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

Assignment 2
Submission Deadline: Fri, 14.01.2011, 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 and cannot pass the course. 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.
- **Assignment Submission:** Once you have completed your solution please upload it to the DSG Teaching Tool[1]. Submit a ZIP-compressed file containing your NetBeans project(s) with the BPEL process and the services from Assignment 1 (yes, you should resubmit your solution from Assignment 1, since you may have changed parts of it). Make sure that your code compiles and is runnable using Java 6 and Apache CXF 2.2.10. For the BPEL part, use the most recent version of OpenESB[2].
- **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`
[2] `https://open-esb.dev.java.net/`

Introduction Congratulations! You passed the first assignment and are ready for the second step - business process design. That means that after setting up a basic Web service infrastructure with Apache CXF you are now ready to orchestrate these services to a business process using WS-BPEL. As in the first assignment, we expect that you understand what is going on "under the hood", so make sure that you read the literature pointers and understand what's going on.

Grading Assignments are graded similarly to the first assignment. The same rules apply. In the following description we will again indicate how many points you can at most receive for each task of the solution.

Prerequisites Before starting with the assignment, you should make yourself sure of the following:

- *Before* you begin designing processes, you should have a decent understanding of the basic concepts of the Web service business process execution language (WS-BPEL for short)[3]. Review the lecture slides and have a look at the literature pointers we provide in the Literature section below if you are unsure.
- For this assignment you need NetBeans[4], in particular the SOA plugins and the BPEL Designer[5]. Additionally, for deploying and executing BPEL processes, the GlassFish application server and the GlassFish Enterprise Service Bus (ESB)[6] are required.

Literature For the assignment, it is important to have a solid understanding of the WS-BPEL principles. Please use the literature pointers below to inform yourself if you feel that you do not understand some of the basic 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 standard (WS-BPEL) as reference material for any detailed questions that you have. Note that you do *not* need to read all details cover to cover, but you might want to keep them close for reference.
- For some specific topics you should look at these internet resources: the following link introduces the concepts of WS-BPEL based orchestration[7]. A good starting point for the usage (and some useful examples) of the Netbeans BPEL Designer[8].

Assignment Description (24 Points)

Basic Process (13 Points) Let us revisit the book ordering process which was our starting point for the first lab. It is depicted again in Figure 1.

By now it should be evident that we have implemented all Web services necessary to implement this process (the mapping between activities in the process and Web service operations should be straight-forward).

Your first task is now to implement a basic version of this process. Have a look at the WSDL interface that we have attached to the assignment. Your process has to implement this interface. For the moment you are not allowed to change this interface at all (you will have to slightly modify for a later part of this assignment).

[3] `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html`
[4] `http://www.netbeans.org/`
[5] `http://www.netbeans.org/features/soa/index.html`
[6] `https://open-esb.dev.java.net/`
[7] `http://www.oracle.com/technology/pub/articles/matjaz_bpel1.html`
[8] `http://www.netbeans.org/kb/trails/soa.html`

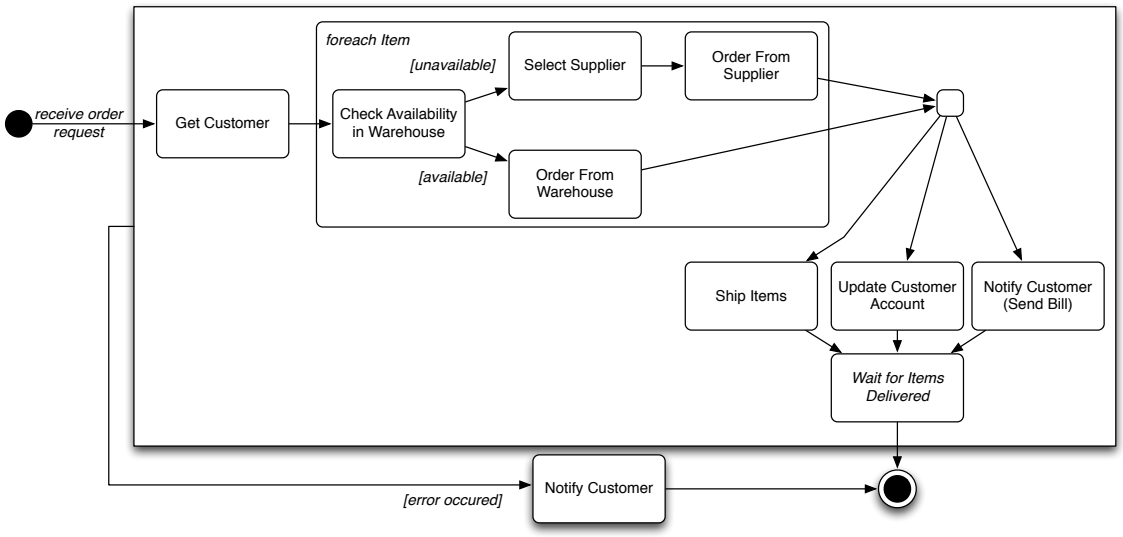

Figure 1: Book Ordering Process

New instances of the process are created when a new order is received (operation `start_process` in the WSDL). Afterwards, you retrieve the customer from the customer service, and iterate over each ordered item. For each item you check if it can be ordered from the warehouse. If this is possible you order from the warehouse, if not you order from a supplier. In the first version you can assume that all items are always ordered from supplier 2 (the second supplier service), that is, you do not need to use your registry service for now. The next step of the process is the parallel execution of the activities "Ship Items", "Update Customer Account" and "Notify Customer". Shipping and notifying the customer are straight-forward. For "Update Customer Account" your process should calculate the total value of the order and increase the customer account by this value (see the `update_account` operation). Finally, your process should block until the items are delivered to the customer, but you may ignore this step for now and just return your result directly. You do not need to handle the custom SOAP faults thrown by our Web services in this basic version of the process.

Fault Handling (2 Points) As a first extension of the basic process, you should implement fault handlers that deal with all the SOAP faults that can be thrown by the Web services we have implemented in Assignment 1. Use the `notify_customer` operation to notify the client of any problems. The message sent to the customer has to be fault-specific, that is, don't just send something generic like "Fault received", but something that actually indicates what the problem is.

Dynamic Service Endpoints (3 Points) Now it's time to put our simple registry service to good use. Add a "Select Supplier" activity to your process, which queries your registry service. Then invoke the supplier service that has been returned by the registry (not always supplier 2, as we have assumed so far). Note that this step should be implemented using exactly two Invoke activities: the first one queries the registry, the second is an Invoke activity whose partner link is set dynamically. It is not ok to just implement Invoke activities for both suppliers and select them via branching.

Some ideas of how to do that can be found here[9]. Note that your solution should be flexible in the sense that it should also work if a different URI endpoint would be returned (but you can assume that the service returned has the same abstract WSDL interface as your two supplier

[9] `http://wiki.open-esb.java.net/Wiki.jsp?page=UsingDynamicPartnerLinks`

services). If your invocations to one of the supplier services fail you should verify that both services have the same name and port type, and that all namespaces etc... are identical.

Asynchronous Process (5 points) Finally, you should implement the "Wait for Items Delivered" activity. Basically, this activity does not map directly to a Web service invocation. Instead you need to find a way of blocking your process until you get a callback from the customer that the items have been delivered. This should be implemented using the WS-BPEL support for asynchronous notifications.

Since we want to keep things simple we don't actually implement a customer for this assignment. Instead, you can simulate this notification from the customer by extending your shipping service a little. Implement some functionality in the service that, whenever the `ship` operation is invoked, waits for some time (say, 10 seconds, but the timeout needs to be long enough that we can see that the process is indeed blocked!) and then calls the `callback` operation of the process to indicate that the items have arrived.

Tip 1: take a look at the asynchronous BPEL sample of NetBeans, it uses most constructs necessary for this subtask. You will need correlation sets and non-initial receives to implement this part of the solution. Please keep in mind that we actually expect you to use asynchronous notifications. It is not ok to implement, e.g., a polling method which is periodically invoked from the process. The result needs to be asynchronous and use a callback for you to receive all points.

Tip 2: it may be necessary to extend the WSDL interface of the process slightly for this task. You are allowed to add variable properties (`vprop:property`) and property aliases (`vprop-:propertyAlias`) to the WSDL, as well as add imports of other WSDL files, but please don't change the operation parameters etc.

Submission Package and Test Cases (1 point) The outcome of this assignment is a deployable and executable WS-BPEL NetBeans project. Make sure that your project can be built and deployed using NetBeans (i.e., using the "Clean and Build", and "Deploy" functions). Additionally, we need a way to actually invoke the deployed process. You should therefore create some test cases using the "New Test Case" functionality of NetBeans, and execute the process by running the test case. Make sure that your backend services are actually invoked during your process runs (remember, they should produce some debug output when invoked!). Please make sure that the BPEL process integrates only services from `localhost`! Otherwise we cannot deploy and test the process.