PRODYNA information architects

PRODYNA ESD

PRODYNA Architecture Certification

# PAC Conference – Service Architecture

10.11.2013 v1.0

Martin Monshausen

# Change history

| Version | Date | Author | Change description | Chapter |
|---|---|---|---|---|
| **1.0** | 10.11.2013 | Martin Monshausen | Creation of initial Document | All |
| | | | | |

# Contents

# 1. Overview

The purpose for the application “PAC Conference” is to provide a conference platform for end-users and administrators.

End-Users are able to browse conferences and dive into details of their locations, rooms, talks and speakers.

Administrators are able to manage locations, conferences, rooms, speakers and talks. The application allows them to create, update and delete all kinds of locations, conferences, rooms, speakers and talks.

The basic architecture of the application follows the three-tier approach (Model-View-Controller) based upon Java EE 6 technology stack.

There are two different client implementations both using injected services:

- a JSF 2 web frontend
- a REST-client usable for mobile applications or as integration into existing applications. As transport format JSON has been chosen.

As Java EE 6 compatible application server, JBoss AS 7.1 was chosen. This application server fully supports the Java EE 6 specification, allows clustering and provides the possibility to upgrade to an enterprise version including support if necessary later on.

JBoss application server provides several default implementations for Java EE 6 standards. The project uses these default implementations because they are well integrated into JBoss. Relevant to this project are Hibernate 4 as implementation for JPA 2, Mojarra 2 as implementation for JSF 2, Weld as implementation for CDI, HornetQ as implementation for JMS, RESTeasy as implementation for JAX-RS and Jackson for JSON serialization and deserialization.

As database server, MySQL 5.5 is used. This database server is a very mature and provides the possibility to upgrade to an enterprise version including support if necessary later on. MySQL is integrated into JBoss as module and the connection is exposed as JNDI datasource; this allows changing database backend easily if requirements changes and another database should be used.

The following figure shows the deployment architecture of the project. The artifacts and execution environments are shortly described in the following table.

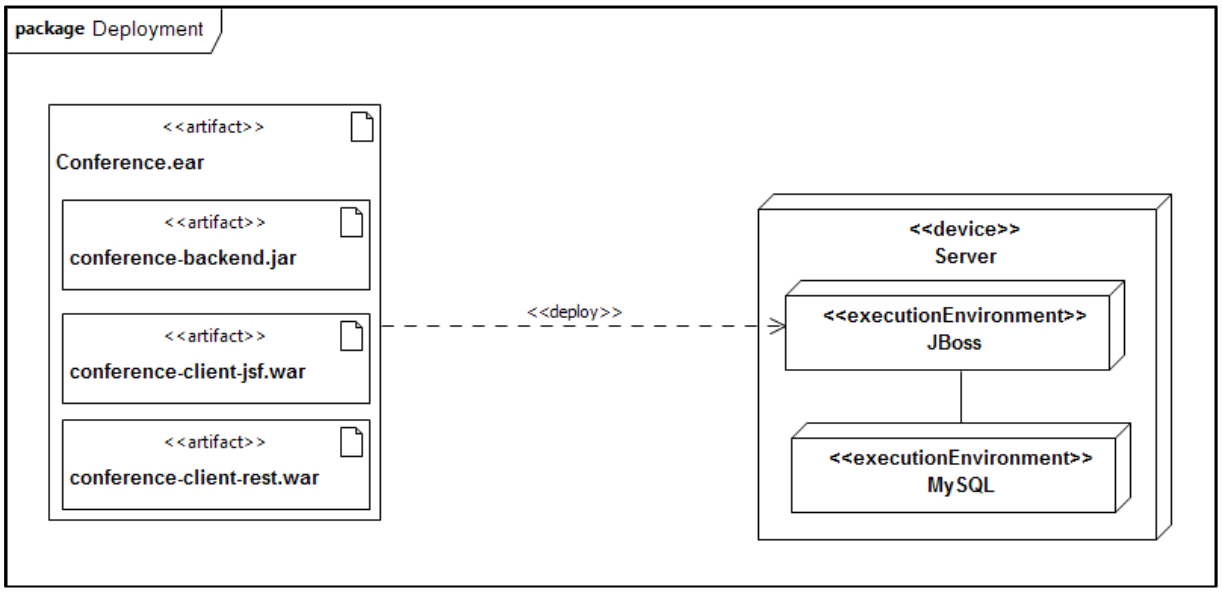


Figure 1: architecture overview

| name | description |
|---|---|
| conference.ear | Artifact which contains all the subprojects of the project and all libraries not provided by application server.<br>Deployed to JBoss server. |
| conference-backend.jar | Artifact which contains the enties and its crud-services.<br>Part of conference.ear |
| conference-client-jsf.war | Artifact which contains the JSF frontend, including facelets and controllers.<br>Part of conference.ear |
| conference-client-rest.war | Artifact which contains the REST services.<br>Part of conference.ear |
| JBoss | Application server used for deployment of this application. Provides Mojarra as implementation for JSF, Hibernate as implementation for JPA, HornetQ as implementation for JMS and RESTeasy as implementation for REST. |
| MySQL | Database backend used to persist entities.<br>Integrated into JBoss as module; exposed as JNDI datasource |

# 2. Component Architecture

The application is divided into the backend and the two client implementations. The following figure shows the components and their dependencies. The following table shortly describes each component.

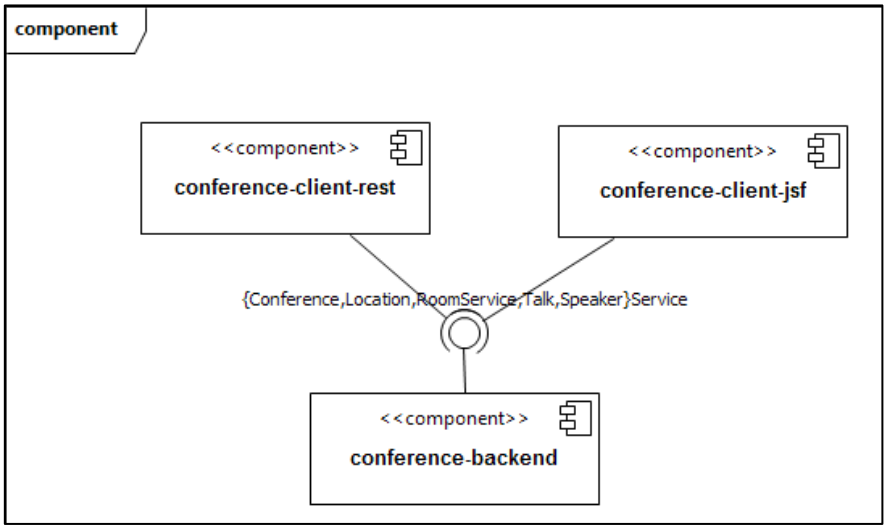


Figure 2: component architecture

| name | description |
|---|---|
| conference-backend | Component which contains the entities and its crud-services.<br>Services are exposed through service interfaces and implemented by stateless session beans.<br>Beside the client services there are also additional backend services for performance measuring, eventing and logging.<br>Responsible for persistence services. |
| conference-client-jsf | Component which contains the JSF2 facelets, controllers and resource files.<br>Uses conference-backend to query for entities, create enties, update entities and delete entities.<br>Service interfaces of the conference-backend services are injected using CDI in controller classes. |
| conference-client-rest | Component which contains the REST services.<br>Uses conference-backend to query for entities, create enties, update entities and delete entities.<br>Service interfaces of the conference-backend services are injected using CDI in REST classes. |

## 2.1 Conference-backend

The component conference-backend itself consists of several sub-components. The following figure shows these components and their dependencies. The following table describes the sub-components shortly.

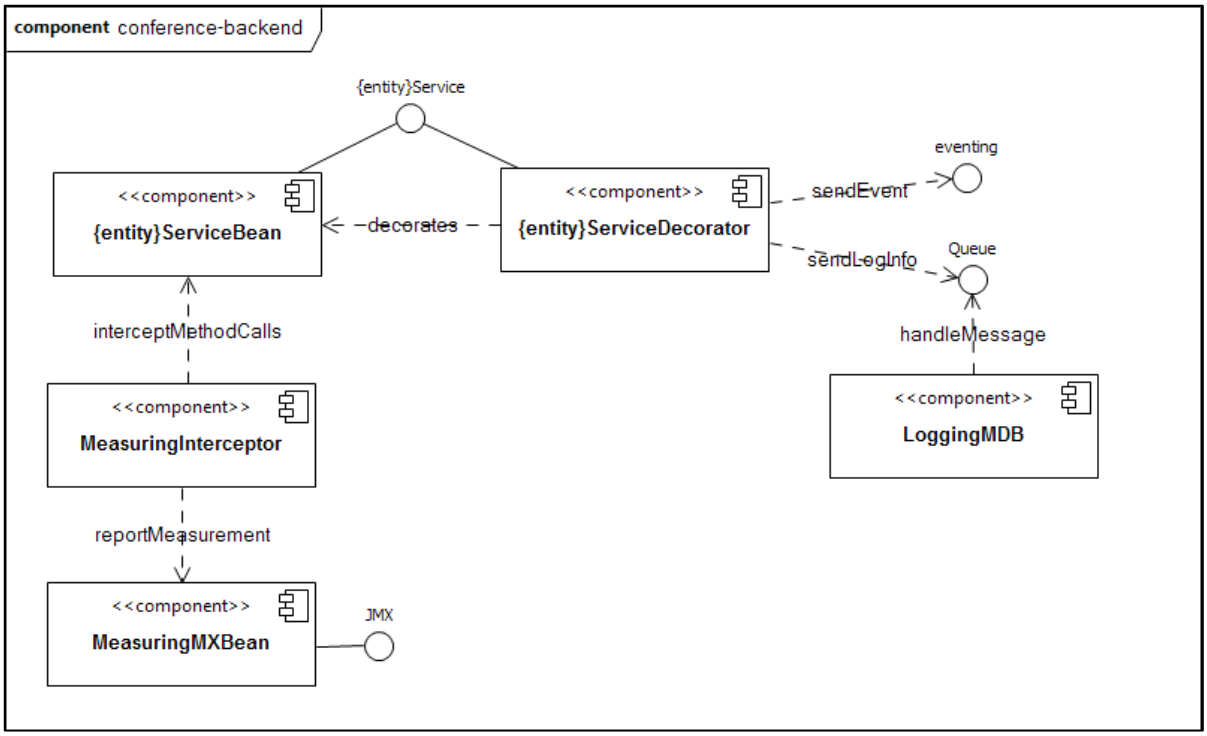


Figure 3: conference-backends sub-components

| name | description |
|---|---|
| {entity}Service | Service interface for entity {entity}.<br>Contains all service methods for that entity. |
| {entity}ServiceBean | Implementation of {entity}s service methods. |
| {entity}ServiceDecorator | Decorates {entity}Service implementations.<br>Sends information about method calls to queue and fires events in case of operations modifying entities.<br>Delegates methods calls to {entity}ServiceBean |
| MeasuringInterceptor | Intercepts all method calls and measures how long method invocation takes.<br>Measurement results are reported to an MXBean. |
| MeasuringMXBean | Stores information about method runtimes and exposes these information via JMX to JMX clients |
| LoggingMDB | MessageDriven listening on a queue and logs messages from queue to log file.<br>Queue messages contain information about Method calls. |

### 2.1.1 Domain model

The domain model is part of the component conference-backend.

The following figure shows the domain model.

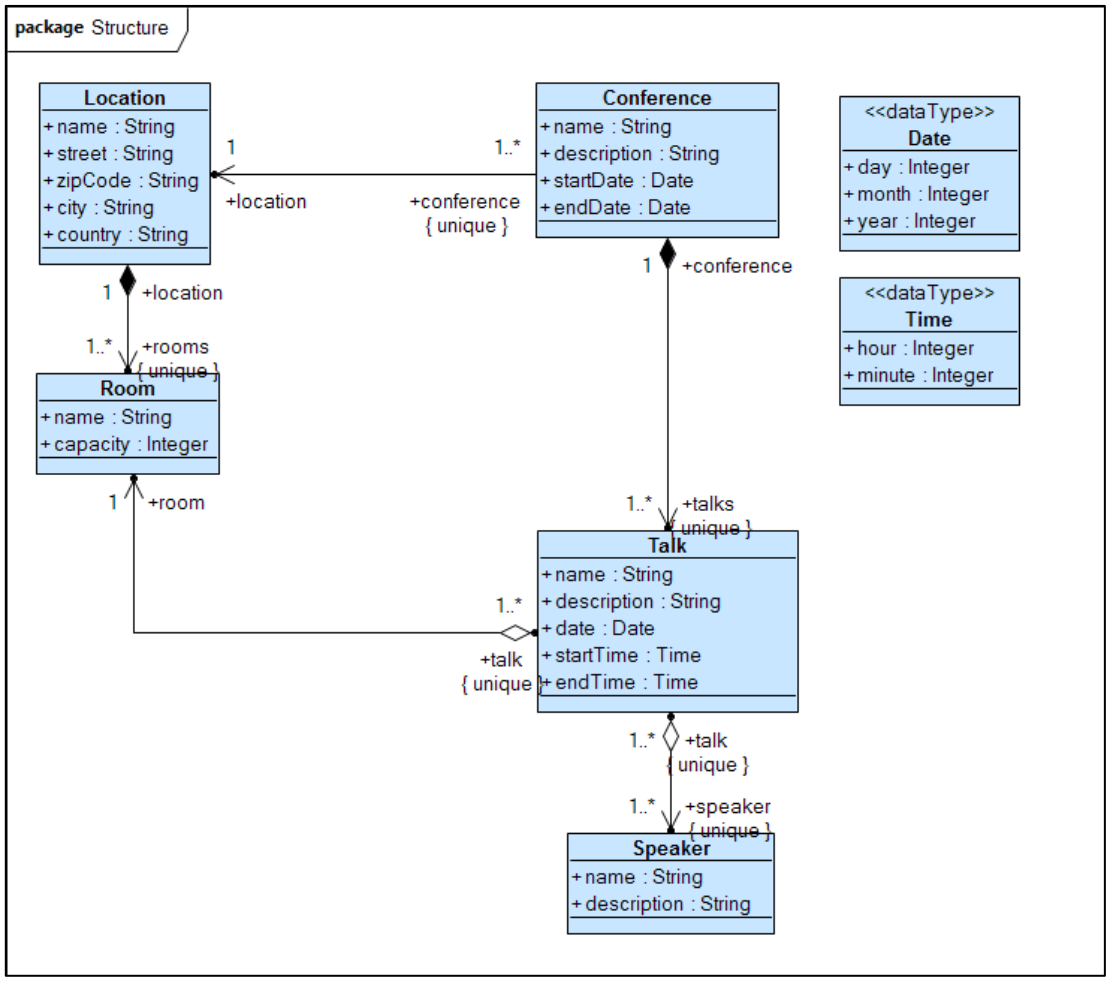


Figure 4: domain model

It is only allowed to use one-directional associations in entity classes. Bi-directional associations would cause problems in for example REST-services because lists of associated objects contain objects with back-references; that would cause endless loops.

The entities of the domain model also include validation logic ensuring that the correct data will be stored in database. In case of validation constraint violations, an exception will be thrown, which has to be handled by clients.

## 2.2 Conference-client-rest

The REST services uses JAX-RS annotations to define the mapping to HTTP specifics like URL, parameter mappings and response handling.

This layer is necessary to separate client logic and backend logic. The backend services contain no REST-specific annotations. Only the REST services contain these annotations.

The REST services delegate the calls to the appropriate EJBs from the backend which will be injected via CDI. Therefore, the REST API does not have dependencies to any implementation modules but only references their public interface.

The REST-Services return special response objects which allow setting the HTTP return code besides the entity.

### 2.3 Conference-client-jsf

The JSF client is realized using JSF2. In order to keep technology stack lean, no additional taglibs or component libraries are used.

Access to EJB services from conference-backend is provided by CDI using service interfaces. Entities returned by backend services are directly usable in facelets as they are POJOs.

For each entity there is a facelet display information to the end-user. Additionally there is a facelet for each entity to allow administrators to manage (create, update and delete) the entity.

For both – the facelet to display information to the end-user and the facelet allowing administrators to manage – are using an entity JSF-controller.

For entity lists there is a list producer, caching the results and therefore reducing backend calls. The list producer updates itself in case of changes to the entities. For this purpose the list producer listens for events fired by the backend decorators.

# 3. Development Guideline

## 3.1 Development environment

It is recommended that the developers of the project use Eclipse Kepler (Eclipse for Java EE Developers) as IDE.
Other IDEs like IntelliJ or NetBeans would also be possible, but no support could be provided for those.

Furthermore the JBoss Tools Plugin for Eclipse should be installed. This plugin provides wizards for JSF, JPA validation and adds recent JBoss application server versions to the list of supported runtime environments.

Build and dependency management is done using Maven 3. Eclipse offers built-in support for Maven, allowing managing and retrieving dependencies and running maven builds.

Source Control is done using git. Developers could work locally but have to push their results to central master branch periodically.

Central Git repository is hosted on GitHub (repository link). GitHub allows browsing the sources, viewing the statistics and getting information from the project wiki.

Based on pushes to central master branch, continuous build server Travis builds the projects and notifies the committer if he broke the build. The build history could be found here (Travis link).

All java classes have to comply with the following coding conventions:

1. Java source files have to be UTF-8 encoded with Unix style line delimiter
2. Java code has to be formatted using Eclipse code formatter template
3. Java sources have to comply with the official Java Coding Conventions
4. Interfaces and each method being public have to be documented using JavaDoc
5. All classes have to include at least JavaDoc explaining the purpose of the class

## 3.2 Project structure

The project consists of a maven project with maven modules as child.

The following figure visualizes the structure and shows all maven modules of the project.

The following table describes the maven modules and the maven project shortly.

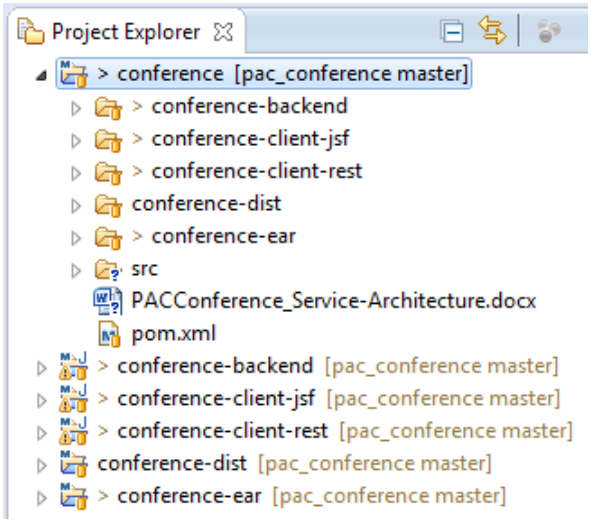


Figure 5: project structure

| name | description |
|---|---|
| **conference** | Maven project. Parent for all maven modules.<br>POM contains common properties and provides profiles for build, test and release.<br>Maven modules inherit properties from parent project. |
| **conference-backend** | Maven module<br>Contains the services for managing conferences and all related entities.<br>POM declares dependencies to libraries and provides profiles for build, test and release. |
| **conference-client-jsf** | Maven module<br>Contains the implementation for the JSF frontend.<br>POM declares a dependency to conference-backend and libraries; it provides profiles for build, test and release.<br>JSF UI will be accessible through<br>• /conference-client-jsf/index.jsf and<br>• /conference-client-jsf/admin/index.jsf |
| **conference-client-rest** | Maven module<br>Contains the implementation for the REST-services.<br>POM declares **dependency** to conference-backend and libraries; it provides profiles for build, test and release.<br>REST-services will be accessible through<br>• /conference-client-rest/rest/{serviceName} |

| name | description |
|---|---|
| **conference-dist** | Maven module<br>This module is used to build the final release unit.<br>This module performs the following steps:<br>• build and package project as ear<br>• run all tests and include test results<br>• add source files<br>• add sql files for create or update database tables<br>• generate JavaDoc<br>The module packages all artifacts into one single unit. |
| **conference-ear** | Maven module<br>This module packages all implementation modules to one single EAR file.<br>The ear file is ready to be deployed to the application server and contains all classes and libraries (as far as they are not provided by the application server). |

### 3.3 Test

All backend and REST services will be tested using Arquillian; a framework which allows testing EJB applications in a managed JBoss-application-server-environment. All services provided by JBoss in real-world are also available in tests, so no mocks are required for testing.

Arquillian provides two test modes:

1. Testing backend service in a managed application environment
2. Testing frontend in client mode

For testing conference-backend services, the first approach was chosen. All public service methods were tested.

For testing REST services, the second approach was chosen. All public REST service methods were tested.

### 3.4 Naming conventions

The base package structure for all packages is:

```
com.prodyna.pac.mmonshausen.conference
```

Based on the base structure the following naming conventions have to be followed:

| artifact | Package name |
|---|---|
| **Domain object** | {basePackage}.model .{CamelCaseName} |
| **Service interface** | {basePackage}.service.{CamelCaseName}Service |
| **Service impl.** | {basePackage}.service.{CamelCaseName}ServiceImpl |
| **Decorator** | {basePackage}.service.{CamelCaseName}ServiceDecorator |

# Appendix 1 – Technology Stack

| Name | Version |
|---|---|
| **Arquillian** | 1.1.0.Final |
| **Eclipse Kepler** | 4.3.1 SR 1 |
| **Hibernate** | 4.0.1 |
| **HornetQ** | 2.2.13 |
| **Jackson JSON processor** | 1.9.2 |
| **Java** | 1.6 |
| **JBoss AS (Java EE 6 Full Profile)** | 7.1.1.Final |
| **JBoss Tools** | 4.1 |
| **Maven** | 3.0.5 |
| **Mojarra** | 2.1.7 |
| **MySQL** | 5.5 |
| **RESTEasy** | 2.3.2 |
| **Shrinkwrap** | 2.0.0 |
| **Validation API** | 1.0.0.GA |