Information Systems Institute
Distributed Systems Group (DSG)
VU Design Methods for Distributed Systems (184.153)

SS 2009
Submission Deadline: Sun. 7. June 09, 23:59

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 DSG Teaching Tool[1] 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 in groups of 2 or alone).
- **Plagiarism**: Please do not copy lab solutions of other groups, or any full solutions 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.
- **Deadline Extensions**: Deadline extensions are only given on special circumstances, and upon individual agreement with the course administration. 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.
- **Grading**: After the submission deadline, the DSG Teaching Tool will offer slots, where we will ask you questions about your implementation. This and your deliverable will determine your score. Every student needs to participate in every assignment. We will check that every student at least fully understands the solution of each 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.
- **Assignment Submission**: Once you have completed your solution please upload it to the DSG Teaching Tool. Submit a ZIP-compressed file containing all your source code and the required build file, but without any third-party libraries. Make sure that your code compiles and is runnable using Java 6 and Ant 1.7.1. Do not use any third-party libraries except the ones from Apache Axis2 (and related).
- **Submission Guidelines**: Your solution should contain an Apache Ant build script with the following targets:
  - `run`: This target should compile and start all services you developed. If your solution makes 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.
  - `clean`: This target should remove all files created during compilation.
  - `test`: This target should perform the JUnit tests. You should use the Ant jUnit task or invoke a Java main method which runs the tests.

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

- **Build Configuration**: Please make sure that your build script contains only relative paths and runs on any machine without the need for modifications. Specifically, Ant scripts generated by Netbeans sometimes contain hard-coded absolute paths, which should be eliminated. If necessary, you can use properties from a build.properties file. Please provide meaningful comments for each necessary modification to this file in your documentation.
- **Testing**: We will not give detailed instructions on what your JUnit tests / demonstrations should look like, however, we require the following:
  - Every functionality of your solution needs to be tested. That is, you need to invoke every operation at least once regularly, and you need to trigger every fault at least once.
  - Test the authorization implementation by trying to invoke an operation without the necessary rights.
  - Write tests for each supported protocol and each plugin.

Introduction

Welcome to the Design Methods for Distributed Systems lab in SS 2009. In this lab, we will implement a flexible middleware for business to business communication ajar to the Extensible Provisioning Protocol (EPP)[2]. This is a flexible protocol designed for allocating objects within registries over the Internet. The motivation behind EPP was to create a robust and flexible protocol that could provide communication between domain name registries and domain name registrars. These transactions are required whenever a domain name is registered or renewed, thereby also preventing domain hijacking. Prior to its introduction, registries had no uniform approach, and many different proprietary interfaces existed. While its use for domain names was the initial driver, the protocol is designed to be usable for any kind of ordering and fulfilment system.

System Description

Our system is peer based and every peer can communicate with each other peer in the system. According to EPP: before real communication can be started, a peer has to send a login message to the communication partner with username and password, which is checked with an Invocation Interceptor. You can hardcode usernames and passwords, but make sure that passwords are not saved or transmitted in plaintext (e.g. use MD5). After a successful login, other communication can take place. At the end the peer has to logout again, otherwise the whole session (all actions performed in it) must be rolled back and closed afterwards (the same must happen if the client sends an invalid message or causes another error). Each single invocation has to meet the ACID[3] conditions:

- **Atomicity**: If an invocation (or a part of it) fails, the whole invocation must be rolled back and an error message returned.
- **Consistency**: After an invocation, the stored data has to be consistently saved in your datastore.
- **Isolation**: Different invocations must not disturb each other. Also think about concurrent data manipulation.
- **Durability**: The result of an invocation has to be stored in your datastore.

Support long running transactions for session rollback. Each invocation in a session (which should be a transaction itself) belongs to a long running transaction. If a long running transaction is rolled back, all transactions in it should be rolled back.

The framework should be flexible regarding the communication protocols used (e.g Sockets, RMI, Web Services, ...). For communication use the XML format as described in the EPP standard under http://tools.ietf.org/html/rfc4930. You don't have to use the described XML extensions in your implementation.

As we dont provide all features of EPP, you should provide following functionality ajar to EPP:

[2] http://en.wikipedia.org/wiki/Extensible_Provisioning_Protocol
[3] http://en.wikipedia.org/wiki/ACID

- **Login**: At the beginning of communication with another peer, a login message has to be sent.
- **Logout**: When a client terminates its session with another peer, it has to send a logout message.
- **CRUD**: Each peer should provide functionality to other peers to create, read, update and delete resources. Not every peer should have all rights on all other nodes.
- **Transfer Request**: A transfer request message allows a peer to contact the owner of a certain resource and request takeover of that resource. The message must include a token that has to be sent back together with a "Transfer Execute" message if the owner acknowledges the transfer. So a resource takeover needs active interaction of its owner. If a transfer request already exists for a requestor, an error message is returned. However, different peers can place transfer requests concurrently. If there are multiple transfer requests for the same resource, the remaining related requests are deleted if the resource is transferred away.
- **Transfer Execute**: An execute message has to include the token that was received with the the transfer request message. If the token is correct, the requestor takes over the requested resource. The resource is thus completely transmitted to the requestor and all related data on the prior owner is deleted (including transfer requests). Also think about sessions that are using the transfered resource during transfer.
- **Transfer Cancel**: A peer can cancel its transfer request by sending a transfer cancel message.
- **Check**: This method sends all metadata about a resource to the requestor, except information about transfer requests of other peers – e.g. the data about the owner, day of creation etc.

An EPP error response MUST be returned if one of these commands cannot be processed for any reason. Use meaningful error messages.

A resource can be any kind of data (text, pictures, files, ...). You dont have to use a database/store resources persistently, you can use hashes instead. If you (really) want to use a database, you are allowed to use hsqldb[4] in file mode and hibernate[5]. You can include the necessary libraries in your submission (hsqldb, hibernate).

For security reasons, do not store or transmit resources as plaintext. Instead of a PKI use base64[6] encoding, which isnt secure, but simulates encryption. You have to store and transmit every resource base64 encoded.

Communication Framework

As basic commuication infrastructure you should use basic remoting patterns, based on Java sockets as communication protocol. For example, you can implement a minimal version of the leela SOA Middleware as described in *Pattern-Based Design of a Service-Oriented Middleware for Remote Object Federations, ACM Transactions on Internet Technology, ACM, 2008*[7].

[4] http://hsqldb.org/
[5] http://www.hibernate.org/
[6] http://de.wikipedia.org/wiki/Base64
[7] http://www.infosys.tuwien.ac.at/staff/zdun/publications/leelaSOAMiddleware.pdf

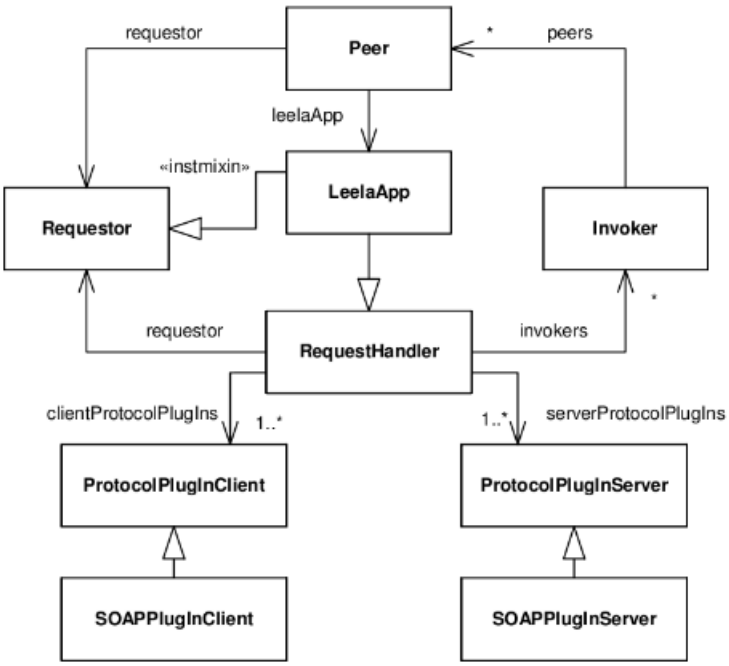

Figure 0.1: Communication Framework architecture

- Each EVS peer uses a RequestHandler, which implements both the functionality of a Client Request Handler and a Server Request Handler. The RequestHandler abstracts communication related issues, and is used to send requests to other services. It also receives requests from other peers, initiates the invocation of the peers service object, and sends the response back to the client.
- Remote invocations are abstracted using Requestor and Invoker: On the client side, the Requestor offers a dynamic invocation interface, which is used to build up remote invocations at runtime and hand them over to the RequestHandler. A Client Proxy shares a common interface with the Remote Object and maps its methods to calls to the Requestors invocation interface.
- On the server side, the RequestHandler hands an invocation over to the Invoker, which performs the invocation of the peer. Both Requestor and Invoker use a common Marshaller to transform program-language constructs to byte arrays, which are then transported over the network.
- The remote invocation data consists of method name and parameters, an Object ID (e.g.: the service name) as well as (protocol-specific) location information. Together the last two implement the Absolute Object Reference Pattern.
- As the middleware should be flexible regarding the protocols used, Protocol Plugins are used to support different communication protocols. For each communication protocol, a RequestHandler instantiates a pair of corresponding client and server plugins.

WS Plugin

Write a plugin for your middleware to support Axis2[8] or JAX-WS[9] web services as an additional protocol at the same time. Your communication framework should already provide a plugin system for this purpose. The WS sends/receives the same XML messages as the socket version and meets the same requirements.

[8] http://ws.apache.org/axis2/
[9] https://jax-ws.dev.java.net/

Score

Task	Points
Communication Framework	20
EPP implementation	20
Testing	10
WS Plugin	10
Total	**60**