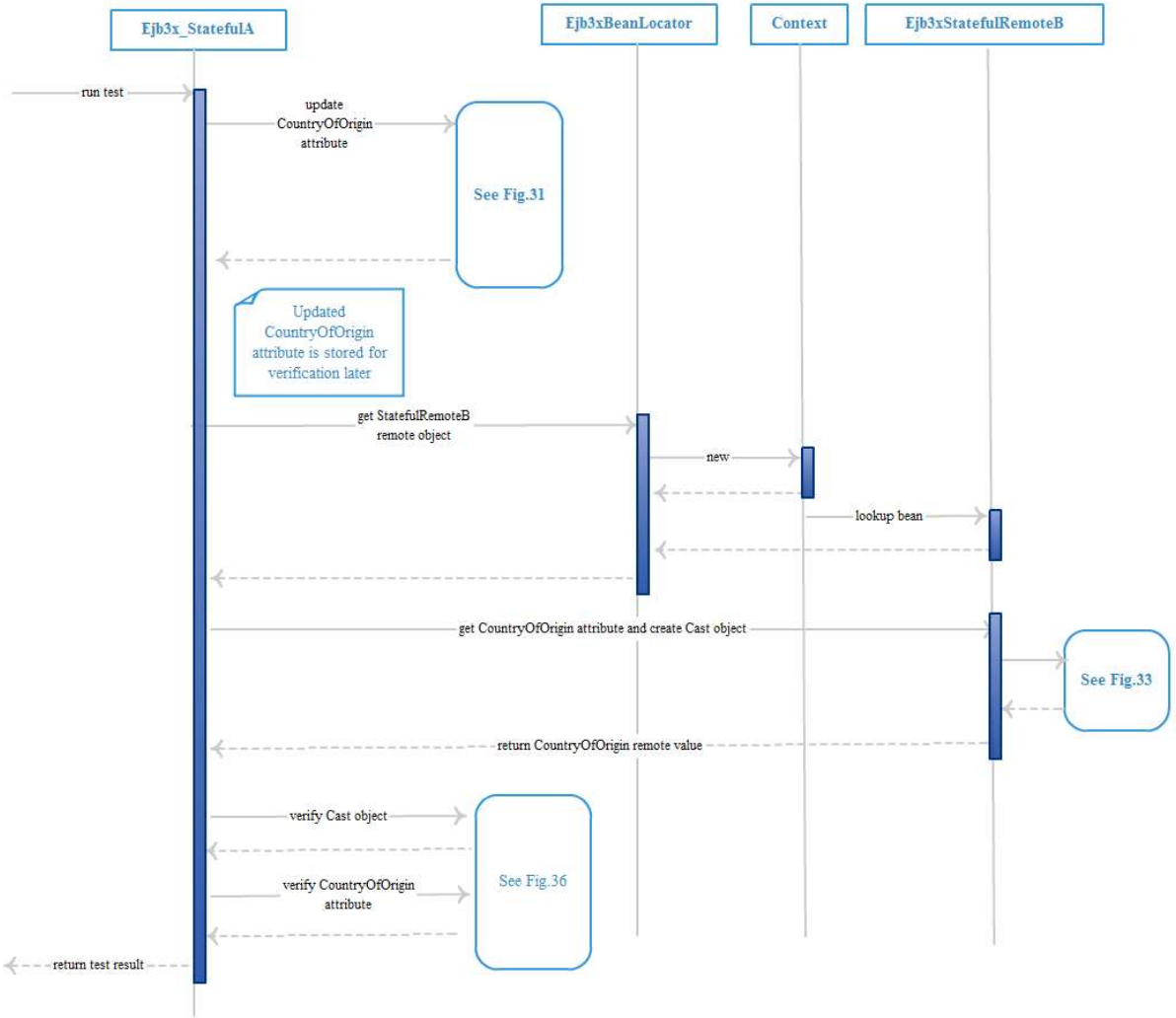
Fig.28 shows the first part of the sequence that is executed when the user clicks the *Execute TC* button of Test Case 1 the EJB3x Stateless bean test. The subsequent sequences are shown in Fig.32, Fig.33 and Fig.37 sequence diagrams.



**Fig 28.** Ejb3x\_StatelessA Run Test Sequence

### 3.4.2.3 Execute Ejb3x\_StatefulA Test Case Sequence

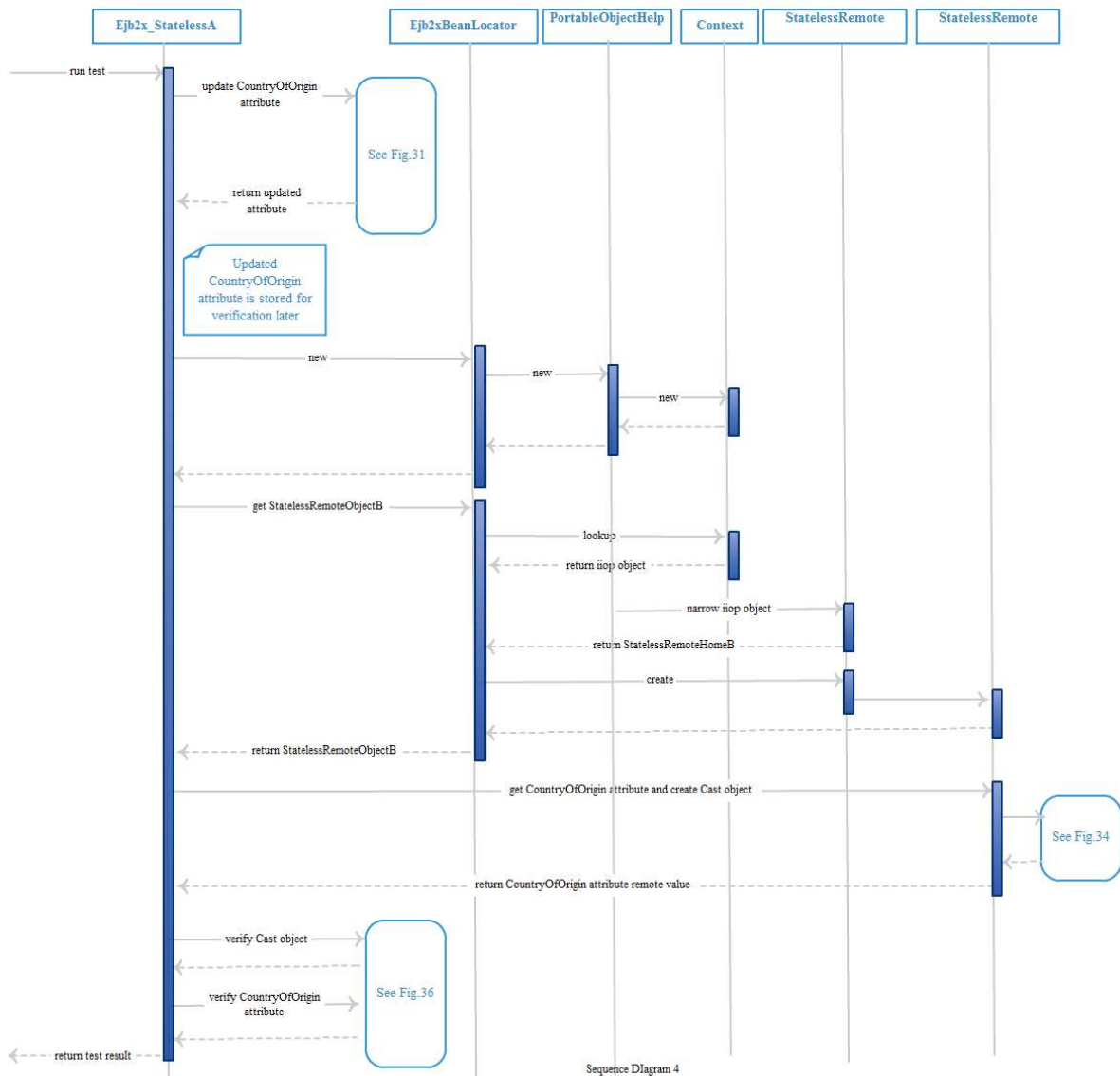
Fig.29 shows the first part of the sequence that is executed when the user clicks the *Execute* TC button of Test Case 2 the EJB3x Stateful bean test. The subsequent sequences are shown in Fig.32, Fig.34 and Fig.37 sequence diagrams.



**Fig.29** Ejb3x\_StatefulA Run Test Sequence

### 3.4.2.4 Execute Ejb2x\_StatelessA Test Case Sequence

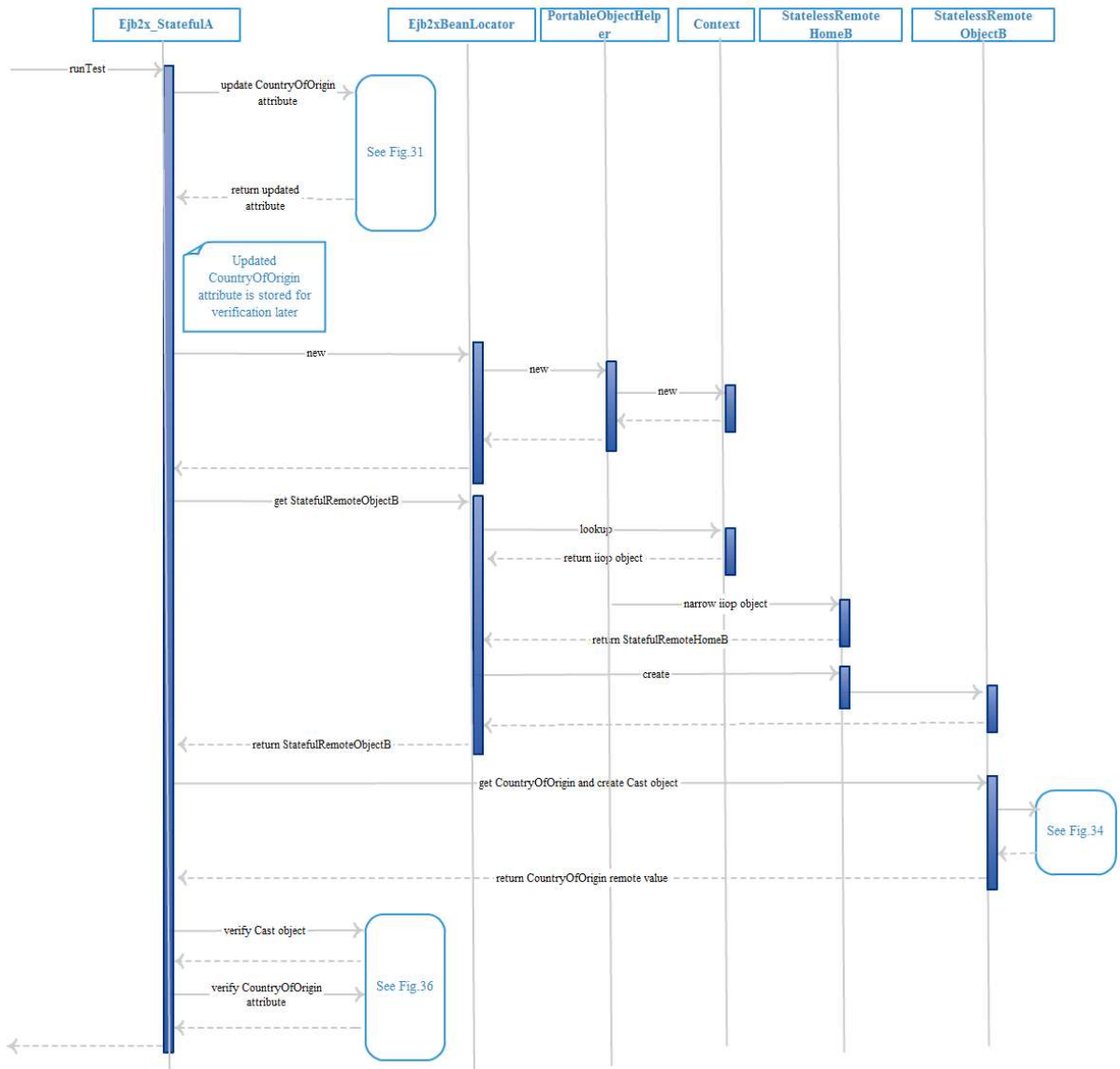
Fig.30 shows the first part of the sequence that is executed when the user clicks the *Execute* TC button of test case 3 the EJB2x Stateless bean test. The subsequent sequences are shown in Fig. Fig.32, Fig.40 and Fig.37 sequence diagrams.



**Fig 30. Ejb2x\_StatelessA Run Test Sequence**

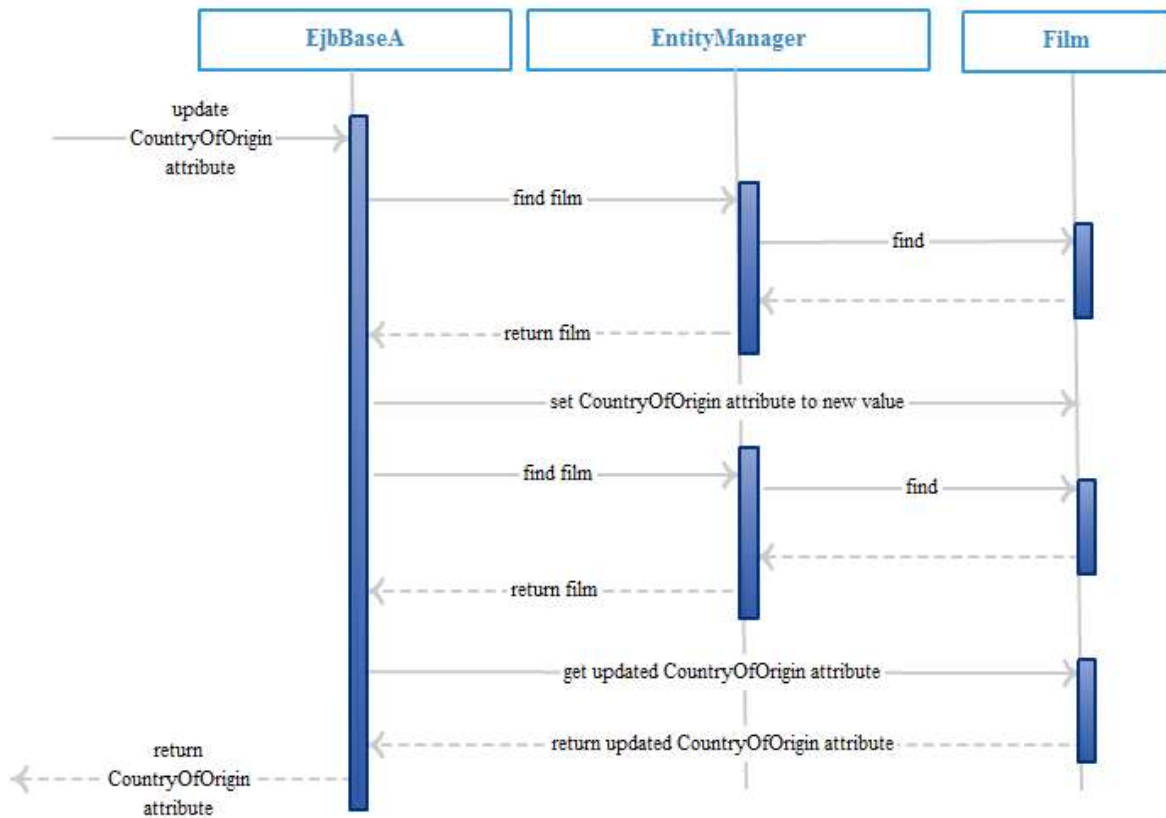
### 3.4.2.5 Execute Ejb2x\_StatefulA Test Case Sequence

Fig.31 shows the first part of the sequence that is executed when the user clicks the *Execute TC* button of test case 4 the EJB2x Stateful bean test. The subsequent sequences are shown in Fig.32, Fig.36 and Fig.37 sequence diagrams.



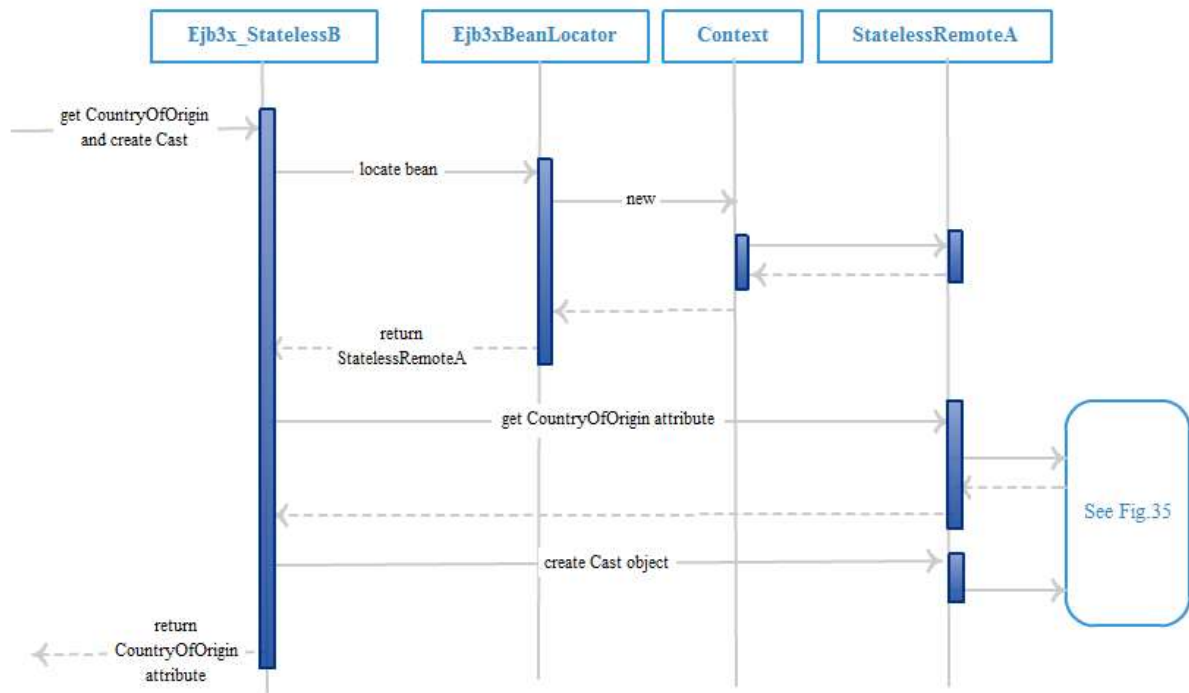
**Fig 31. Ejb2x\_StatefulA Run Test Sequence**

Fig.32 shows the second part of the sequence after the user clicks the *Setup TC* button for all the tests which causes the *CountryOfOrigin* attribute of the *Film* object to be modified.



**Fig 32.** Update CountryOfOrigin Sequence common to all tests

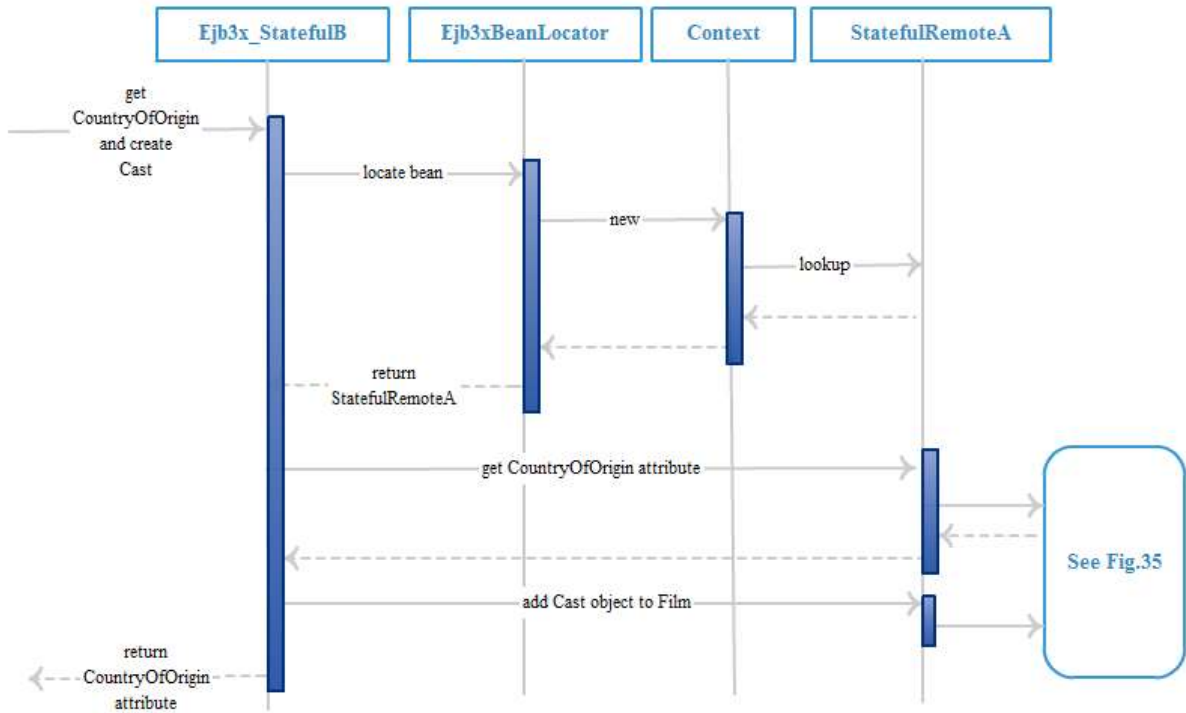
Fig.33 shows the next part of the Execute TC sequence for the EJB3x Stateless bean test where the EJB3x\_StatelessB of server B is remotely invoked from server A.



**Fig 33.** Ejb3x\_StatelessB Get CountryOfOrigin and Create Cast Sequence



Fig.34 shows the next part of the Execute TC sequence for the EJB3.1 Stateful bean test where the EJB3x\_StatefulB of server B is remotely invoked from server A.



**Fig 34.** Ejb3x\_StatefulB Get CountryOfOrigin and Create Cast Sequence

Fig.35 shows the next part of the Execute TC sequence for the EJB2x Stateless bean test where the EJB2x\_StatelessB of server B is remotely invoked from server A.

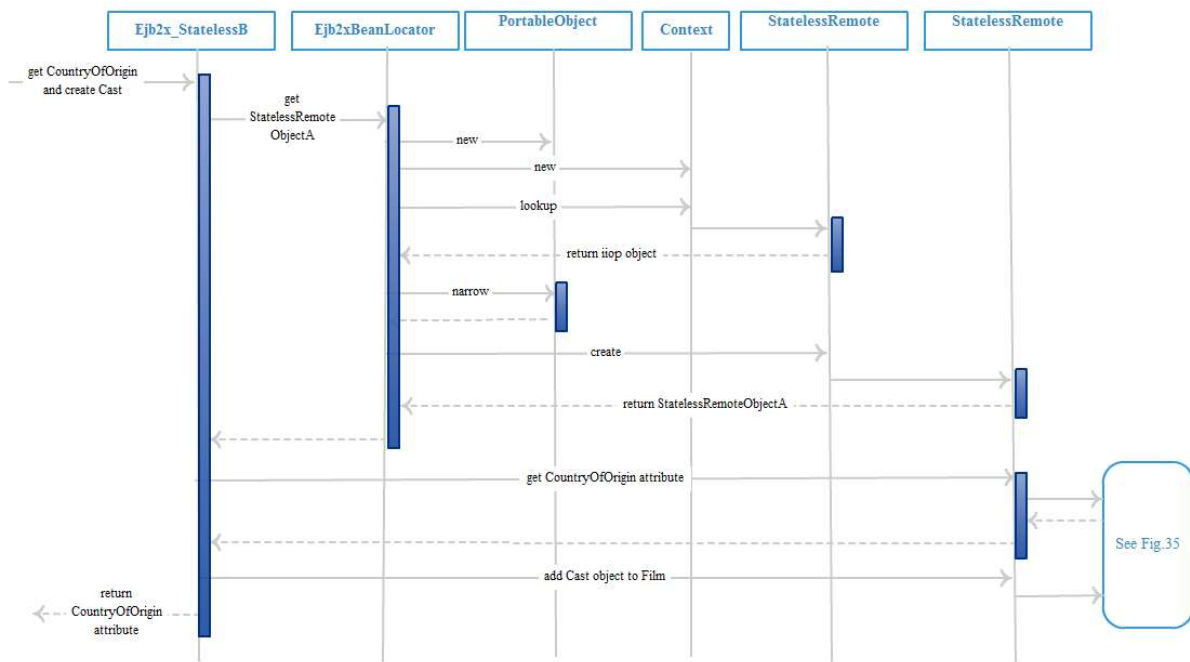
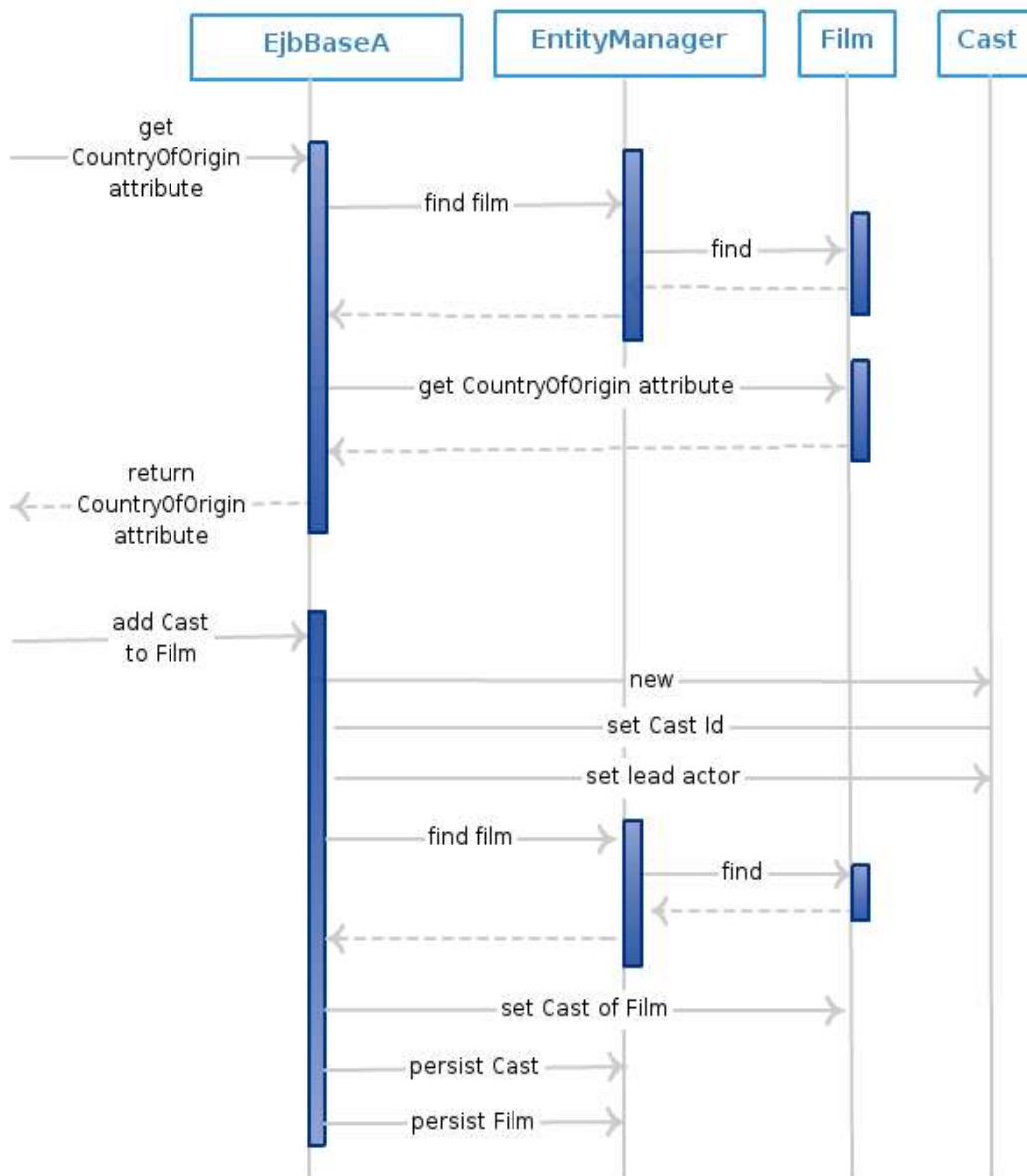


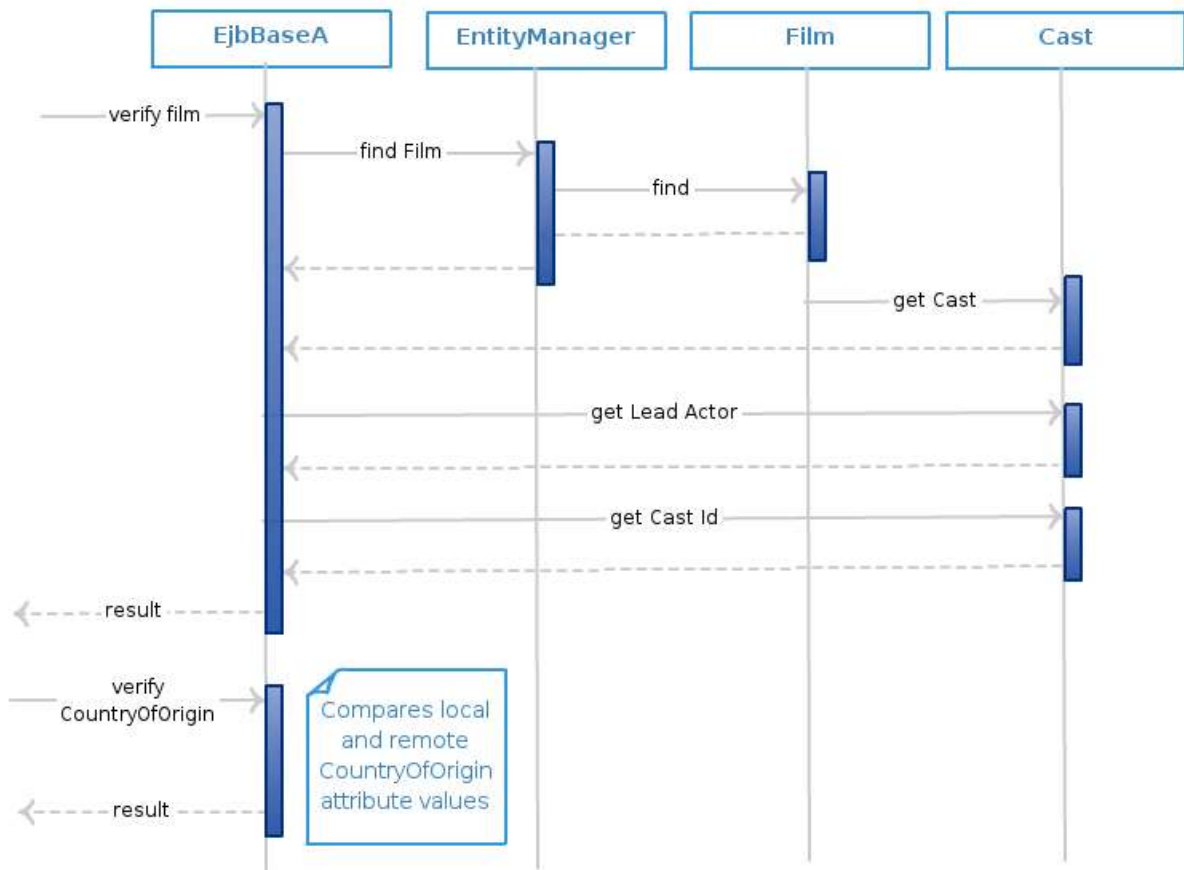
Fig 35. Ejb2x\_StatelessB Get CountryOfOrigin and Create Cast Sequence

Fig.36 shows part of the Execute TC sequence that is common to all the tests where the implementing EJBs of server A are invoked from server B to read the modified *CountryOfOrigin* attribute and create a new *Cast* object associated with the *Film* object.



**Fig 36.** Ejb3x\_StatefulB Get CountryOfOrigin and Create Cast Sequence

Fig.37 shows the verify sequence common to all the tests where the value of the local and remotely fetched *CountryOfOrigin* are compared and also its verified that the *Cast* object exists. If the *CountryOfOrigin* values match and the *Cast* object exists then the test case is successful, anything else is a failed test case.



**Fig 37.** Verify Sequence common to all tests

## 4 System Verification

The system verification was done in 2 steps:

1. Unit testing
2. Integration testing

### 4.1 Unit Testing

The unit testing tests the functionality of the classes in isolation. Unit testing was performed using Junit framework [23] and the mocking framework Mockito [32]. During the development of the unit tests it was necessary to make some minor adjustments to the source code to make the unit testing a bit easier in certain situations; this is a normal refactoring process.

For example the source code contained methods that were creating new instances of objects in the method itself so it made it difficult to inject mocks for those objects as they were no field variables declared for storing references to these objects, so in these cases it was necessary to create a field variable for storing references to these objects which enabled mock classes to be set for these variables during junit testing. See the method `Ejb2x_StatefulA.createBeanFinder()` for an example of this and its test class `Ejb2x_StatefulATest` where the mock object `Ejb2xBeanLocator` is injected in the field `Ejb2xBeanLocator` of `Ejb2x_StatefulA` before the test begins.

The `PortableObjectHelper` class was solely created to help with unit testing by extracting the code that did the corba handling into a separate class which meant that it could be mocked easily in the unit test classes, see `Ejb2x_BeanLocatorTest` for an example of this.


The unit tests can be executed through maven or directly from eclipse. The tests were compiled and executed using the maven command:


```
mvn clean install -o (-o means offline)
```

#### 4.1.1 Unit test results for ejb-module-a

Finished after 0.453 seconds



Runs: 20/20       Errors: 0       Failures: 0


-  test\_runTest\_CreateException (0.256 s)
-  test\_runTest\_RemoteException (0.008 s)
-  test\_getCountryOfOrigin (0.014 s)
-  test\_AddCastToFilm (0.007 s)
-  test\_tearDown (0.006 s)
-  test\_setUp (0.005 s)
-  test\_runTest (0.010 s)
-  test\_runTest\_NamingException (0.010 s)

▼  my.serverA.ejb2.impl.test.Ejb2x\_StatefulATest [Runner: JUnit 4] (0.059 s)

-  test\_runTest\_CreateException (0.012 s)
-  test\_runTest\_RemoteException (0.006 s)
-  test\_getCountryOfOrigin (0.006 s)
-  test\_AddCastToFilm (0.007 s)
-  test\_tearDown (0.007 s)
-  test\_setUp (0.005 s)
-  test\_runTest (0.009 s)
-  test\_runTest\_NamingException (0.007 s)

▼  my.serverA.ejb3.impl.test.Ejb3x\_StatelessATest [Runner: JUnit 4] (0.040 s)

-  test\_runTest (0.037 s)
-  test\_runTest\_NamingException (0.003 s)

▼  my.serverA.ejb3.impl.test.Ejb3x\_StatefulATest [Runner: JUnit 4] (0.023 s)

-  test\_runTest (0.017 s)
-  test\_runTest\_NamingException (0.006 s)

-----  
TESTS

-----  
Running my.serverA.ejb3.impl.test.Ejb3x\_StatelessATest

log4j:WARN No appenders could be found for logger

(my.serverA.ejb3.impl.Ejb3x\_StatelessA).

log4j:WARN Please initialize the log4j system properly.

log4j:WARN See <http://logging.apache.org/log4j/1.2/faq.html#noconfig> for more info.

Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.353 sec

Running my.serverA.ejb3.impl.test.Ejb3x\_StatefulATest

Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.02 sec

Running my.serverA.ejb2.impl.test.Ejb2x\_StatefulATest

Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.076 sec

Running my.serverA.ejb2.impl.test.Ejb2x\_StatelessATest

Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.081 sec

Results :

Tests run: 20, Failures: 0, Errors: 0, Skipped: 0

#### 4.1.2 Unit test results for ejb-module-b

Finished after 0.363 seconds

Runs: 32/32

Errors: 0

Failures: 0

my.serverB.ejb3.impl.test.Ejb3x\_StatefulBTest [Runner: JUnit 4] (0.211 s)

testCreateCast (0.187 s)

testCreateCast\_Locator\_NamingException (0.005 s)







testCreateCast\_StatefulRemoteA\_RemoteException (0.005 s)

testGetCountryOfOrigin (0.004 s)

testGetCountryOfOrigin\_Locator\_NamingException (0.003 s)

testGetCountryOfOrigin\_StatefulRemoteA\_RemoteException (0.007 s)



- ▼  my.serverB.ejb3.impl.test.Ejb3x\_StatelessBTest [Runner: JUnit 4] (0.039 s)
    -  testGetCountryOfOrigin\_StatelessRemoteA\_RemoteException (0.013 s)
    -  testCreateCast\_StatelessRemoteA\_RemoteException (0.005 s)
    -  testCreateCast (0.004 s)
    -  testCreateCast\_Locator\_NamingException (0.003 s)
    -  testGetCountryOfOrigin (0.005 s)
    -  testGetCountryOfOrigin\_Locator\_NamingException (0.009 s)
  
  - ▼  my.serverB.ejb2.impl.test.Ejb2x\_StatefulBTest [Runner: JUnit 4] (0.053 s)
    -  testgetCountryOfOrigin (0.020 s)
    -  testgetCountryOfOrigin\_NamingException (0.002 s)
    -  testgetCountryOfOrigin\_RemoteObjectA\_RemoteException (0.004 s)
    -  testgetCountryOfOrigin\_CreateException (0.003 s)
    -  testgetCountryOfOrigin\_RemoteException (0.004 s)
    -  testCreateCast\_Locator\_CreateException (0.004 s)
    -  testCreateCast\_Locator\_RemoteException (0.003 s)
    -  testCreateCast\_remoteObjectA\_RemoteException (0.002 s)
    -  testCreateCast (0.003 s)
    -  testCreateCast\_Locator\_NamingException (0.007 s)
  
  - ▼  my.serverB.ejb2.impl.test.Ejb2x\_StatelessBTest [Runner: JUnit 4] (0.043 s)
    -  testgetCountryOfOrigin (0.009 s)
    -  testgetCountryOfOrigin\_NamingException (0.003 s)
    -  testgetCountryOfOrigin\_RemoteObjectA\_RemoteException (0.004 s)
    -  testgetCountryOfOrigin\_CreateException (0.003 s)
    -  testgetCountryOfOrigin\_RemoteException (0.004 s)
    -  testCreateCast\_Locator\_CreateException (0.003 s)
    -  testCreateCast\_Locator\_RemoteException (0.003 s)
    -  testCreateCast\_remoteObjectA\_RemoteException (0.013 s)
    -  testCreateCast (0.000 s)
    -  testCreateCast\_Locator\_NamingException (0.001 s)
- 

-----

## TESTS

-----

Running my.serverB.ejb3.impl.test.Ejb3x\_StatefulBTest

log4j:WARN No appenders could be found for logger  
(my.serverB.ejb3.impl.Ejb3x\_StatefulB).



log4j:WARN Please initialize the log4j system properly.

log4j:WARN See <http://logging.apache.org/log4j/1.2/faq.html#noconfig> for more info.

Tests run: 6, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.298 sec

Running my.serverB.ejb3.impl.test.Ejb3x\_StatelessBTest

Tests run: 6, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.035 sec

Running my.serverB.ejb2.impl.test.Ejb2x\_StatefulBTest

Tests run: 10, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.061 sec

Running my.serverB.ejb2.impl.test.Ejb2x\_StatelessBTest

Tests run: 10, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.06 sec

Results :

Tests run: 32, Failures: 0, Errors: 0, Skipped: 0

#### 4.1.3 Unit test results for remote-api

Finished after 0.272 seconds

Runs: 5/5    Errors: 0    Failures: 0

- my.remote.bean.locator.test.Ejb2x\_BeanLocatorTest [Runner: JUnit 4] (0.083 s)
  - testConstructor (0.031 s)
  - testGetRemoteObjectA (0.027 s)
  - testGetRemoteObjectB (0.025 s)
- my.remote.bean.locator.test.Ejb3x\_BeanLocatorTest [Runner: JUnit 4] (0.018 s)
  - testLocateBean (0.016 s)
  - testLocateBean\_NamingException (0.002 s)

-----

## TESTS

---

Running my.remote.bean.locator.test.Ejb3x\_BeanLocatorTest

Apr 13, 2015 3:52:45 PM my.remote.bean.locator.Ejb3xBeanLocator locateBean

INFO: Located bean successfully.

Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.241 sec

Running my.remote.bean.locator.test.Ejb2x\_BeanLocatorTest

log4j:WARN No appenders could be found for logger

(my.remote.bean.locator.Ejb2xBeanLocator).

log4j:WARN Please initialize the log4j system properly.

log4j:WARN See <http://logging.apache.org/log4j/1.2/faq.html#noconfig> for more info.

Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.111 sec

Results :

Tests run: 5, Failures: 0, Errors: 0, Skipped: 0

## 4.2 Integration Testing using Arquillian

The integration tests in this system test the API's of the complete system by deploying the EAR files *ear-module-a* and *ear-module-b* in running JBoss application server containers. All the test cases are controlled by software that uses a combination of the JBoss Arquillian test framework [24] and Junit [23] to execute the tests.

The JBoss Arquillian framework is an open source framework that is very powerful to use as it provides a means for testing that the deployment of artifacts such as EAR/JAR/WAR files are trouble free and that any server configuration settings are configured correctly. It also tests that the JEE wiring semantics e.g. annotations for CDI, EJB transaction handling, session handling, interceptors are correct which is something that cannot be done using straight forward unit testing as these tests do not execute in a running applications server so cannot test the overall functionality of the system. For example, an arquillian test can deploy the artifacts to be tested, then deploy a testable artifact like a WAR file that contains the test cases and execute the tests towards these deployed artifacts, and then finally undeploy everything when all the tests have been ran. The JBoss ShrinkWrap API [33] is also heavily used by arquillian tests to created JAR,WAR and EAR files.

The integration tests are all contained in a single test class called `PersistenceContextScopeTest.java`. The test is triggered from maven using a plugin called the *maven-surefire-plugin* and by running the maven command:

```
mvn clean install -o
```

This command will clean the output target directory, compile the code and execute the arquillian test. The output of the command above shows that all the test cases passed as show here:

```
Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 73.929 sec - in  
my.arquillian.test.PersistenceContextScopeTest
```

Results :

```
Tests run: 8, Failures: 0, Errors: 0, Skipped: 0
```

[INFO]

[INFO] --- maven-jar-plugin:2.3.2:jar (default-jar) @ wookie-integ-test ---

[WARNING] JAR will be empty - no content was marked for inclusion!

[INFO] Building jar: /home/eeikkah/git/wookie-integ-test/target/wookie-integ-test-1.0-SNAPSHOT.jar

[INFO]

[INFO] --- maven-install-plugin:2.3.1:install (default-install) @ wookie-integ-test ---

[INFO] Installing /home/eeikkah/git/wookie-integ-test/target/wookie-integ-test-1.0-SNAPSHOT.jar to /home/eeikkah/.m2/repository/my/arquillian/test/wookie-integ-test/1.0-SNAPSHOT/wookie-integ-test-1.0-SNAPSHOT.jar

[INFO] Installing /home/eeikkah/git/wookie-integ-test/pom.xml to /home/eeikkah/.m2/repository/my/arquillian/test/wookie-integ-test/1.0-SNAPSHOT/wookie-integ-test-1.0-SNAPSHOT.pom

[INFO] -----

[INFO] BUILD SUCCESS

[INFO] -----

[INFO] Total time: 1:59.925s

[INFO] Finished at: Wed Apr 15 21:01:21 IST 2015

[INFO] Final Memory: 27M/436M

[INFO] -----

#### 4.2.1 Test environment

One of the main required files for running tests using the Arquillian framework is the arquillian.xml file. Arquillian works by looking for this file on the classpath. This file tells the

arquillian test what applications servers to start as part of the test, what are the servers names so that can be referenced as a target by the test cases and any properties to use. The arquillian.xml used by this integration test in this system tells it to start 4 applications servers forming 2 test environments *Test Env1* and *Test Env2* as shown in Fig.37. Both environments have 2 JBoss application servers running that are configured for remote EJB outbound communication with each other.

The integration test PersistenceContextScopeTest needs 4 servers in total running; 2 with JTS enabled (Env1) and 2 without (Env2). As all the servers are running on the same local machine it is necessary to start them using different port offsets to avoid port clashes that would prevent the startup of the servers.

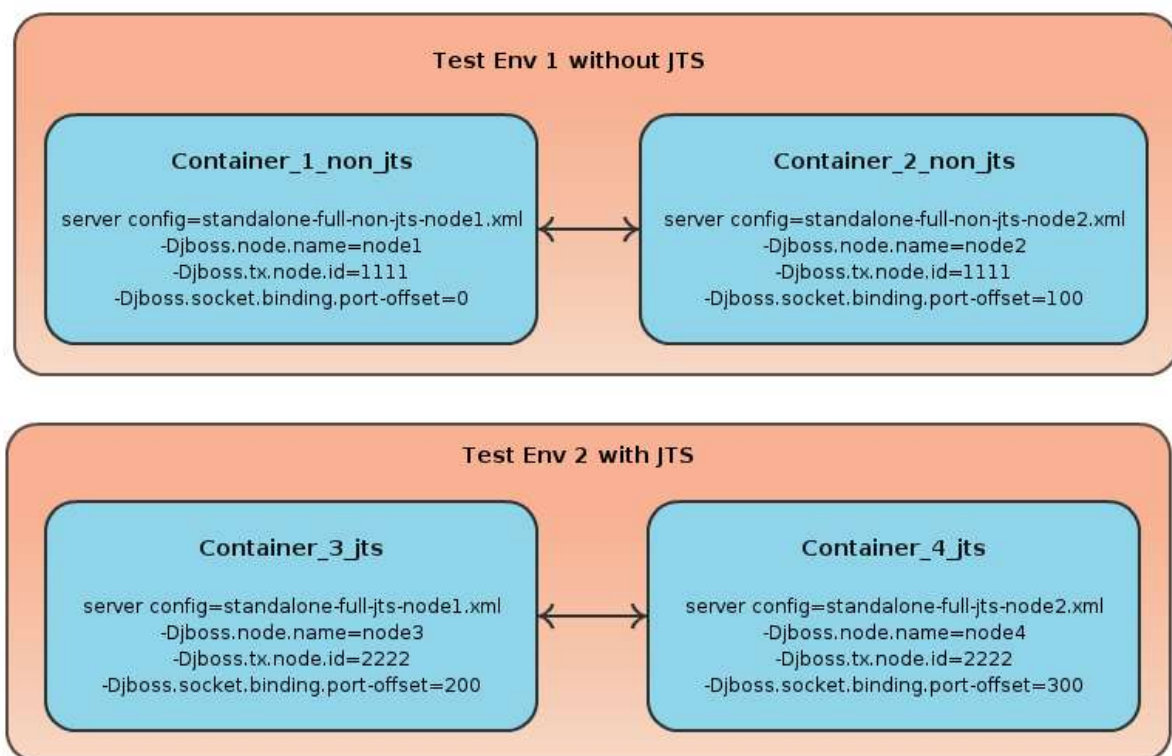


Fig.38 Overview of the 2 Arquillian Test Environments

- **Container\_1\_non\_jts**

This server has a socket binding port offset of 0. This server has a remote ejb socket binding connection configured to be able to communicate to *Container\_2\_non\_jts*. As *Container\_2\_non\_jts* is started with a port offset of 100 then the standard port of 4447 is incremented by 100.

```
<outbound-socket-binding name="remote-ejb">  
    <remote-destination host="localhost" port="4547"/>  
</outbound-socket-binding>
```

- **Container\_2\_non\_jts**

This server has a socket binding port offset of 100. This server has a remote ejb socket binding connection configured to be able to communicate to *Container\_1\_non\_jts*. As *Container\_1\_non\_jts* is started with a port offset of 0 then the standard port of 4447 does not need to be changed.

```
<outbound-socket-binding name="remote-ejb2">  
    <remote-destination host="localhost" port="4447"/>  
</outbound-socket-binding>
```

- **Container\_3\_jts**

This server has a socket binding port offset of 200. This server has a remote ejb socket binding connection configured to be able to communicate to *Container\_4\_jts*. As *Container\_4\_non\_jts* is started with a port offset of 300 then the standard port of 4447 needs to be incremented by 300. The server is also configured for JTS.

```
<outbound-socket-binding name="remote-ejb">
  <remote-destination host="localhost" port="4747"/>
</outbound-socket-binding>
```

- **Container\_4\_jts**

This server has a socket binding port offset of 300. This server has a remote ejb socket binding connection configured to be able to communicate to *Container\_3\_jts*. As *Container\_3\_non\_jts* is started with a port offset of 200 then the standard port of 4447 needs to be incremented by 200. The server is also configured for JTS.

```
<outbound-socket-binding name="remote-ejb2">
  <remote-destination host="localhost" port="4647"/>
</outbound-socket-binding>
```

Here is the arquillian.xml that is used by PersistenceContextScopeTest to start the 4 servers:

```
<?xml version="1.0" encoding="UTF-8"?>
<arquillian xmlns="http://jboss.org/schema/arquillian"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://jboss.org/schema/arquillian
http://jboss.org/schema/arquillian/arquillian_1_0.xsd">

  <defaultProtocol type="Servlet 3.0" />
```

```

<engine>

    <property name="deploymentExportPath">${export.path}</property>

</engine>

<group qualifier="jboss_managed">

    <container qualifier="Container_1_non_jts" default="false">

        <configuration>

            <property name="jbossHome">${jboss.home}</property>

            <property name="serverConfig">standalone-full-non-jts-
node1.xml</property>

            <property name="javaVmArguments">-Xmx1024m -
XX:MaxPermSize=512m

            -Djava.net.preferIPv4Stack=true

            -Djboss.node.name=node1

            -Djboss.tx.node.id=1111

            -Djboss.server.log.dir=${jboss.home}/standalone/log

            -Dcom.arjuna.orbportability.initialReferencesRoot=${jboss.home}

        </property>

        <property name="managementPort">9999</property>

    </configuration>

</container>

```



```

<container qualifier="Container_2_non_jts" default="false">

    <configuration>

        <property name="jbossHome">${jboss.home}</property>

        <property name="serverConfig">standalone-full-non-jts-
node2.xml</property>

        <property name="javaVmArguments">-Xmx1024m -
XX:MaxPermSize=512m

        -Djava.net.preferIPv4Stack=true

        -Djboss.server.base.dir=${jboss.home}/standalone2

        -Djboss.socket.binding.port-offset=100

        -Djboss.node.name=node2

        -Djboss.tx.node.id=1111

        -Djboss.server.log.dir=${jboss.home}/standalone2/log

    </property>

        <property name="managementPort">10099</property>

    </configuration>

</container>

<container qualifier="Container_3_jts" default="false">

    <configuration>

        <property name="jbossHome">${jboss.home}</property>

        <property name="serverConfig">standalone-full-jts-
node1.xml</property>

```

```

        <property name="javaVmArguments">-Xmx512m -
XX:MaxPermSize=256m

        -Djava.net.preferIPv4Stack=true

        -Djboss.server.base.dir=${jboss.home}/standalone3

        -Djboss.socket.binding.port-offset=200

        -Djboss.node.name=node3

        -Djboss.tx.node.id=2222

        -Djboss.server.log.dir=${jboss.home}/standalone3/log

        -Dcom.arjuna.orbportability.initialReferencesRoot=${jboss.home}
    </property>

    <property name="managementPort">10199</property>

</configuration>

</container>

<container qualifier="Container_4_jts" default="false">

    <configuration>

        <property name="jbossHome">${jboss.home}</property>

        <property name="serverConfig">standalone-full-jts-
node2.xml</property>

        <property name="javaVmArguments">-Xmx512m -
XX:MaxPermSize=256m

        -Djava.net.preferIPv4Stack=true

        -Djboss.server.base.dir=${jboss.home}/standalone4

        -Djboss.socket.binding.port-offset=300

```

```
-Djboss.node.name=node4

-Djboss.tx.node.id=2222

-Djboss.server.log.dir=${jboss.home}/standalone4/log

</property>

<property name="managementPort">10299</property>

</configuration>

</container>

</group>

</arquillian>
```

#### 4.2.2 Test steps

There are a number of setup and configuration steps that are done before the actual arquillian test is executed to set up the test environment described above. The different maven plugins in the pom.xml are used to carry out steps 1-10 below before the maven-surefire-plugin is used to start the actual execution of the arquillian test:

1. First the target directory is cleaned of any files:

```
[INFO] --- maven-clean-plugin:2.4.1:clean (default-clean) @ wookie-integ-test ---
```

```
[INFO] Deleting /home/eeikkah/git/wookie-integ-test/target
```

2. Next the jboss tar file is extracted to the target directory:

```
[INFO] --- maven-dependency-plugin:2.1:unpack (unpack_jboss) @ wookie-integ-test
```

[INFO] Configured Artifact: com.ericsson.oss.itpf.eap:jboss-temp-tar:2.1.9:tar.gz

[INFO] Unpacking /home/eeikkah/.m2/repository/com/ericsson/oss/itpf/eap/jboss-temp-tar/2.1.9/jboss-temp-tar-2.1.9.tar.gz to

/home/eeikkah/git/wookie-integ-test/target/jboss-eap

with includes null and excludes:null

[INFO] Expanding /home/eeikkah/.m2/repository/com/ericsson/oss/itpf/eap/jboss-temp-tar/2.1.9/jboss-temp-tar-2.1.9.tar.gz to /tmp/tmp1112378441374362097.tar

[INFO] Expanding: /tmp/tmp1112378441374362097.tar into /home/eeikkah/git/wookie-integ-test/target/jboss-eap

3. Next a new *standalone2* directory is created by making a copy of the *standalone* directory that was extracted in the previous step:

[INFO] --- maven-resources-plugin:2.6:copy-resources (create-standalone2-directory) @ wookie-integ-test ---

[INFO] Using 'UTF-8' encoding to copy filtered resources.

[INFO] Copying 11 resources

4. Similarly a new *standalone3* directory is created:

[INFO] --- maven-resources-plugin:2.6:copy-resources (create-standalone3-directory) @ wookie-integ-test ---

[INFO] Using 'UTF-8' encoding to copy filtered resources.

[INFO] Copying 11 resources

5. Similarly a new *standalone4* directory is created:

[INFO] --- maven-resources-plugin:2.6:copy-resources (create-standalone4-directory) @ wookie-integ-test ---

[INFO] Using 'UTF-8' encoding to copy filtered resources.

[INFO] Copying 11 resources

6. The *application-users.properties* file is copied to all 4 standalone configuration directories as this file contains the user and password for handling the ejb security realm to enable ejb remoting communication between the servers:

[INFO] --- maven-resources-plugin:2.6:copy-resources (copy-application-users-properties-standalone) @ wookie-integ-test ---

[INFO] Using 'UTF-8' encoding to copy filtered resources.

[INFO] Copying 1 resource

[INFO]

[INFO] --- maven-resources-plugin:2.6:copy-resources (copy-application-users-properties-standalone2) @ wookie-integ-test ---

[INFO] Using 'UTF-8' encoding to copy filtered resources.

[INFO] Copying 1 resource

[INFO]

[INFO] --- maven-resources-plugin:2.6:copy-resources (copy-application-users-properties-standalone3) @ wookie-integ-test ---

[INFO] Using 'UTF-8' encoding to copy filtered resources.

[INFO] Copying 1 resource

[INFO]

[INFO] --- maven-resources-plugin:2.6:copy-resources (copy-application-users-properties-standalone4) @ wookie-integ-test ---

[INFO] Using 'UTF-8' encoding to copy filtered resources.

[INFO] Copying 1 resource

7. The *standalone-full-non-jts-node1.xml* file is copied to the *standalone/configuration* directory:

[INFO] --- maven-resources-plugin:2.6:copy-resources (copy-full-non-jts-node1-config-to-standalone) @ wookie-integ-test ---

[INFO] Using 'UTF-8' encoding to copy filtered resources.

[INFO] Copying 1 resource

8. The *standalone-full-non-jts-node2.xml* file is copied to the *standalone/configuration2* directory:

[INFO] --- maven-resources-plugin:2.6:copy-resources (copy-full-non-jts-node2-config-to-standalone2) @ wookie-integ-test ---

[INFO] Using 'UTF-8' encoding to copy filtered resources.

[INFO] Copying 1 resource

9. The *standalone-full-jts-node1.xml* file is copied to the *standalone/configuration3* directory:

[INFO] --- maven-resources-plugin:2.6:copy-resources (copy-full-jts-node1-config-to-standalone3) @ wookie-integ-test ---

[INFO] Using 'UTF-8' encoding to copy filtered resources.

[INFO] Copying 1 resource

10. The *standalone-full-jts-node2.xml* file is copied to the *standalone/configuration4* directory:

[INFO] --- maven-resources-plugin:2.6:copy-resources (copy-full-jts-node2-config-to-standalone4) @ wookie-integ-test ---

[INFO] Using 'UTF-8' encoding to copy filtered resources.

[INFO] Copying 1 resource

#### 4.2.3 Test Cases

There is a single test class `PersistenceContextScopeTest.java` that is added to a test EAR created using the ShrinkWrap API that is deployed in containers `Container_1_non_jts` and `Container_3_jts` only that enables the test cases to be executed. Here is a description of each of the 8 test cases:

#	Test Case Name	Test Case Description	Expected Result
1	test_Ejb3x_StatelessA_Without_Jts	Tests when the PersistenceContext is injected into stateless EJB3.1 and deployed in container with no JTS active and has EJB remote connections configured between containers.	EJBTransactionRolledbackException
2	test_Ejb3x_StatefulA_Without_Jts	Tests when the PersistenceContext is injected into stateful EJB3.1 and deployed in container with no JTS active and has EJB remote connections configured between containers.	EJBTransactionRolledbackException

3	test_Ejb2x_StatelessA_Without_Jts	Tests when the PersistenceContext is injected into stateless EJB3.1 that declared to use the EJB2.0 remoting specification and configured to use CORBA IIOp endpoints for communication that deployed in container with no JTS active.	ERROR no adapter activator exists
4	test_Ejb2x_StatefulA_Without_Jts	Tests when the PersistenceContext is injected into stateful EJB3.1 that declared to use the EJB2.0 remoting specification and configured to use CORBA IIOp endpoints for communication that deployed in container with no JTS active.	ERROR no adapter activator exists
5	test_Ejb3x_StatelessA_With_Jts	Tests when the PersistenceContext is injected into stateless EJB3.1 and deployed in container with JTS active and has EJB remote connections configured between the containers.	EJBTransactionRolledback Exception



6	test_Ejb3x_StatefulA_With_Jts	Tests when the PersistenceContext is injected into stateful EJB3.1 and deployed in container with JTS active and has EJB remote connections configured between the containers.	EJBTransactionRolledback Exception
7	test_Ejb2x_StatelessA_With_Jts	Tests when the PersistenceContext is injected into stateless EJB3.1 that's declared to use the EJB2.0 remoting specification and configured to use CORBA IIOP endpoints for communication that's deployed in container with JTS active.	Successful
8	test_Ejb2x_StatefulA_With_Jts	Tests when the PersistenceContext is injected into stateful EJB3.1 that's declared to use the EJB2.0 remoting specification and configured to use CORBA IIOP endpoints for communication that's deployed in container with no JTS active.	Successful

#### 4.2.4 Test Results

This table lists the integration test results:

#	Test Case Name	Expected Result	Test Result
1	test_Ejb3x_StatelessA_Without_Jts	Failed	Passed
2	test_Ejb3x_StatefulA_Without_Jts	Failed	Passed
3	test_Ejb2x_StatelessA_Without_Jts	Failed	Passed
4	test_Ejb2x_StatefulA_Without_Jts	Failed	Passed
5	test_Ejb3x_StatelessA_With_Jts	Failed	Passed
6	test_Ejb3x_StatefulA_With_Jts	Failed	Passed
7	test_Ejb2x_StatelessA_With_Jts	Passed	Passed
8	test_Ejb2x_StatefulA_With_Jts	Passed	Passed

The results show that all 8 integration tests pass with the expected outputs but only tests 7 & 8 pass with the desired results.

## 5 Summary of Results and Conclusions

This work evaluated backward compatibility issues with EJB 3.X and EJB 2.X. It illustrated that an issue occurs when an EJB3.x (EJB A) bean is deployed on an application server A, which is the only bean in the system that has access to the underlying database. The proposed solution uses configuration changes to the standalone xml of the JBOSS application servers to enable JTS over IIOP transport, telling the application server that EJB3.x beans will be using the EJB2.x client views by declaring so using the jboss-ejb3.xml, changing the business method interfaces of the EJBs to extend EJBHome and EJBObject interfaces and to expose these interfaces via IIOP.

The results presented show that for test cases 1, 2, 5 & 6 which use pure EJB3.x beans, and with JTS and non JTS enabled application servers, that the outcome was the same. The server B log file shows that the EJB3.x beans do not retrieve the modified attribute *CountryOfOrigin* value of 'IRELAND' from server A but instead get the value 'USA' which is the value of the *CountryOfOrigin* attribute before it was modified i.e. the value it is created with during the test setup. This means that even though these EJB3.x beans join the transaction and became participants of the transaction, as they have the ability to rollback the transaction, they are not able to access the same persistence context which had the modified Film object attribute *CountryOfOrigin* with value 'IRELAND'. Instead they can only get the *Film* object before the transaction was started through a different persistence context. Interestingly in all cases the additional invocation method back to server A to create a child Cast object under the parent Film object fails when the business method returns due to a session handling error and the transaction is rolled back. This can be seen clearly from the server B log:

### Server B log

```
20:31:04,247 INFO [my.serverB.ejb3.impl.Ejb3x_StatelessB][getCountryOfOrigin:43] [Server-B]
received CountryOfOrigin value [USA] from [Server-A]
```

```
20:31:04,274 INFO [my.serverB.ejb3.impl.Ejb3x_StatelessB][createCast:55] [Server-B] createCast
invoked with id [1]
```

20:31:04,277 INFO [my.remote.bean.locator.Ejb3xBeanLocator][locateBean:33] Located bean successfully.

20:31:04,324 INFO [org.jboss.ejb.client.remoting][resultReady:213] EJBCLIENT000011: Discarding result for invocation id 2 since no waiting context found

20:31:04,354 ERROR [my.serverB.ejb3.impl.Ejb3x\_StatelessB][createCast:60] [Server-B] [javax.ejb.EJBTransactionRolledbackException], exception msg [a different object with the same identifier value was already associated with the session: [my.database.entity.Cast#2]]

20:31:04,355 ERROR [org.jboss.as.ejb3][handleInCallerTx:144] javax.ejb.EJBTransactionRolledbackException: a different object with the same identifier value was already associated with the session: [my.database.entity.Cast#2].

The results for test cases 3 & 4 which use EJB3.x beans that implement the EJB2.x client view in non JTS enabled application servers show that they too also fail to get the modified attribute value 'IRELAND' but instead retrieve the value 'USA' from server A. This shows also that these beans do not get access to the same persistence context belonging to the transaction started in server A. However no exception is thrown in these tests and so the transaction started by server A is not rolled back which allows the test to verify that the Cast object has not been created and the modified attribute value is not returned from server B.

### **Server B log**

20:32:20,364 INFO [my.serverB.ejb2.impl.Ejb2x\_StatelessB][getCountryOfOrigin:49] [Server-B] received CountryOfOrigin value [USA] from [Server-A]

20:32:20,367 INFO [my.serverB.ejb2.impl.Ejb2x\_StatelessB][createCast:59] [Server-B] createCast invoked with id [1]

### **Server A log**

21:30:47,019 INFO [my.serverA.common.EjbBaseA][verifyCast:127] [Server-A] verifying Cast

21:30:47,020 INFO [my.serverA.common.EjbBaseA][find:152] [Server-A] find object with id [1]

21:30:47,021 INFO [my.serverA.common.EjbBaseA][verifyCast:138] [Server-A] verify cast result passed? [false]

Results presented illustrated there were only 2 test cases, test case 7 **test\_Ejb2x\_StatelessA\_With\_Jts** and test case 8 **test\_Ejb2x\_StatefulA\_With\_Jts** which were successfully executed. Both these tests had EJB3.1 session beans that were implemented to use the EJB2.x client views for remote EJB communication, had their remote interfaces registered as IOP endpoints in the Corba naming service and had the JTS enabled in the containers. Only when these conditions were met was a solution to the problem found, where transactional changes made to the entity and its associated persistence context on server A accessible within the same transaction by server B - or to put it in simpler terms, server B was remotely able to see the changes made within the transaction of the persistence context started by server A, and become a participant of the same transaction and create a new object within the same transaction.

#### **Server B log**

20:46:29,581 INFO [my.serverB.ejb2.impl.Ejb2x\_StatelessB][getCountryOfOrigin:49] [Server-B] received CountryOfOrigin value [Ireland] from [Server-A]

20:46:29,589 INFO [my.serverB.ejb2.impl.Ejb2x\_StatelessB][createCast:59] [Server-B] createCast invoked with id [1]

#### **Server A log**

20:46:29,617 INFO [my.serverA.common.EjbBaseA][verifyCast:127] [Server-A] verifying Cast

20:46:29,618 INFO [my.serverA.common.EjbBaseA][find:152] [Server-A] find object with id [1]

20:46:29,619 INFO [my.serverA.common.EjbBaseA][verifyCast:138] [Server-A] **verify cast result passed? [true]**

20:46:29,620 INFO [my.serverA.common.EjbBaseA][verifyCountryOfOrigin:143] [Server-A] verifying CountryOfOriginAttribute, localValue [Ireland] remoteValue [Ireland]

20:46:29,621 INFO [my.serverA.common.EjbBaseA][verifyCountryOfOrigin:147] [Server-A] **verify CountryOfOrigin passed? [true]**

It was thought that using EJB3.1 stateful session beans may have been a viable solution to the problem as the state of the field variable for the persistence context would be kept but this turned out not to be the case as even then the EJB of server B was not able to get a handle on the same PersistenceContext when it remotely called back to server A.

Also it was found during testing that EJB3.1 singleton sessions beans cannot be used as they do not support having EJB 2.X views as the following error is seen when attempting to deploy them in the container - *"Singleton beans cannot have EJB 2.x views"*. This makes sense as singleton session beans are new to EJB3.x and did not exist in the EJB2.x specification.

A solution to the transaction scope persistence context use case has been demonstrated by the application in this study running on RedHat JBOSS AS 7.1 servers [15] as it supports legacy compatibility with EJB2 and indeed this is the case for later versions of JBOSS AS 7.x as they support the JEE6 specification. In the JEE7 specification the support for EJB2.x is made optional meaning vendors are not required to support it. The JEE7 specification is supported by RedHat with their WildFly application servers [25].

The JEE8 specification is not fully defined yet but the latest information gives no guarantees that support for EJB2.x client view will make the final draft, *"In accordance with the pruning process defined by the Java EE 6 specification, we will consider designating the following as Proposed Optional in this release: the EJB 2.x client view APIs (EJBObject, EJBHome, EJBLocalObject, EJBLocalHome) and support for CORBA IIOP interoperability."* Therefore it is possible that future vendor application servers will no longer support this. This means that any product solution in this area would have to be re-designed in order to be able to continue functioning so that it could take advantage of newer technology, which is a natural progression for software and would of course incur an unwanted maintenance cost along the way which could be quite costly.

## 6 References

- [1] JSR 151: Java™ 2 Platform, Enterprise Edition 1.4 (J2EE 1.4) Specification  
<https://jcp.org/en/jsr/detail?id=151>
- [2] Sun Technologies  
<http://www.suntechnologies.com/>
- [3] JSR 153: Enterprise JavaBeans 2.1  
<https://jcp.org/en/jsr/detail?id=153>
- [4] JSR 244: JEE5 Specification  
<https://jcp.org/en/jsr/detail?id=244>
- [5] Apache Maven  
<http://maven.apache.org/>
- [6] Spring Framework  
<http://projects.spring.io/spring-framework/>
- [7] JSR 220: Enterprise Java Beans 3.0  
<https://jcp.org/en/jsr/detail?id=220>
- [8] “EJB 3.0 Migration”, An Oracle White Paper, October 2005
- [9] Java Persistence API  
<http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html>
- [10] JSR 316: JEE6 Specification

- <https://jcp.org/en/jsr/detail?id=316>
- [11] JSR 318: Enterprise Java Beans 3.1  
<https://jcp.org/aboutJava/communityprocess/final/jsr318/>
- [12] JSR 314: Java Server Faces 2.0  
<https://www.jcp.org/en/jsr/detail?id=314>
- [13] JSR 317: Java Persistence 2.0  
<https://jcp.org/en/jsr/detail?id=317>
- [14] JBoss Application Server 7 - JBoss Community  
<http://jbossas.jboss.org/>
- [15] JBOSS AS 7.1.0 Final Release Notes  
<https://developer.jboss.org/wiki/AS710FinalReleaseNotes>
- [16] JSR 907: Java Transaction API  
<https://jcp.org/en/jsr/detail?id=907>
- [17] The Java EE 6 Tutorial  
<http://docs.oracle.com/javaee/6/tutorial/doc/docinfo.html>
- [18] Java Transaction Service, Sun Micro Systems  
<http://www.oracle.com/technetwork/java/javaee/jts-spec095-1508547.pdf>
- [19] Object Management Group, Object Transaction Service  
<http://www.omg.org/cgi-bin/doc?formal/2003-09-02>
- [20] Object Management Group, CORBA Service  
<http://www.corba.org/>
- [21] Object Management Group, IIOP



- <http://www.omg.org/library/iop4.html>
- [22] JBoss AS 5  
<http://jbossas.jboss.org/docs/5-x.html>
- [23] Junit Framework  
<http://junit.org/>
- [24] Arquillian Framework  
<http://arquillian.org/>
- [25] RedHat Wildfly  
<http://wildfly.org/>
- [26] JSR 342: JEE 7 Specification  
<https://jcp.org/en/jsr/detail?id=342>
- [27] JSR 366: JEE8 Specification  
<https://jcp.org/en/jsr/detail?id=366>
- [28] Ziyu Wang; Minghui Zhou; Donggang Cao; Haiyan Zhao, "Dual-Container: Extending the EJB2.x Container to Support EJB3.0," Computer Software and Applications Conference, 2009
- [24] Johnson, R., "J2EE development frameworks," *Computer*, vol.38, no.1, pp.107,110, Jan. 2005
- [25] Deron Liang; Win-Tsung Lo; Kao, Y.M.; Yuan, S.M.; Yue-Shan Chang, "A fault tolerant object transaction service in CORBA," Computer Software and Applications Conference, 1997. COMPSAC '97. Proceedings., The Twenty-First Annual International, vol., no., pp.115,120, 11-15 Aug 1997

- [26] Ram, P.; Do, L.; Drew, P.; Tong Zhou, "Object Transaction Service: experiences and open issues," Distributed Objects and Applications, 1999. Proceedings of the International Symposium on , vol., no., pp.296,304, 1999
- [27] Frolund, S.; Guerraoui, R., "e-Transactions: end-to-end reliability for three-tier architectures," Software Engineering, IEEE Transactions on , vol.28, no.4, pp.378,395, Apr 2002
- [28] Huaigu Wu; Kemme, B., "Fault-tolerance for stateful application servers in the presence of advanced transactions patterns," Reliable Distributed Systems, 2005. SRDS 2005. 24th IEEE Symposium on , vol., no., pp.95,105, 26-28 Oct. 2005
- [29] Felber, P.; Narasimhan, P., "Experiences, strategies, and challenges in building fault-tolerant CORBA systems," Computers, IEEE Transactions on , vol.53, no.5, pp.497,511, May 2004
- [30] Lin Zuo; Shaohua Liu; Jun Wei, "A Fault-Tolerant Scheme for Complex Transaction Patterns in J2EE," Enterprise Distributed Object Computing Conference, 2006. EDOC '06. 10th IEEE International , vol., no., pp.165,174, Oct. 2006
- [31] JBOSS H2 Database Tutorial  
<http://www.mastertheboss.com/jboss-server/jboss-datasource/h2-database-tutorial>
- [32] Mockito Framework  
<http://docs.mockito.googlecode.com/hg/org/mockito/Mockito.html>
- [33] Shrinkwrap API  
<http://docs.jboss.org/shrinkwrap/1.0.0-beta-6/>

## 6.1 GitHub - Source Code

- The complete EJB source code is stored in a GitHub repository called **wookie-prototype** and can be viewed and cloned by the command:

- git clone [git@github.com:kevinkav/wookie-prototype.git](https://github.com/kevinkav/wookie-prototype.git)

- The complete integration test source code is also stored in a Git Hub repository called **wookie-integ-test** and can be viewed and cloned by the command:

- git clone [git@github.com:kevinkav/wookie-integ-test.git](https://github.com/kevinkav/wookie-integ-test.git)