Introduction

In the simple server/client model, we allowed different processes to communicate between each other remotely, but we were still bound with some constraints such as:

- We had to develop protocols from scratch due to the fact it is hard to adapt existing protocols. We also handle errors.
- We also had to manage connections (in java case it was sockets) and the streams.

Allowing such communication using the server/client model becomes hard to implement. Therefore, to ease our constraints and provide us with the luxury to invoke remote functionalities as if they were local, we adopted the RPC paradigm.

In essence, we have three parties that comes into play during the communication: Service Provider, Service Consumer, Service API (Contract).

Service Provider:

The latter is the part where we have the actual implementation of the contract or service API. Our service provider was coded in java.

(We will go into details after)

Service Consumer:

In this part invokes the functionalities put by the service provider. This part is not concerned of how the implementation is done but more of what is the expected output and what input to send. Our Consumer service was done in javascript.

Contract or service API:

This is the part that plays as the middleware between the *service provider* and the *service consumer*. It has the definition of the functionalities of the *service provider* that will be requested by the *service consumer.*

Into the bargain, we adopted the Code-first approach, we implemented the service provider first with its actual business implementation in java and we used *XML/SOAP Web services* to generate the WSDL and XSD file that will act as the service API between the two languages.

Asynchronous Promise Based Calls.

In the previous assignment, our client calls were done in a serial fashion. If our clients try to connect in parallel/ concurrently, the waiting time will be inevitable for our other client since our server is busy interacting with the current client. This type of paradigm which blocks the calls will not be feasible if our service is popular, and we have several clients. Therefore, we will call our functions Asynchronosouly, in a manner that our client calls will not experience a delay while our server is calling another server. Instead of one thread serving all the clients one after the other, we will have multithreading approach which will offer us the luxury of having different lines of excution.
But, within each thread the execution will be synchronous.

Moreover, it is a huge waste of CPU time.

Technology Enablers

Now, since we made context, we will go into details on how the *XML/SOAP Web Services* Technology works.

1. ***XML/ SOAP Web Services*** is an XML based protocol that stands for **S**imple **O**bject **A**ccess **P**rotocol and defines how two parties (service consumer and service provider) will communicate with each other.

XML/SOAP Web services dictates that all information/data exchanged between the service provider and service consumer must be in a common unified format and in the SOAP case it is XML so both parties can understand what is communicated. In details, the XML has structure constraints, meaning it needs to have Envelope, Header and Body we refer to it as the SOAP message. For the communication to be successful the consumer needs to know the functionalities available, the parameter requests and responses and the way to call the web services. For that the Service Provider publishes an interface for its services and attributes in a URL that will be accessed by ***WSDL(explained below)*** file which is an XML based file.

To get hold of the WSDL by the consumer we generally use UDDI but, in our case,, we provide the direct link of the WSDL since both parties are running on our local machine.

Moreover, XML/SOAP protocol is a language agnostic technology meaning that there are no constraints on the language chosen on both sides (i.e., Service Provider side and Service Consumer). In other words, we can serialize and deserialize objects in a unified and establish connection no matter what languages we have in the provider side or consumer side.

1. **WSDL** is an XML based interface (not a protocol) that defines and describes the functionalities of the service provider which is called and used by the service consumer. In other words, it is the contract between the provider and the consumer that gets shared to the consumer for potential request. The WSDL file has what are the methods that can be called, the arguments and the return type.

2. **XML** stands for **e**Xtensible **M**arkup **L**anguage. It is a markup language to format and structure data using tags either for storage, data transmission etc.

3. ***Javascript***: We handled the three function which are, reboot, get os Name, and get screenshot, in an asynchronous way. We programmed our consumer in javascript. The latter has a different concurrency model, meaning that the way the code is written it seems synchronous but behind the scenes the asynchronous approach is working. In

other words, the runtime takes our call and execute it in a dedicated thread.

We adopted the promise based asynchronous calls approach to make our function calls execute async.

4. **Promises:** In essence a promise in JavaScript is like promise that could be between people in a real life. The promise contains two parts, either fulfill or reject. If everything goes as planned the promise will be fulfilled, otherwise nothing goes as planned the promise will be rejected. In terms of JavaScript, we define the work that needs to be performed async. Then, we publish the outcome if it is rejected or fulfilled using *resolve* and *reject*.

We then , defines the plan using .then to execute a plan once the outcome is published. In an nutshell, a promise is a proxy for a value which can be not necessarily known upon creation and it represents the eventual completion or failure of an async operation and the resulting value.