Institut für Informationssysteme
Abteilung für Verteilte Systeme
VU Advanced Internet Computing (184.269)

Assignment 1
Submission Deadline: Fri, 12.11.2010, 18:00 CET

General Remarks

- **Group Size:** This lab is a group lab. Group size is either 3 (preferred), or in special circumstances, 2 students. Please use the TISS forum to form groups of 3. Students *are not* encouraged to work alone, and your assignment will not be graded differently if you do (that is, we are not going to reduce the scale of the assignment if you decide to work alone, or in groups of 2).
- **Plagiarism:** Please do not copy lab solutions of other groups, or any full solution publicly available on the Internet. If we find you cheating on your assignment you will get 0 points for this assignment. It is of course allowed to discuss problems and possible solutions with your colleagues (within or outside of your lab group), but the overall assignment of your group has to be unique. It is not allowed to submit the same (or a very similar) solution for two groups, even if the groups worked together on it.
- **Other libraries:** You may use other external libraries, such as Log4J, JUnit or HSQLDB (if you want to use an in-memory database as data backend for the assignment). However, if you do, please make sure that you include them in your final assignment (e.g., in the lib folder of your solution).
- **Assignment Submission:** Once you have completed your solution please upload it to the DSG Teaching Tool[1]. Submit a ZIP-compressed file containing all your source code and the required build file, but without the Apache CXF libraries. Make sure that your code compiles and is runnable using Java 6 and Apache CXF 2.2.3.
- **Deadline Extensions:** Deadline extensions are only given on special circumstances, and upon individual agreement with the course administration. Obviously, starting too late does not entitle you to a deadline extension. However, if you are unable to finish your assignment duly, try to submit what you have before the deadline. Submitting an unfinished assignment is still a lot better than submitting nothing at all. Keep in mind that there is always the possibility that something goes wrong during submission, and plan accordingly – try to upload your submission earlier than 15 minutes before the deadline. If you have any problems with the DSG submission tool that you cannot solve in time you should send a hash value (e.g., MD5) of your submission to the course administration before the end of the submission deadline. Using this hash we can verify later on that you did not change your submission after the deadline.
- **Distribution of work within the group:** Every student needs to participate in every assignment. We will check that every student at least fully understands the solution of this assignment during the lab interview, and reserve the right to give single students within a group fewer points if we feel that he/she did not participate appropriately.

[1] `https://stockholm.vitalab.tuwien.ac.at/dsg-teaching-web/student/studentLogin.htm`

Introduction Welcome to the Advanced Internet Computing lab in WS 2010/2011. In the first assignment, we will use the Apache CXF toolkit to set up a basic Web service infrastructure that we can use as foundation for the second assignment. In the first assignment, we will specify in detail what you have to do; this is so that every group has a similar set of services to work with in Assignment 2. Please note that even if the toolkit we use hides many complexities of Web services from the programmer, you are still expected to have a good understanding of what goes on in the background, so read through the literature pointers we provide.

Grading Assignments are graded during the lab interview. We will live-test your assignments there, and ask some *basic* theoretical questions. *Important:* we expect that every student participates at (and, therefore, fully understands) each assignment. We reserve the right to subtract up to 50% of the total assignment points of single students if we get the strong impression that (s)he did not fully participate and tries to "swim along". You may reach 24 points in this assignment. We have indicated the total points dedicated to each part of the assignment as part of the header (e.g., Data Model (4 points)) in the assignment description. However, please make sure that your solution is compilable and that your test client executes without exception. You can receive max. 33% of the possible total points for parts which are not executable or testable (even if you have implemented them correctly!). This also means that you will receive only 33% of the points for functionality for which no test exists, so write the test client (see description below) with care.

Lab Background For the lab, we will use the following simple scenario: consider you are a software architect of a big online book seller. You are responsible for setting up an infrastructure for order processing. Since you aim for the highest degree of automation you decided to base your IT infrastructure on standards-based and loosely coupled components, i.e., Web services. Your business process is as depicted in Figure 1. All activities except for *Wait for Items Delivered* need to be covered with Web services.

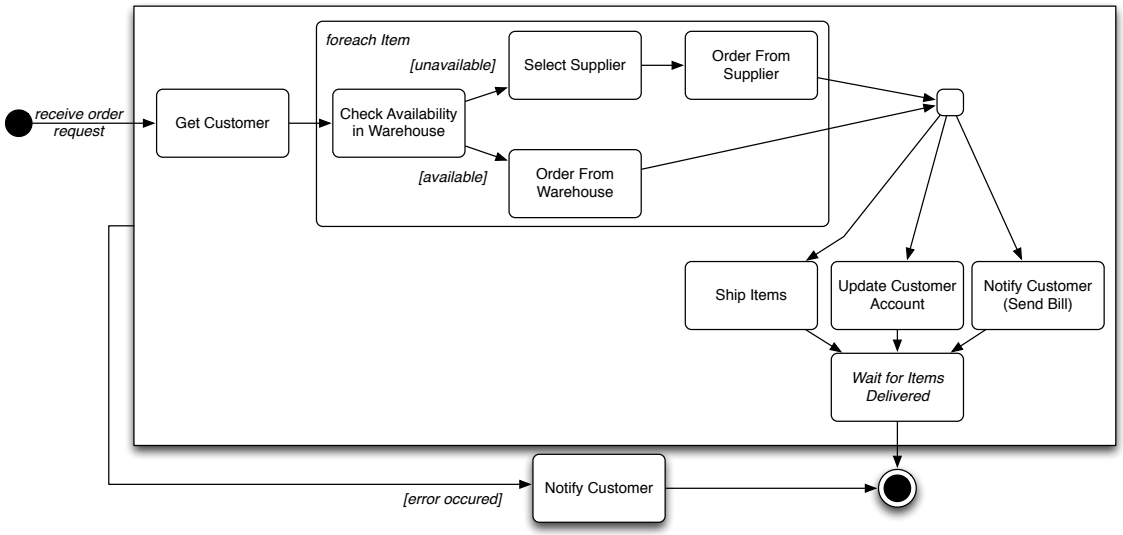

Figure 1: Book Ordering Process

Prerequisites Before starting with the assignment, you should ensure the following:

- *Before* you begin coding, you should have a decent understanding of the basic concepts of Web services. Review the lecture slides and have a look at the literature pointers we provide in the Literature section below if you are unsure.

- As a first step in the assignment, go to the Apache CXF web page and download the 2.2.3 version of Apache CXF[2]. *In your own best interest, please use version 2.2.3 and not the recent one. You will have problems in Assignment 2 if you use the recent version of CXF!* Additionally, you'll need an up-to-date Java JDK (Java 6), and a Java IDE may also come in handy (NetBeans 6.5.x or 6.7.x may be a good choice, since you need this tool for Assignment 2 anyway.

Literature For the assignment, it is important to have a solid understanding of Web service principles. Please use the literature pointers below to inform yourself if you feel that you do not understand some of the basic WS concepts:

- First and foremost, have a look at the lecture slides. Use the other material to detail the topics presented during the lecture.
- Secondly, use the relevant standards and specifications (SOAP, WSDL, WS-I Basic Profile, JAXB, JAX-WS, JAX-RS) as reference material for any detailed questions that you have. Note that you do *not* need to read all these standards cover to cover, but you might want to keep them close for reference.
- Finally, for some specific topics you should look at these internet resources: the following link describes the different encoding variants for Web services[3]; this link describes the JSON data transmission format[4]; here is an article that (very briefly) summarizes the main ideas of REST[5]; and here you can find some details on the WS-I Basic Profile[6].

Please keep in mind that we will check that you actually understand these concepts by asking some background questions during the lab interview. Don't worry, these questions are not going to be in-depth, but you should be able to tell the difference between e.g., RPC/encoded and document/literal, or describe the fundamental elements in a WSDL description.

Assignment Description (24 Points)

Data Model (4 Points) As a first step in this assignment, you need to create the underlying data model for the rest of the lab. The simple data model of your domain is sketched in Figure 2. Your first task is now to implement this data model in Java using standard Java Beans – that means, you need to provide getter and setter methods for each field of every class, and every class needs to have a default (no-arg) constructor. It is also a good practice to override the `equals()` and `hashcode()` methods of the beans sensibly.

Furthermore, you need to add appropriate JAXB annotations (see for instance here[7]) to all classes and fields, to make sure that your data beans are easily serializable to XML (this will be important later). Consider the following rules when developing your JAXB mapping:

- Assign a meaningful XML root tag to all classes.
- Serialize all fields except the IDs to XML elements.
- Serialize IDs to XML attributes.
- Use a custom Java type adapter to map the `Date` value in the order class to "long" (in XML, your dates should be represented as milliseconds since 1970-01-01).

[2] `http://cxf.apache.org/`
[3] `http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/`
[4] `http://www.json.org/`
[5] `http://www.xfront.com/REST-Web-Services.html`
[6] `http://www.ibm.com/developerworks/webservices/library/ws-basicprof.html`
[7] `http://www.caucho.com/resin/doc/jaxb-annotations.xtp`

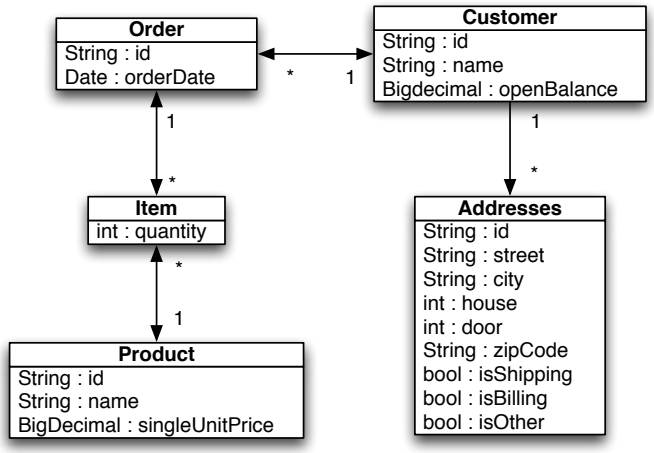

Figure 2: Order Processing Domain Model

Do not use explicit namespace declarations (i.e., leave the default values). Furthermore, we need a data backend where we can store our data objects; set up simple lists, which we can later on use to store and exchange objects between services. You do not need to use a full-fledged relational database, a few in-memory data structures should be sufficient (however, make sure that your data backend is thread-safe). Insert some demo test data into your data backend.

Basic SOAP Web Services (5 Points) Now it's time to set up some JAX-WS Web services using Apache CXF, which use the data model that we have developed earlier. First we start off slowly by implementing our most simply Web service, the `Shipping Service`. This service has only one Web service operation, `ship_items`, takes an array of items and an address as input. Use the JAX-WS annotations (supported by Apache CXF) to implement the service as a SOAP Web service (see a tutorial here[8] to get you started). Assign the target namespace `http://infosys.tuwien.ac.at/aic10/ass1/dto/shipping`, the port type name `ShippingPT` and the service name `ShippingService`, and use explicit annotations to define all relevant Web service details, such as operation name, parameter names, or optionality. In the implementation of the services you can simply log a message to `System.out` (see example below). Generate a random shipping identifier (an `java.util.UUID`) and return this ID as simple String to the service invoker. If the address or one of the products do not exist, you should throw an `UnknownAddressFault` or UnknownProductFault SOAP fault.

```
1 [ShippingService] Thu Aug 28 16:26:32 CEST 2010
2 [ShippingService] Sending items 'War and Peace', 'Moby Dick', 'David Copperfield' to
3 [ShippingService] Argentinierstrasse 8, 1040 Wien
```

Of course we need some more services to implement the business process depicted in Figure 1. The next task is not more complicated than the `Shipping Service`. Implement two (identical) `SupplierServices`, we call them `Supplier1` and `Supplier2`. These suppliers have also just one operation, `order`. This operation takes a product and and amount (as integer) as input, and returns the total price. Throw an UnknownProductFault SOAP fault if the product is not known by the supplier. Assign service name, namespace, port type, etc. as above (of course adapting the names). However, it will be important later on that both suppliers are assigned the same name and namespace. Print some debug output as above.

[8] `http://cwiki.apache.org/CXF20DOC/a-simple-jax-ws-service.html`

In addition to those identical suppliers, we also need a third service which is basically a special form of supplier. We call this service the `WarehouseService`. The `WarehouseService` has an additional method, `check_availability`, with the same parameters as the `order` operation. `check_availability` returns a boolean value indicating if the requested amount of product is available in the warehouse, plus an integer value indicating the estimated delivery time. When implementing this service you should hard-code a table of products, their availability, and their delivery times.

The general idea of these Web services is that for every ordered product the process first checks the warehouse, and if available there, gets the item from the warehouse. If an item is not available in the warehouse it is ordered from one of the two suppliers. Every supplier is responsible for certain products. Of course we still need a component that is able to decide which supplier should be used for which product. This is the task of the `SupplierRegistry`.

A Simple Service Registry (3 Points) As you will learn in the lecture service registries are considered to be a central element in service-oriented computing. They decouple service clients from providers, and can be used as a broker in order to find out about suitable service instances for a given task. Originally, UDDI was meant to be the standard service registry in SOA scenarios, however, due to its low prevalence in real-life Web service environments we will not use UDDI in this lab. Instead you should implement your own specific registry service.

The idea of the registry is to return the WS-Addressing[9] endpoint reference of one of the two supplier services, based on the product passed as parameter. We have attached the XML Schema of WS-Addressing to this assignment text[10]. Obviously, you need to use a different approach when implementing this service. You may want to check the CXF documentation about WSDL-First development[11] for this task. In your implementation, your registry should keep a static mapping of products and providers, e.g., in a hashtable to decide which endpoint to return. As the suppliers, the supplier registry should throw a `UnknownProductFault` if an unknown product is passed as parameter.

RESTFul Web Service (5 Points) After setting up our JAX-WS service infrastructure, it is now time to get our hands on the second recent WS-related Java specification: JAX-RS, the Java API for RESTful Web Services. Luckily, JAX-RS is also implemented in Apache CXF, so that we can continue with the same tooling that we used so far. To get started, you should have a look at the CXF JAX-RS documentation[12]. What we are going to do now is implement a JAX-RS based RESTful service that we can use to manage our customer data (the `CustomerManagementService`). This service should implement standard CRUD (create-retrieve-update-delete) functionality for customers. Additionally, implement an operation `notify` with two parameters (customer and message, as string) and an operation `update_account`, again with two parameters (customer, changed value as `BigDecimal`). `update_account` just takes the customer and adds the `changedValue` parameter to the customer's open balance field.

Simple parameters (e.g., the ID for the retrieve operation) should be passed as path parameters, while complex parameters (e.g., the customer object for updating) should be passed in the request body. Make sure that your RESTful service follows the design principles dictated by the REST architectural style. Make sure that you use the "right HTTP method for the job", e.g., for retrieving customers you should use HTTP GET, for adding customers you should use HTTP PUT, and so on. Think about nice ways to add the two additional operations on the customer resource.

One of the key features of JAX-RS is the transparent support of multiple data bindings – that means you can easily expose your service so that it accepts and provides various data formats. You should make use of that feature, and accept and provide data **only in the JSON format**.

[9] `http://www.w3.org/Submission/ws-addressing/`
[10] `http://www.infosys.tuwien.ac.at/teaching/courses/IntAppl/addressing.xsd`
[11] `http://cwiki.apache.org/CXF20DOC/developing-a-service.html#DevelopingaService-WSDLFirstDevelopment`
[12] http://cwiki.apache.org/CXF20DOC/jax-rs.html

As always, print some debug output analogous to the JAX-WS services to the console so that you can see what's going on.

```
1 [CustomerManagementService] Thu Aug 28 14:44:13 CEST 2010
2         Adding customer with ID 279cedd6-ba3b-4a30-a63e-8e54f28f0037 and name MiniMe
```

Unfortunately, as much fun as this REST service is, it poses certain problems for integration, since as of today no WS-BPEL engine exists which is able to cope with the rather undefined nature of REST. Therefore, we have to wrap the REST operations with a (JAX-WS) SOAP-based service, in order to use our `CustomerManagementService` in the second assignment. Wrap all operations of the `CustomerManagementService` with SOAP operations. You simply need to implement the adequate Web service operations which forward SOAP requests to the REST service, converting data between XML and JSON (basically, you should implement a SOAP adapter for your REST service, in the sense of the GoF Adapter pattern[13]). Please note that "copy and pasting" the `CustomerManagementService` business logics is not a satisfactory solution for this task. The wrapper has indeed to use the REST service to implement its functionality.

Service Starter (1 Point) Obviously, we also need a component that actually launches all these services. Fortunately, we do not need to set up a specific application server for this task – Apache CXF already comes with a bundled Jetty HTTP server, which we can use to programmatically self-host our services. Create a new Java class that has a standard main method, and publish your services there. After publishing, you can find the WSDL contract of a SOAP-based Web service by appending `?wsdl` to the endpoint address of the service, e.g., `http://localhost:8088/shippingservice?wsdl`. Examine the WSDL descriptions of your services: how does the WSDL change if you e.g., change the JAXB settings on your beans? what is the difference between wrapped and bare parameter encoding? are your services WS-I Basic Profile compliant?

Test Client (3 Points) Now we have set up all the infrastructure that we want for now. However, we did not quite test all these services so far. The last step is therefore to create one or more clients that can be used to invoke the Web service operations implemented earlier. You can either write standard executable Java applications that demonstrate the functionality of your services, or a set up JUnit tests. You should again use Apache CXF to implement these clients[14]. You can use compiled stubs for the JAX-WS services (Apache CXF includes a `wsdl2java` tool for this purpose). We will not give detailed instructions on what the tests / demonstrations should look like, however, we require that every functionality of your services is tested. That is, you need to invoke every service operation at least once regularly, and you need to trigger every SOAP fault at least once.

All these test clients need to be reasonable verbose, i.e., it should be obvious from looking at the console of both services and clients what is currently going on. Additionally, enabling CXF logging in the JAX-WS services will help you enormously (use the logging interceptors as described here[15]). The logging interceptors will print out all SOAP messages going into and out of the JAX-WS services. For the JAX-RS service, set the logging interceptors on the server factory bean, not with annotations.

[13] `http://en.wikipedia.org/wiki/Adapter_pattern`
[14] `http://cwiki.apache.org/CXF20DOC/developing-a-consumer.html`
[15] `http://cwiki.apache.org/CXF20DOC/debugging.html`

Ant Build File (2 Points) Your solution should contain an Apache Ant[16] build script with the following targets:

- **run-services**: this target should compile and start all Web services you developed. As your solution will make use of threads, it is advisable to use the attribute `fork=''true''` with the `<java>` task to prevent Ant from killing its process after the execution of the `main` method, which starts the threads, has finished.
- **run-client**: this target should perform the client test invocations. If you use JUnit, you can use the Ant `<junit>` task. Otherwise invoke a Java `main` method which contains the tests.

Please make sure that your build script contains only relative paths and runs on any machine without the need for modifications. Specifically, build files generated by Netbeans sometimes contain hard-coded absolute paths (look at `nbproject/project.properties`), which should be eliminated. Please provide a short readme file with the necessary steps to run your solution, if just copying the CXF 2.2.10 libraries into the subfolder `lib` (or the like) of your solution and starting ant with the targets mentioned above is not enough.

[16] `http://ant.apache.org/`