TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Software and Multimedia Technology, Software Technology Group

SS 2021 – Component-based Software Engineering

Implementing Component-Based Systems–Part I

Lectuer: Dr. Sebastian Götz
Tutor: Markus Hamann

Task 1 Transparency Problems (T)

A transparency problem describes software concerns that should be transparent (invisible, hidden) when you write or deploy a component. This task repeats the different kinds of transparency problems.

a) What can be subject of secrets wrt. transparency problems of component-based systems?

Solution:
Content secrets are secrets that deal with the concrete implementation of a component. Moreover, according to Szypersky, all context-dependencies should be specified in the interface.
Connection secrets are concerned with the connection and communication of components. They should be hidden in connectors.

b) What aspects of transparency do you know? How are they aligned with the secret subjects?

Solution:

- *Content secrets*
  - **Language transparency**: Provide interoperability of components using different programming languages.
  - **Persistency transparency**: Hide whether component has persistent memory.
  - **Lifetime transparency**: Hide whether component has to be started first, or was already running.
- *Connection secrets*
  - **Location transparency**: Hide the distribution of components between local and remote servers.
  - **Naming transparency**: Hide how components providing a service are named and invoked.

- **Transactional transparency**: Hide whether a component supports consistent parallel writes with transactions.

c) What is language transparency and how can it be achieved?

Solution: Interoperability of components which is independent of the concrete programming language. An Example would be SOAP-Web Services (they use a standardized XML-based exchange protocol for communication).

d) Why is location transparency important? Give an example.

Solution: Location Transparency is the interoperability of components independent of the concrete location of the component (device running the component). When a component changes its location, the implementation of dependent components should not change. Many modern systems are distributed component-based systems. Especially, in case of applications for Internet-of-Things (IoT), a multitude of heterogeneous, distributed devices should be integrated dynamically to form a context-dependent ad-hoc system of systems. In those systems, location transparency is one of the most important features.

Task 2 Enterprise JavaBeans (EJB) (T)

Enterprise JavaBeans (`EJB`)[1] is a Java-based composition system for designing and executing modularized, component-based systems.

a) Is `EJB` a composition system? Describe the component model, composition technique and composition language.

Solution: Yes, it encompasses many aspects required for composition systems.
Component Model: `EJB` uses a Java-based component model, where static components contain classes and dynamic components contain objects. It distinguishes three kinds of Beans, i.e., Session Beans, Message-Driven Beans, and Entity Beans. Required component types are specified declaratively with annotations, dynamically requested via JNDI or directly requested via the `Home Interface`.
Composition Technique: `EJB` provides multiple mechanisms for connecting components, i.e., JNDI for name lookup, interceptors (server-side skeletons) for adaptation, Glue code generated by EJB Container. Moreover, `EJB` supports aspect separation by specifying/changing middleware services in the deployment descriptor or with Interceptors. Furthermore, `EJB` benefits from the metamodeling capabilities of Java, e.g., reflection and annotations.
Composition Language: While in `EJB 2.0` the **deployment descriptor** serves as the composition language, `EJB 3.x` uses the declared dependencies of beans to determine the composition.

b) Compare `EJB` components to the definition of components by Szyperski et al. [1].

Solution:

- Unit of composition: **class**
- Specified interfaces: **home interface, remote interface** and **declared dependencies**
- Explicit context dependencies: Environment and requirements are specified via *annotations* per Bean or in the *deployment descriptor*.
- Independently deployed: Different kinds of Beans have different **life cycles** and can be independently deployed on third party application servers (e.g., *JBoss* or *GlassFish*) via a `BeanContainers`.
- Third-party composition: `EJB` is a *de-facto* **standard** for Java-based enterprise applications, as it is part of the Java Enterprise Edition (JavaEE).

[1] https://www.oracle.com/technetwork/java/javaee/ejb/index.html

c) Which transparency problems does `EJB` address? Which transparency problems are not addressed?

Solution:

- *Content secrets*
  - **Language transparency**: Not supportet, as `EJB` requires a `JDK` and `JRE` to generate stubs and run Beans, respectively.
  - **Persistency transparency**: Yes, as persistency is hidden behind the definition of Entity Beans, or handled via JPA annotations (cf. EJB 3.0).
  - **Lifetime transparency**: Yes, as deployment is handled by the application service and the life-cycle is hide inside the Bean container.
- *Connection secrets*
  - **Location transparency**: Not supported, because developer must distinguishe between calling a local or a remote interface.
  - **Naming transparency**: Yes, this can be achieved with the JNDI naming service.
  - **Transactional transparency**: Yes, transactions between Beans are hidden and declared with annotations.

Task 3 Implementation of the Factory Automation Application – Part 1 (P) (Part I till Part III - Week I till Week III)

Up to 3 Point for DSE(2020) (Differentiation below)

In the last exercise you designed a simple management application for factory automation for a 3D-printing service. In this task you will start to implement parts of your design in EJB *(Version 3)*. EJB is a mature and powerful composition system. We will use EJB to implement parts of the factory automation use case, described in exercise 2. To get familiar with EJB, you will install the required tools and work yourself through the listed tutorials. In this first part you are going to implement 3 components, such as the customer-, product-, and order-management. All components offer interfaces to add, remove and list customers, products and orders. You do not have to implement a front-end for your components. However, you must test their individual functionality.

a) Setup your development environment following the tutorial for *WildFly 10.1*.[2] That includes:

- Download and install *Eclipse IDE for Enterprise Java Developers*.[3]
- Download a *Java 8 JDK*.[4]
- Download the *WildFly 10.1.0.Final* application server.[5]

b) Read and reconstruct the tutorials for creating a simple *EJB 3* application. Please use the WildFly server from *a)* and a *Java 8 project* for the tutorials.[6] [7]

c) Now create a new EJB project for your *Factory Automation Application*.

- Implement the customer manager component.
  1 Point for DSE(2020) (Part I - Week I)
- Implement the stock manager component.
  1 Point for DSE(2020) (Part II - Week II)
- Implement the order manager component.
  1 Point for DSE(2020) (Part III - Week III)

[2] http://www.ejbtutorial.com/j2ee/getting-started-with-j2ee-installing-wildfly-on-eclipse
[3] https://www.eclipse.org/downloads/packages/release/2020-03/r/eclipse-ide-enterprise-java-developers-includes-incubating-components
[4] For example: AdoptOpenJDK
[5] https://wildfly.org/downloads/
[6] http://www.ejbtutorial.com/ejb/hello-word-tutorial-for-writing-stateless-session-enterprise-java-bean-ejb
[7] http://www.ejbtutorial.com/j2ee/tutorial-how-to-invoke-an-ejb-from-java-application-client-using-wildfly-and-eclipse

Hint 1: In a production scenario, you will need to set up *Entity Beans* for persistently saving Data. That will need some additional configuration[8] [9]. In our case, it also would be sufficient to use a *Stateful Bean* with a simple Java data structure.
Hint 2: Please note that the work on the first week's task will be more extensive then the work on parts for weeks 2 and 3, since you need to set up your system and learn EJB.
Hint 3: You can use your solution of *Exercise 2* as the base for the design of your EJB project or you can also use the example solution of *Exercise 2*. Please take notice of the requirements in the task description, too.
Hint 4: The most important part of this exercise is how you design your interfaces. An also important point is how to export and import data through them.

d) Test your components. This can, for example, be done by creating a small client application with tests methods.

Solution: A possible solution can be downloaded from OPAL.

References

[1] Clemens Szyperski, Jan Bosch, and Wolfgang Weck. Component-oriented programming. In *European Conference on Object-Oriented Programming*, pages 184–192. Springer, 1999.

[8] **Tutorial Entity Bean:** https://www.lalilуna.de/articles/posts/ejb-3-tutorial-jboss.html
[9] **Entity Bean:** You can also use the the example H2 Datasource
`<jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>`
in the *persistence.xml*.