TECHNISCHE UNIVERSITÄT DRESDEN

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

SS2019 – Component-based Software Engineering

Web Services and Service-Oriented Architectures

Professor: Prof. Dr. Uwe Aßmann
Tutor: Dr.-Ing. Thomas Kühn

Task 1 Web Services Architectures

Web service architectures denote the first step to *service-oriented architectures* (SOA). They encompass services, which are offered, discovered, downloaded, and executed; whereas their composition is described by a *workflow* combining the specification of both the control flow and data flow.

a) Classify *Web Services* as a composition system? Describe the component model, composition technique and composition language.

Solution: Yes, it encompasses all aspects required for composition systems.
Component Model: Components are represented as *Services* with *Ports*, whose interface and dependencies are explicitly declared by means of *Web Service Description Language* (WSLD). The actual platform specific component model is hidden.
Composition Technique: Services can be externally composed, whereas the connections between services can be flexibly changed by means of WSDL bindings. Moreover, *Web Services* provide automatic transactions and recovery.
Composition Language: Web Services introduces *workflow languages*, e.g., *BPMN* or *BPEL*, to describe both the architectural and functional composition of services.

b) Compare *Services* to the definition of components by Szyperski et al. [2].

Solution:

- Unit of composition: *Web Services*
- Specified interfaces: *Web Service Description Language* (WSDL) completely define the *types*, *messages*, *operations*, *port types*, *bindings*, and *service*.
- Explicit context dependencies: The actual implementation dependent properties of a service are hidden. Consequently, only the WSDL declares the context-dependence of each *Web Service*.
- Independently deployed: *Web Services* are addressed by their URI and their corresponding WSDL file. Through service brokers and load balancers *Web Services* can be transparently replaced and replicated, respectively.
- Third-party composition: *Workflow engines* enable third-party composition, as they can execute *BPMN* and *BPEL* workflows, and permit composing/-connecting independent third-party *Web Services*-

c) Which transparency problems do *Web Services* address? Which transparency problems are not addressed?

Solution:

- *Content secrets*
  - **Language transparency**: *Yes*, *Web Service* declaration is done in XML, independent of the actual implementation language and/or platform.
  - **Component model transparency**: *Web Service Architectures*, additionally, hide the underlying component model, such that EJB, COM+ or CORBA components can be transparently deployed and used as *Web Services*.
  - **Persistency transparency**: *Yes*, *Web Services* hide whether the state is persisted or not.
  - **Lifetime transparency**: *Yes*, *Web Service* life cycle and replication is completely hidden.
- *Connection secrets*
  - **Location transparency**: *Yes*, *Web Services* are identified by URI and WSDL file.
  - **Naming transparency**: *Yes*, if a *service broker* is used.
  - **Transactional transparency**: *Yes*, transactions can be specified in BPMN or BPEL, regardless of the service used. They are supported by most *business process engines*.
  - **Communication transparency**: *Yes*, WSDL bindings can be defined and changed flexibly.

Task 2 Web Services from Enterprise Java Beans

In the last exercise you implemented a management application for automating the factory for a 3D-printing service in `EJB` *(Version 3)*. In this task you will transform parts of the applications, i.e., the `CustomerManagement` and the `OrderManager` into *Web Services* by deploying the corresponding *web service description language* (WSDL) and generating a corresponding service proxy.

a) Add the `@WebService` annotation to both your `CustomerManagementBean` and `OrderManagerBean`. This will task `EJB` to generate a WSDL service description upon deployment (*Run* > *Run on Server*).
b) Inspect the generated WSDL files via the deployment link in your browser, e.g., `http://localhost:8080/CustomerManagement/CustomerManagementBean?wsdl` (in case JBOSS is running on `localhost` and is configured to port `8080`).
c) Finally, generate a *web service client* for both the `CustomerManagementBean` and `OrderManagerBean`. To do this use the *New* > *Project...* menu entry, and select *Web Service Client*. In the project wizard simply add the links to the WSDL files (see above) and hit next. On the next page, provide a package name, e.g., `cbse.example.service`, for the classes to be generated. Hit *Finish* to complete the creation.
d) Test the generated client by opening the package `cbse.example.service.client`, modify and run the `SampleClient` as Java application.

Solution: A possible solution can be downloaded from the website.

Task 3 Business Process Model and Notation

The *Business Process Model and Notation* (BPMN)[1] [1] is a standardized graphical notation for workflows. It is comprised of *activities* which are linked to other activities via *gateways*, representing conditions, and *connections*. Moreover, activities can be triggered and/or trigger *events*. Additionally, they provide *swim lanes* to model the collaboration of multiple distinct actors, yet, these will not be used during this task.

In this task, you should use BPMN to automate the registration, approval, and ordering in the *Factory Automation* example. In detail, after picking a *user name* and the *order details*, the process should try to register the user, wait for an approval, and afterwards place a given order. Henceforth, you will be tasked to model and implement this process via BPMN and web service calls. To run and deploy the BPMN workflow we will utilize the *Camunda Business Process Managment*[2] engine.

a) Use *Camunda Modeler*[3] to model the business process for the *Factory Automation*. Alternatively, you can use the web app *BPMN.io*.[4]

b) Now use the *Camunda BPM*[5] to deploy and execute your BPMN.

- Deploy the *Camunda BPM* using Docker with the following commands:

```
docker pull camunda/camunda-bpm-platform:latest
docker run -d --name camunda -p 9090:8080
    camunda/camunda-bpm-platform:latest
```

  After the container is running you can access the BPM via `http://localhost:9090/camunda/app/welcome/default/` using the default user and password `demo` and `demo`, respectively.

- Deploy your BPMN, where the service calls are replaced by dummy operations and the selection of the user name is implemented as a *human task*. To execute the BPMN via *Camunda BPM*, you should use the *Camunda Modeler* application, as it permits deploying the modeled BPMN.[6]
  Note, the deployment configuration must be changed to query port `9090`.

- Run your BPMN via the *Camunda Cockpit*. Make sure to assigning the *human task* to a user, e.g., `Demo`.

[1] https://www.omg.org/spec/BPMN/2.0/About-BPMN
[2] https://docs.camunda.org
[3] https://docs.camunda.org/manual/7.11/modeler/camunda-modeler
[4] https://demo.bpmn.io
[5] https://docs.camunda.org/manual/7.11/modeler/camunda-modeler
[6] https://docs.camunda.org/get-started/quick-start/deploy

c) Optionally, now change your workflow replacing all dummy services with, *service calls* to the services `CustomerManagement` and `OrderManager` generated in Task 2.

- Read the tutorial for invoking EJBs from BPMN.[7]
- Import the WSDL definition for the `CustomerManagement` into your BPMN by providing the WSDL link.
- Create a *service call* invoking the previously created `CustomerManagement` web service.
- Import the WSDL definition for the `OrderManager` into your BPMN by providing the WSDL link.
- Finally, create a *service call* invoking the previously created `OrderManager` web service.

Note: This task should be completed by each student individually, yet reusing the EJB application created previously. The created BPMN should be handed in before the next exercise.

Solution: A possible solution can be downloaded from the website.

References

[1] Thomas Allweyer. *BPMN 2.0: introduction to the standard for business process modeling*. BoD–Books on Demand, 2016.

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

[7] https://docs.camunda.org/get-started/javaee7/service-task