HTWG Konstanz
Department of Computer Science
Prof. Dr. Jürgen Wäsch

E-Business Technologies – Lab – Summer Term 2018

Assignment 4:
Systems Architecture (FMC Modeling), Web Service Design (WSDL), and REST-based Service Implementation

***** 20 Points *****

1 Overall Scenario

In assignment 3, you implemented an E-Business system for wholesalers with product management functionality and BMEcat-based product catalog import/export functionality via Web-UI.

In this assignment, your E-Business system should be—*conceptually*—extended by the following ***web services***:

- BMEcat-based *product catalog import web service* (offered to your suppliers),
- BMEcat-based *product catalog export web service* (offered to your business customers), and
- openTRANS[1]-based *order processing web service* (offered to your business customers).

Furthermore,

- your E-Business system should be able to use (consume) an *update product web service* offered by your suppliers.

As an alternative to the web service API above, your E-Business system should offer the following ***REST-based services***:

- BMEcat-based *product catalog import service* (offered to your suppliers) and
- BMEcat-based *product catalog export service* (offered to your business customers).

Only the REST-based services need to be implemented later on (for details, see Section 2.3).

[1] `www.opentrans.de/`

2 Task Assignments

2.1 Systems Architecture / FMC Modeling (*** No Assessment ***)

Describe your systems architecture with *two* FMC block diagrams:

1. Overall systems architecture for your extended E-Business system, focussing on the *service extensions described in Section 1*.
2. Detailed systems architecture, focussing on the *product catalog import/export functionality via Web-UI* as implemented in assignment 3.

Your block diagrams have to contain all information necessary to explain your E-Business system to a potential customer having a technical IT-background.

Hints

- Your block diagrams have to be complete and correct with respect to the requirements stated above.
- Choose an appropriate level of details: your block diagrams should not be more complex than necessary.
- Choose a readable design / layout for your block diagrams.
- You *can* use one of the FMC stencils, e.g., for MS Visio, announced on the FMC homepage[2].

2.2 Web Service Design / WSDL (*** 8 Points ***)

Design the *update product web service* (cf. Section 1), i.e., **develop its WSDL description**.

This web service takes a list of products as input and returns a (sub-)list of products that have been modified at the supplier's site as well as a (sub-)list of products that have been deleted at the supplier's site.

Products are described by their supplier order number, their short description, their long description and, optionally, their prices.

The WSDL description for the update product web service has to be designed as follows:

- Target namespace should be `http://dbis.in.htwg-konstanz.de/EBUT/WS/UP/<NumberOfTandem>` .

Abstract part (interfaces/portTypes, messages, and types):

- The web service offers only "interface" (`portType` in WSDL 1.1 / `interface` in WSDL 2.0), named `updateCatalog_IF`.
- This "interface" contains only one operation, named `updateCatalog`.

[2] `www.fmc-modeling.org/fmc_stencils/`

- The `updateCatalog` operation gets as *input message* an `updateCatalog_Request` message.
- The `updateCatalog_Request` input message should contain only one part, named `request`; this part should be specified using an *element* `updateRequest` (type: complexType).
- `updateRequest` consists of `authentication` (type: complexType, minOccurs='1', maxOccurs='1') *and* `listOfProducts` (type: complexType, minOccurs='1', maxOccurs='1').
- `authentication` consists of `wholesalerName` (type: string, minOccurs='1', maxOccurs='1'), `wsUsername` (type: string, minOccurs='1', maxOccurs='1') and `wsPassword` (type: string, minOccurs='1', maxOccurs='1').
- `listOfProducts` consists of `supplierProduct` (type: complexType, minOccurs='1', maxOccurs='unbounded').
- `supplierProduct` consists of `supplierAID`, i.e., the supplier order number (type: string, minOccurs='1', maxOccurs='1'), `shortDescription` (type: string, minOccurs='1', maxOccurs='1'), `longDescription` (type: string, minOccurs='0', maxOccurs='1') and `price` (type: complexType, minOccurs='0', maxOccurs='unbounded').
- `price` consists of `amount` (type: decimal, minOccurs='1', maxOccurs='1'), `currency` (type: string, minOccurs='1', maxOccurs='1'), `tax` (type: decimal, minOccurs='1', maxOccurs='1'), `pricetype` (type: string, minOccurs='1', maxOccurs='1'), `countryISOCode` (type: string, minOccurs='1', maxOccurs='1') and `lowerbound` (type: integer, minOccurs='1', maxOccurs='1').
- A *fault message* with name `updateCatalog_Fault` indicates web service failures, e.g., authentication failures.
- The `updateCatalog_Fault` fault message should contain only one part with name `fault`; this part should be specified using an *element* `updateFault` (type: complexType).
- `updateFault` contains *either* an element `authenticationFault` (type: string) *or* an element `processingFault` (type: string): `authenticationFault` should be used to indicate and describe authentication failures; `processingFault` should be used to indicate and describe other failures occuring during the processing of the web service's business logic at the supplier's site.
- The `updateCatalog` operation returns as *output message* an `updateCatalog_Response` message.
- The `updateCatalog_Response` output message should contain only one part, named `response`; this part should be specified using an *element* `updateResponse` (type: complexType).
- `updateResponse` consists of `updateDate` (type: date, minOccurs='1', maxOccurs='1'), `listOfModifiedProducts` (type: complexType, minOccurs='1', maxOccurs='1') and `listOfUnavailableProducts` (type: complexType, minOccurs='1', maxOccurs='1').

- `listOfModifiedProducts` contains the products of the supplier that are different to the products in the update request; it consists of `supplierProduct` (type: complexType, minOccurs='0', maxOccurs='unbounded').

  For the sake of simplicity, only the short description and the long description should be used to determine if two products (i.e., their descriptions) are different—differences in prices can be neglected.

- `listOfUnavailableProducts` contains the products which are no longer available at the supplier; it consists of `supplierProduct` (type: complexType, minOccurs='0', maxOccurs='unbounded').

Concrete part (bindings and services):

- A single binding, named `updateCatalog_SOAP`, should be specified as follows:
  - The transport protocol should be SOAP over HTTP.
  - The binding style should be document/literal.
  - Only one operation is offered, named `updateCatalog` (cf. abstract part).
  - The SOAP action for this operation should be `<targetNamespace>/updateCatalog` .
- A single service, named `UpdateProduct_WebService`, should be specified as follows:
  - The service offers a single port, named `UpdateProduct_SOAP`, using the binding declaration decribed above.
  - The service endpoint (URL) of this port should be specified as `http://ServiceProvider/ServiceLocation` .

    Note: later on, you need to set the service endpoint to the actual URL of your update product web service, at deployment of the web service within your E-Business system!

2.2.1 Hints

- You can write the WSDL file(s) using Altova XMLSpy, Eclipse IDE, or any other XML- or WSDL-Editor.
- Please use Altova XML Spy to check / validate your WSDL file(s)!

2.2.2 Acceptance

Please send an **E-Mail** including the WSDL-file(s) as a single ZIP-file to the lecturer **until June 22 – 13:00h**.

The subject of the E-Mail must be `EBUT-Lab-Part4-Tandem-<NumberOfTandem>`; the body of the E-Mail must include your names, matriculation numbers, the number of your tandem, and the name of the main WSDL file (if your WSDL description of the web service consists of multiple files).

The attached ZIP-file must be named `EBUT-Lab-Part4-Tandem-<NumberOfTandem>.zip`.

2.3 REST-based Service Implementation (*** 12 Points ***)

Extend your E-Business system (resulting from assignment 3) by **implement**ing the following ***REST-based services*** (cf. Section 1):

- BMEcat-based *product catalog import service* (offered to your suppliers) and
- BMEcat-based *product catalog export service* (offered to your business customers).

The *product catalog import service* should behave like the GUI-based import functionality implemented in assignment 3:

- It should check (1) if the product catalog is a (well-formed / valid) XML file in simple BMEcat format and (2) if the supplier is already in registered in the system.
- If so, the product information should be imported into the system, applying the same update semantics as in assignment 3.

The *product catalog export service* should behave like the GUI-based export functionality implemented in assignment 3:

- Coverage of articles:
  - Export of the whole product catalog (i.e., all articles in your database).
  - Export of only those articles where the short description matches a given search string, i.e., case-insensitive substring matching (not exact matching).
- Format (should be choosen based on the MIME types accepted by the client):
  - Simple BMEcat XML format (MIME type `application/xml`).
  - XHTML (MIME type `application/xhtml+xml`).

Additionally—as proof of concept—implement a simple ***REST-based client*** that uses the above REST-based services

- to upload a product catalog from the file system to your wholesaler E-Business system (i.e., simulating a supplier's E-Business system) and
- to download a product catalog from your wholesaler E-Business system to the file system (i.e., simulating a customer's E-Business system).

2.3.1 Implementation Requirements

- Use the *Java API for RESTful Web Services* (JAX-RS).
- Use the reference implementation of JAX-RS, i.e., *Jersey*.

2.3.2 Hints

- The Wholesaler Web Demo Application (cf. Moodle) contains already a very simple example how to implement REST-based services based on JAX-RS / Jersey.
- There is also a Java project available on Moodle that contains a very simple example how to implement a JAX-RS client.
- A short introductory tutorial to JAX-RS, Jersey and the examples provided is available on Moodle.
- For more information, please refer to the JAX-RS JavaDoc and the Jersey manual.

2.3.3 Assessment

The deadline for the practical **demonstration of your solution** to the lecturer in the EBUT lab is **July 3**!

Earlier demonstration of your solution is appreciated.