Al Akhawayn University

Design Document

Homework 2

CSC 3374

Done by: Anass Zouine

Table of Contents:

1. Purpose of the Application
2. User Manual
3. API (Application Programming Interface)
4. Development Process
5. Code Structure

1. Purpose of the Application

The purpose of application is allowing a client to take control of a remote system. The client can get the properties (name and version) of the operating system of the remote system, take screenshots of it, or reboot it. The application enables the client to call these functions that will run on the server side while having the impression of having them running locally at the client side. For this reason, RPC paradigm is used by the application to make this possible.

2. User Manual

2.1. Running the Service Provider

To run the service provider, follow these steps:

- Open the terminal and go the directory of the project folder RSC_app_WS.
- Build the project using the command: **./gradlew build**
- Run the project using the command: **./gradlew run**

2.2. Running the Service Consumer

To run the service consumer, follow these steps:

- Open the terminal and go to the directory where 'consumer.py' is saved: when you reach the directory of the project, 'consumer.py' can be found in app/src/main/python/Consumer/**consumer.py**
- Run the consumer application from the command line using the command: **python consumer.py**
- The application will show the following menu:

  Available Services:

  1 - Get Name and Version of the operating system of the remote host

  2 - Take a screenshot of the remote system

  3 - Reboot the remote system

4 – Exit

- The application user will be prompted to enter a choice from the list of available services.

2.2.1. Description of Services Available for Consumer

- **Get Name and Version of the operating system of the remote host:** the name and the version of the operating system of the remote host, and they will be printed to the user.
- **Take a screenshot of the remote system:** a screenshot of the remote system will be taken and stored under the folder Screenshots. The folder Screenshots is in the same directory as consumer.py.
- **Reboot the remote system:** the remote system is rebooted.
- **Exit:** the program stops executing.

3. API

The API is the agreement between the service provider and the service consumer. It contains the prototypes of the methods that the provider can provide for methods' invokers. The methods that the consumer can invoke are:

- **getProperties() -> return: xsd:string[]**

The method retrieves the name and the version of the operating system of the remote system.

Parameters: None

Return: array of strings of length 2. The first string of the array is the name of the operating system, the second string is the version. If the method fails to retrieve the properties, null is returned.

- **screenshot() -> return: xsd:base64Binary**

The method returns the actual bytes that represent the screenshot

Parameters: None

Return: the bytes that constitute screenshot of the remote system’s screen. If the method fails to take the screenshot, null is returned.

- **reboot() -> return: xsd:boolean**

The method reboots the system.

Parameters: None

Return: *true* if the system was rebooted successfully, otherwise *false*.

4. Development Process

Service provider:

- Normally, before starting writing the provider-side program, we must start by writing the API. However, because I do not master WSDL language and it is not convenient to write the API using this language, I adopted a **code-first approach**. I started by developing the service real implementation in the target programming language (Java), then I used a tool to generate the service API in WSDL.

Service consumer:

- I used Python Zeep module to generate the client stub. Then I wrote the code that interacts with the user and invokes the remote methods through the stub as if they run locally.

5. Code Structure

Service provider:

- This part contains the real/business implementation of the methods that the client can invoke. Under the folder Provider, there are two classes. The class remoteControl that contains the business implementation of the three methods. @WebService annotation is added to this class to define it as a web service endpoint. The second class is the main class provider under which the web service is published. Many parts of the code are hidden like connection establishing, marshalling and unmarshalling of parameters. These steps are done by the skeleton.

Service consumer:

- This part contains the consumer that will consume the service that the provider provides. It invokes the functions as if they run locally. It only knows the prototype of the functions. The marshalling of parameters, unmarshalling of the result, connection establishing are done by the client stub.