Distributed Systems (H0N08A)

Report: Remote communication

Martijn Leplae *(r0737706)*
Andreas Hinderyckx *(r0760777)*

November 5, 2021

Question 1 **Stub**: The stub is representative of the remote object one is communicating with. It's ought to create a black box from the communication with the remote entity. As such, it acts as a gateway for the client to send outgoing requests to the server side (remote entity). To achieve this, the stub applies marshalling to the requests of the client.

Skeleton: the skeleton is similars to the stub, but it's situated at the server side (remote side). All client requests are made on the skeleton: as such it unmarhalls these incoming requests and calls the corresponding methods on the server-side objects.

Question 2 The skeleton at the server side implements the Java `Remote` interface as its methods may be invoked by non local clients. On the other hand, the plain old Java objects (POJOs) at the server side which should be able to received by the client implement the `serializable` interface as they should be marshalled when sent to the client side.

Question 3 The java `rmiregistry` (RMI registry) is a central unit which advertises the methods and objects available at the server side which can be used by the clients. Firstly, a server side component registers its services with the registry. Afterwards, a client can look up these services. Finally, the client can directly call methods on these looked up objects for example.

There is no need for a registry service in other remote communication technologies as the clients are ought to know the services provided by the server side. In SOAP for example, the services could be advertised on a web page or its WSDL. Based on this information, the client can compose its XML-request.

Question 4 The WSDL file provides extra information with regards to the binding style, the transport protocol being used, the XML encoding used, etc. More specifically, operations that can be invoked by the client are specified in the 'bindings' section of the WSDL file. Herein, every message corresponds to an operation and the message's parts their actual meaning are defined by the corresponding binding.

The used SOAP binding style is of type `Document`, which means that — in contrast to RPC Style — there's no type information and in general no SOAP formatting rules. The transport configuration specifies the transport layer protocol being used: in this case HTTP is used.

Question 5 The main advantage of the hypermedia-driven approach is that the structure of the server URL's may be modified without breaking existing code. This is because each response includes hyperlinks to other server endpoints which can be used to discover the server side API. The coupling is reduced based on this reasoning. Future upgrades can be discovered more efficiently by developers, as the reflection of this hypermedia-driven approach always provides links to related conepts in its responses.

Question 6 Commonalities:

- Both of them offer an interface of the available methods.

Differences:

- Java RMI has support for remote object references (as specified in its RMI interface), whose methods can be invoked, whereas gRPC only supports the remote invocation of the methods specified in its `.proto` file.
- The `.proto` file is used to automatically generate the required classes, whereas the RMI interface is simply a facade to map the client requests onto the corresponding methods.
- In order to connect to a Java RMI interface, the server must first register its services with the name service and the client must connect to this name service to call the registered services. With gRPC, there's no notion of a name server.

Question 7 The advantage of gRPC's code generation is that the associated `.proto` file is written in the proto3 language. It's universally usable with different underlying languages, whereas de Java RMI interface is only usable with the Java language. The disadvantages of code generation in general are:

- There must exist support for code generation of the language that you're using.
- One must recompile the project eacht time a change is made in the interface, for example each time the `.proto` file is modified.
- The code is automatically generated, therefore the programmer has initially no notion of its strucutre, which can make it harder to debug possible errors. Therefore, coupling with generated code must be kept low.

Question 8 The advantages of the binary format are:

- The can be serialized more quickly thanks to the binary format,
- The binary payload only uses a small amount of bandwidth, which makes it also suitable limited bandwidth scenarios.

The disadvantages are:

- The binary format is not human-readable, whereas JSON files are,
- gRPC doesn't have browser support, whereas SOAP and REST do as they're based on the text based HTTP 1.1 protocol.

Question 9 The WSDL interface is based on XML whereas the gRPC interface is based on the proto3 language. The former is much more verbose, which makes it less readable for the implementor and is slower to consume for JavaScript-based browsers. The WSDL language specifies services in terms of ports, whereas there's no notion of ports in the `.proto` file.

Personally, we find the `.proto` file learning curve to be steeper, although it does offer a more concise way of specifying the interface. The WSDL file is very verbose and unclear to read in our opinion, which makes the `.proto` file more appealing.

Question 10

(a) One shouldn't use Java RMI as it only supports Java applications, which isn't suitable for this use case where all applications should be able to use the API, independent of their underlying language. REST, SOAP and gRPC can all be used for this use case. Advantages of REST are that it's widely used and supports readable JSON-type responses and HTTP requests. On the other hand: gRPC can be implemented really efficiently due to it's support for the binary format.

(b) As mentioned above, for high performance scanrios one should use gRPC as it's built on the highly efficient binary format. Futhermore, gRPC's interface (`.proto` file) can be used with any language, which makes it an excellent option for this use case as there are various programming languages being used.

(c) SOAP, REST, gRPC and Java RMI could be used for this scenario. The disadvantage of choosing for Java RMI is that the company can't add additional applications or services which aren't written in Java, neither can it expand its infrastrucutre later on to non-Java applications or services. The other options do support this extensibility, with their own advantages and disadvantages as mentioned above.