Mobile and Distributed Systems

IT University Copenhagen

Jorge Y. Castillo
jyur@itu.dk
Gunn Weihe Reinert
gwre@itu.dk
Efrin Gonzalez Borrayo
efag@itu.dk
Rasmus Kreiner
rkre@itu.dk

November 22, 2013

Introduction

Task Manager Servelet

Description:

The main functionality of a *Task Manager* for this assignment is to store and show the tasks assigned to users. The *Task Manager* also needed to keep track of the task that are been executed.

This first version of the *Task Manager* did consist in finding strings, integers or booleans in a provide *xml* file such as: Name, date or a status field which indicates if the task has been executed before or not. Also if such elements contains node childrens in the xml data, there is the goal to read this values to be able to know who is aloud to read, write or execute such tasks.

Important Points:

Setting up local Server

Since most of class excises are required to be written in *Java* it was advice to use either *Eclipse* or either *Netbeans* to be able to develop this programs, furthermore since this application need to have a web server which is not located in any dedicated server there was the need to have a **localhost** such as Tomcat[1] which is an apache server that is able to run servlets and JSP pages.

Even though Tomcat's installation was quiet simple when the existing eclipse EDI if existing, does not have it as a extra package already. Other wise there a version of eclipse that does have this as a default package at the time of download.

Errors In Input File:

One of the main problems in the part of the assignments was mostly trying to get the file location with in the servlet, since path use in the example is the following:

```
FileInputStream stream =
new FileInputStream("C:/path/to/file/task-manager-xml.xml");
```

In this way it created some errors in the parser, since such file did not exist in either of the computers of the others group members, even though *FileInputStream* is the right solution for reading raw bytes from a file.

There still was the problem with the string that contained the file path, even tho there is a lot methods to solve this for example placing the file in a dedicated server with it respective API calls and security keys, but since there is only need for now to prepare the file to be in a local computer at the moment, the solution looked as follow:

```
    String root = System.getProperty("user.dir");
String filepath = "";

if(System.getProperty("os.name").toLowerCase().equals("mac os x"))
    filepath = "/src/resources/task-manager-xml.xml";

    else filepath = "\\src\\resources\\task-manager-xml.xml";

path = root+filepath;
```

By doing this, we can run all needed test without having to change any configuration or path strings in any of the projects since we define the path of the file depending of it s a mac or Windows machine.

[1] http://tomcat.apache.org/

Getting Users By ID

Relavant Theory

Using getInputStream [2]

[2]http://docs.oracle.com/javase/6/docs/api/java/net/URLConnection.html

TCP Task Manager and Server Xml Serialization

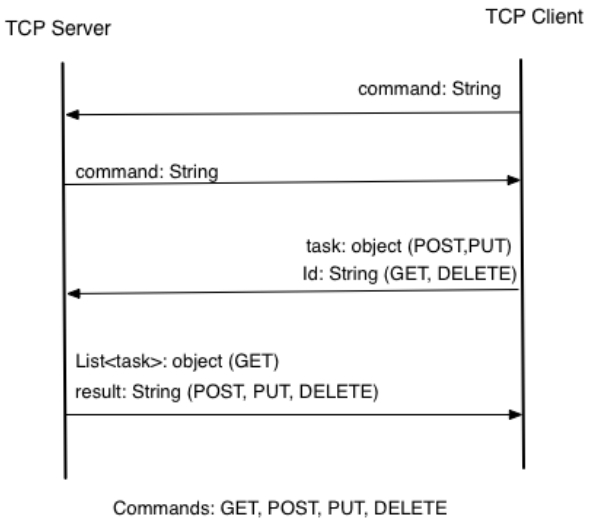
Description:

The *The Transmition Control Protocol***TCP** is part of the core protocols of the Internet, since TPC provides with a reliable, ordered, error-checked delivery of a stream of octets between programs running on computers connected to a local network, intranet or public Internet since it resides in the transport layer.[3]

Since web browsers uses **TCP** when they connect to the world wide web, there is the possibility to use its functionality to clients so data can be modified in the server and create task dynamically. This task manager **TCP** server offers some predefined commands that can be use to provide a service, such as:

1. **GET** to get items
2. **POST** to add items
3. **PUT** to update
4. **DELETE** to remove items

TCP server and client follow a strict structured communication based on the following conversational protocol:



The Task:

Since all tasks are based of dependency of the $task - manager.xml$ file to store, read, update or delete data with subsequent task runs by Task Manager TPC Server. In this lab exercise it was require to develop the code for the following functionalities:

1. Develop java serialization classes for task-manager-xml using Java Architecture.
2. Develop *TaskManagerTCPServer* and *TaskManagerTCPClient* classes.

[3] http://java.dzone.com/articles/understanding-transport-layer

Solutions:

```
public class TcpClient {
//extends how it is created
//which method it takes. parameter
//security ,improvements
//how handle multiple requests
```

Conclutions:

REST

Description

As an elaboration on the task manager application this chapter is about building a *RESTful* web service, such that multiple clients can access and modify the task-manager.xml file. For this we are using the JAX-RS API and the Jersey Framework. In order to keep control on multiple and concurrent requests on the web server, we are using a task provider, called *TaskManagerDAOEnum.java*, that functions as a singleton data access object. We will be using the already existing classes Task.java and Cal.java to handle serialization.

Implementation

The web service exposes four operations that are based on the explicit *HTTP* methods: *GET*, *POST*, *PUT* and *DELETE*.

Figure 0.1: REST Protocol

Relevant Theory

Up until now we have been dealing a set of modules that communicate with each other. In order to control the possible interactions between the modules, each module is presented by an explicit interface. This interface defines the procedures and variables that can be accessed by other modules.

In this original client-server model, both client and server are functionally specialized. This restricts the potential scope of applications. Web services return to this original structure, in which an application-specific client interacts with a functionalize interface over the Internet. In particular web services allow complex applications to be developed by providing services that integrate several other services. Due to the generality of their interactions, web services cannot be accessed directly by browsers.

A service interface can provide operations for accessing and updating the data resources it manages. The interaction between client and server is similar to RMI, where client uses a remote object reference to invoke an operation in a remote object. For a web service, the client uses a URI to invoke an operation in the resource named by that URI.

Web services either use a synchronous request-reply pattern of communication with their clients or communicate by means of asynchronous messages. An event-style patterns may also be used; notifications for example.

More generally web services are designed to support distributed computing in the Internet, in which many different programming languages and paradigms coexist. They are independent.

Loose coupling is a desirable design principle within web services. Programming with interfaces provides one level of loose coupling. The trend is moving towards more generic an simple interfaces in distributed systems. The World Wide Web and REST are example of this. This trend presents data-orientation, where the data is more important than operations.

REST versus SOAP

Microsoft developed in 1998 a platform-independent and neutral alternative to the Common Object Request Broker Architecture (CORBA), this new alternative was called Simple Object Access Protocol or SOAP for short. Together with WDSL and XML schemas, SOAP became the standard within designing the Web Services. A SOAP message consists of four elements.

- *SOAP <Envelope>*. This is the root element which contains two of the other elements.
- *SOAP <Header>*. Sub element of *<Envelope>* which contain application related information, such as authentication of users.

- *SOAP <Body>*. This is the content of the message.
- *SOAP <Faults>*. Sub element of *<Body>* which is used for error reporting.

SOAP was meant to be a simple protocol, but SOAP messages can become very long and complex, the alternative to SOAP is Representative State Transfer or REST for short. REST consists of clients and servers which share resources with each other. A resource can be anything, as long as the client is able to understand and use the resource. A representation of a resource is a document describing the state of the resource. The client can be in two states, application-state or at rest. When the client is at rest, a user can interact with the client and it creates no load on the server The emphasis is on the manipulation of data resources rather than the interfaces. Clients are provided with the entire state of a resource instead of calling an operation to get some part of it.

REST is based on simple point-to-point communication via HTTP and it uses plain old XML (POX). There are four operations in REST which defines how a resource can be accessed: **GET**, **PUT**, **POST** and **DELETE**.

There are five constraints which a Web Service must adhere in REST:

- Client-Server separation. The client is not involved with server functionality and the server is not involved with user interface.
- Stateless. The server does not contain any states.
- Cacheable. The client must be able to store responses from the server and thus remove redundant requests.
- Layered System. The client does not know that it is directly connected to the server or if the connection is going through a proxy-server. This enables scalability and load-balancing.
- Uniform interface. Uniform interface separates the individual parts of the system and adds Separation of Concern.

There are four principles which must be adhered in the uniform interface between client and server.

- Resources must be identifiable by a unique address or key.
- Manipulation of resources via representations.
- Self-describing messages. Each message contains enough information about how the message must be handled.
- HATEOAS[4]. Hypermedia as the Engine of Application State. HATEOAS describes how a hypermedia can control the state of the resource. This is done by using links to URL's which contain the operations that change the state.

[4]http://en.wikipedia.org/wiki/HATEOAS

Secure Task Manager

Description

The objective of this chapter is to implement a simple security protocol for our task manager. We have been developing a simple role based access control mechanism for tasks based on an authentication using ITU credentials. Moreover, we have been using one of the crypto algorithms to ensure security among all/some parts of communication.

Furthermore, in order to support role based access controls for tasks, the task-manager-revised.xml has been updated with a role on the task element.

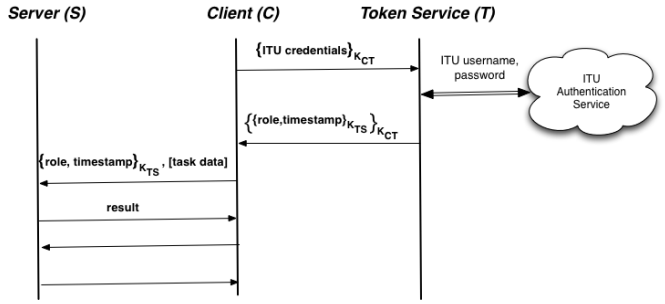


Figure 0.2: Task Manager Security Protocol

There are 3 interacting principals offering the same functionality. But the communication between different principals is encrypted with keys shared in between them. The three principals are:

Token Service:

It authenticates clients's credentials (ITU username and password). Furthermore, the token service also maintains authorization of tasks by having role mappings for a given ITU user account that matches to the roles assigned to the tasks in the task-manager-revised.xml. On successful validation of client's credentials, the token service will generate an encrypted server token containing client's role, timestamp, client's name and a encryption key for sharing between client and server (K_SC). *In addition to the server token, the token service will also send the encryption key* (K_SC).

Server:

The server offers two functions: authenticate and execute. The client will first authenticate itself by sending the server token. On successful authentication, the server will generate a nonce (N) and send it to the client. On the other hand, in case of authentication failure, the server provides error message only.

Client: The client first contacts the token service by proving ITU username and password, to receive the double encrypted server token. First, it decrypts the server token with its key shared with token service (K_CT), *to extract the encrypted server token. Then it will send the encrypted server token along with task data to ser*

Implementation

implementation will go here:

Related Theory

Unlike SOAP with ws-secure, REST does not come with a security protocol. But as any web application the Rest services also nee to be secure. The question is which threats are we concerned about, and how can we beat these.

We will have to assume that there is a mischievous person, who will attempt to read, remember, intercept, interrupt, modify and fabricate messages.

Design security protocols using:

- Cryptography
- Authentication
- Secure channels

When building web applications and services it is wise to predict worst case assumptions:

- Exposed interfaces
- Insecure networks
- Secrets between many participants hard to be kept for a long period
- Algorithms and code known to enemy
- Enemy has access to large resources

Security mechanisms:

- Cryptography
- Certificates
- Access control
- Credentials
- Firewalls

Even though REST doesn't come with a dedicated security protocol, there are external protocols that are recommended to use together with REST instead if implementing your own. OAuth is one popular choice. OAuth is an open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications. The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service.

Conclusion

Collaborative Task Manager with jGroups

Description:

Important Points:

Relavant Theory

Concurrency Control

Mobile Client

Appnedix Code

Appendix