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

Assignment 1
Submission Deadline: Nov. 13th, 2011, 8 PM 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. We will use the JPlag[1] software to semi-automatically check for suspicious similarities.
- **Assignment Submission:** Once you have completed your solution please upload it to the DSG Teaching Tool[2]. Submit a ZIP-compressed file containing all your source code and all required libraries. Make sure that your code compiles and is runnable using Java 6.
- **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 the 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://www.ipd.uni-karlsruhe.de/jplag/`
[2] `https://stockholm.vitalab.tuwien.ac.at/dsg-teaching-web/student/studentLogin.htm`

Introduction Welcome to the Advanced Internet Computing lab in WS 2011/2012. In this first assignment, you will be tasked with devising and implementing some enterprise services used in a simple business process. You may use any Web service toolkit to implement these services, but we suggest that you use the current version of the Apache CXF framework[3] for your work. Furthermore, you should use JAXB to map Java objects to XML. If you are using a half-way recent version of Java, JAXB should already be part of the system libraries. Please note that, even if modern toolkits hide many complexities of Web services from the programmer, you are still expected to have a good understanding of what goes on in the background.

Grading Assignments are graded during the lab interview. We will live-test your assignments there, and ask some background questions. You may reach 30 points in this assignment. Points are determined based on (1) completeness of solution, (2) ability of *all* students to answer questions during the interview, (3) code quality and quality of the solution architecture. Furthermore, if you implement the process as a Web application, you may receive up to 5 bonus points (see below).

Literature For the assignment, it is important to have a solid understanding of Web service principles. Please use the literature pointers below to find further reading and tutorials on the tools and concepts used in this lab.

- Firstly, 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.
- For implementing Web services using Apache CXF, the CXF User Guide[4] is a good starting point.
- For mapping your data transfer objects to XML, you can look at this very extensive JAXB tutorial[5] (but note that you will need much less than what is described here).
- Finally, for some specific topics you should look at these internet resources: the following link describes the different encoding variants for Web services[6]; this link describes the JSON data transmission format[7]; and here is an article that (very briefly) summarizes the main ideas of REST[8].

Lab Description For the lab, we will use the following simple scenario: consider you are software architects at a small bank, and you are tasked of bringing the ancient, COBOL-based credit approval process into the 21st century. For this, you should (1) identify appropriate services and service operations in your domain, (2) implement these services as Web service wrappers around the old legacy application, and (3) build a new frontend for bank clerks, which integrates these new service wrappers, and some external services provided by business partners. From the business analysts, you have received the diagram in Figure 1 as a high-level description of the credit approval process (based on BPMN 1.1 notation[9]). Essentially, the bank clerk places a credit request for a customer. As a first step, the system needs to retrieve the rating for both, the customer and all his warrantors (as specified by the clerk as part of the request). With this information, an offer can be generated, which can be accepted or declined by the customer. Additionally, the customer can decide to update the request (i.e., change the amount or duration of the credit), which leads to a new offer being generated (for simplicity, we exclude the case that also more,

[3] http://cxf.apache.org/
[4] http://cxf.apache.org/docs/index.html
[5] http://jaxb.java.net/tutorial/
[6] http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/
[7] http://www.json.org/
[8] http://www.xfront.com/REST-Web-Services.html
[9] http://www.omg.org/spec/BPMN/1.1/PDF/

or more trustworthy, warrantors could be added). If the offer is declined, the offer is removed from the system. If it is accepted, a written contract is printed and shipped to the customer for physical signature, who has to fax the signed contract back. In parallel to this activity, the actual disbursement process (transferring the money to the customer account) is started. Note that "in parallel" actually means in parallel, i.e., it is not ok to execute these activities one after the other.

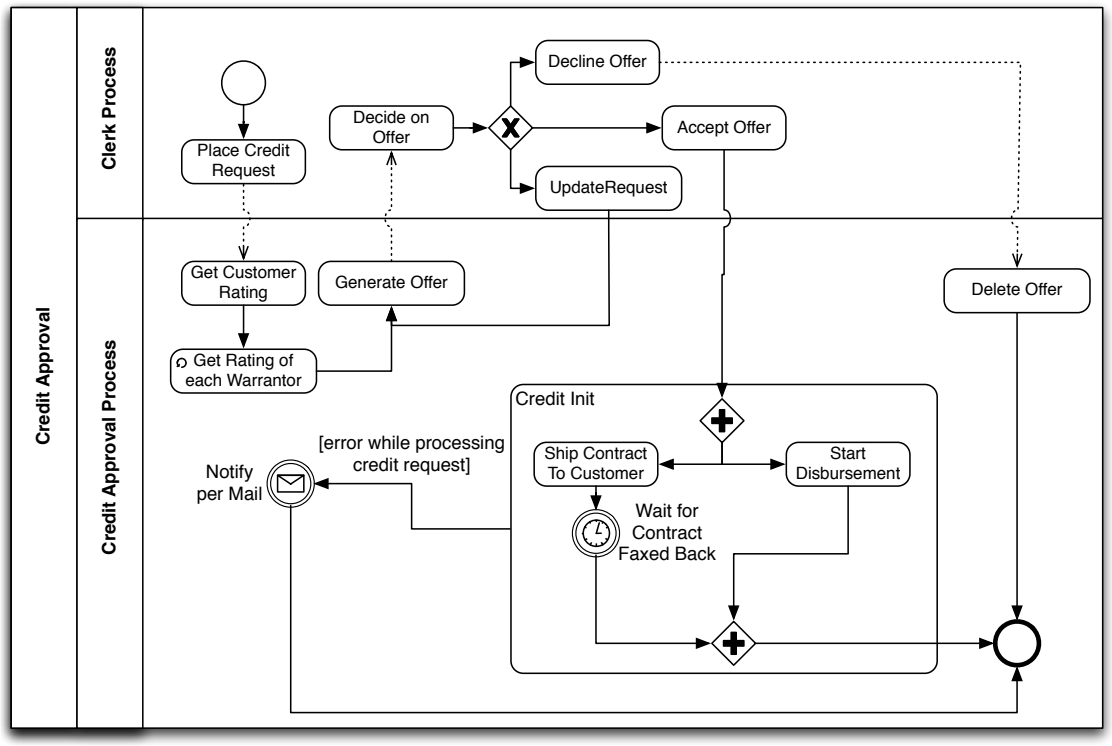

Figure 1: Credit Approval Process

The expensive SOA consultant hired by the bank for this project suggests (read: orders you) to implement four separate services as basis for this process, a **shipping service**, a **rating service**, a **contract management service** and a **customer relations management service** (we will assume that, for some reason, no services for any of these tasks exist up to now). Additionally, the consultant suggests to implement the rating service as a RESTful Web service, using JSON as a data transport format, to ease integration with Web-based tools of other departments. Make sure that your RESTful service is actually using the REST principles (resource-orientation, statelessness, correct usage of HTTP verbs, etc.). For security reasons, the contract management service should be properly secured against unauthorized access and eavesdropping using WS-Security (*note: verify that your messages are indeed encrypted, and prepare to demonstrate this during the assignment interview*).

All services should be implemented as thin layers on top of the legacy application. Your services should log invocations, do input data validation (triggering meaningful custom SOAP faults on problems, or retiring meaningful fault resources in the case of REST), and then forward the request to a *mock* of the legacy application. The mock does not need to be sophisticated, but it should behave in a way that the process is testable and demonstrable.

Another input that you have received from the business analysts is a domain model for your credit approval process (Figure 2). The analysts suggest that you base the data transfer objects used by your services on this domain model.

Disbursements are handled by a different department, located in a different country. You will have to make use of the disbursement services that these guys provide, and you have no access to

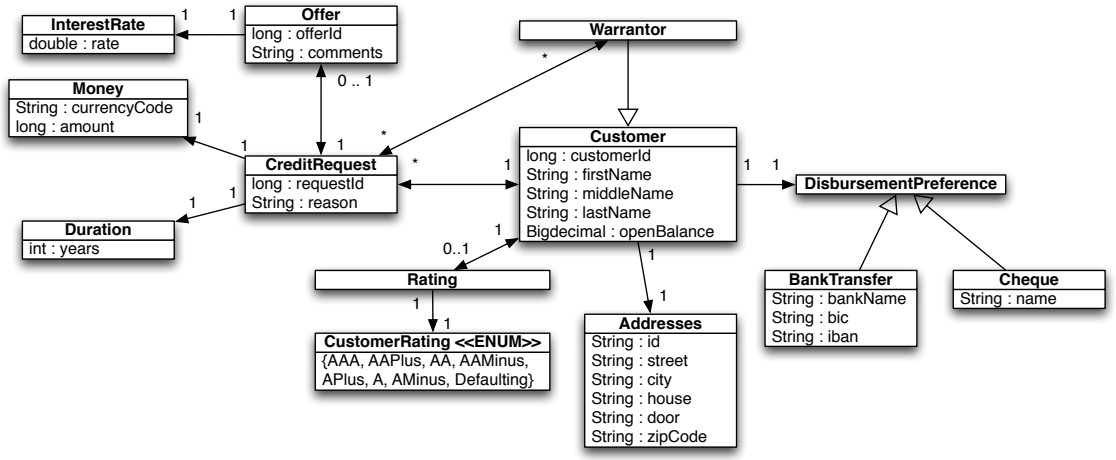

Figure 2: Credit Approval Domain Model

either the source code of these external services, or to the developers. However, you have access to a custom (non-UDDI) registry service, which returns disbursement services (as WSDL locations) for customer preferences (i.e., you pass the registry a disbursement preference, and it returns the metadata of the correct service to use as WSDL description).

After setting up the basic Web services of the process, you should also implement a frontend for bank clerks. For simplicity, this demo frontend can be as simple as a command-line client. However, if you do a more sophisticated Web-based frontend, your boss will be very happy (and we will award up to 5 extra points, depending on the extra effort you have put into the web app). You are free to choose any technology that you like for this task, e.g., you can use Java, C#, Groovy, Ruby or Perl for your command-line client. Similarly, you are free to use any Web application framework to implement the Web frontend, e.g., PHP, Grails, Ruby on Rails or JSF. However, please make sure to include detailed installation, deployment and usage information, so that we are able to test your frontend with minimal effort.

Company SOA Guidelines The following list is an excerpt of the company-wide SOA coding guidelines, put in place by the powers that be, and adapted for this specific project. Adherence to these guidelines is mandatory.

- The namespace of all services for this project is `at.ac.tuwien.infosys.aic11.services`.
- Service implementations need to be split into an interface and an implementation. The interface is named like the service functionality (e.g., `ContractManagement`). The implementation is named <Java interface name>Impl (e.g., `ContractManagementImpl`). All Web service annotations go to the interface.
- The names of all services should be <Java interface name>Service, e.g., `ContractManagementService`.
- All services need to be WS-I Basic Profile 1.1 compliant.
- You are generally discouraged from using simple data types as input or output of Web service operations (avoid the *Primitive Obsession* code smell[10]). RESTful Web services are an exception to this rule, where simple parameters can be used as path, query or header parameters. However, this exception does not mean that you are free to abuse simple types to represent complex data (e.g., representing a `java.util.UUID` as String is ok, but doing

[10] `http://sourcemaking.com/refactoring/primitive-obsession`

the same for a an address object is not). As a rule of thumb, if there is no clear, "natural", way to represent the data as a primitive type, it probably shouldn't be one.

- In RESTful services, metadata should be bound as header parameter. Identifiers should be bound as path parameters. Additional parameters should be bound as query parameters. Stay clear from matrix and form parameters.
- The namespace of all data transfer objects for this project is `at.ac.tuwien.infosys.aic11.dto`.
- Members of DTOs should generally be mapped to XML elements, except for IDs. IDs are mapped to XML attributes.
- Names of XML elements and attributes should be derived from the Java names, but with underscores (_) instead of camel-case, and without capitals (e.g,, `ProductName` becomes `product_name`).
- Make sure that optional information in DTOs is indeed marked optional in XML Schema, and that mandatory information is marked mandatory.
- Use `java.util.Logging` for logging purposes. All service operations need to log to INFO when an invocation has started (also log all parameters) and when an invocation has finished (logging the return value). All service operations need to log each thrown fault to SEVERE. Other logs should be written if useful. Writing to System.out or System.err directly is not permitted.
- Every service has its own logger, with a name equal to the service implementation class. The parent logger of each of these loggers is `at.ac.tuwien.infosys.aic11.services`.