Distributed Systems (H0N08A)

Report: Remote communication

Andi Škrgat *(r0876363)*
Daniel Rey Šparemblek *(r0883565)*

February 7, 2022

1. What is the role of stub and skeleton in Java RMI? Do the other remote communication technologies have similar concepts?

A stub is an object that resides on the client-side. It acts as a gateway for all outgoing requests from the client. The stub first initiates a connection with the remote Virtual Machine and then marshals the parameters to it. Then it waits for the result, unmarshals the return value or exception, and returns the result to the caller. Skeleton in Java RMI is an object which reads parameters for some remote method, calls a method on the remote object, and later marshals the result to the caller. Sun RPC also uses stub functions, client stub on client-side and server stub on the server-side. Their functionalities are the same as in Java RMI(marshaling, creating packages for transferring over the network, and unmarshaling). The difference is that the server stub executes a local procedure call to the actual server function. In CORBA a stub is a mechanism that issues requests as a client; the skeleton delivers them to the CORBA object implementation. Of course some other technologies have different approaches, e.g gRPC don't use stub/skeleton approach.

2. What is the difference between serializable and remote? In your Java RMI assignment, which classes were made serializable and which classes were made remote? Motivate your choice.

The first difference is that "Remote" is a class and "Serializable" is an interface. The remote interface serves to identify which methods can be called from a non-local machine. Only methods from objects which extended the Remote class are visible and can be accessed remotely. Serializable is used for all objects which are transferred as arguments from client to server or as a result from server to client. BookingDetail is the only class that extends Serializable because this is the only class whose instance is transferred as an argument in the addBooking method. IBookingManager is Remote so it "publishes" its methods to the outside. Every method from an object which extends Remote must throw RemoteException. In Java RMI all primitive types and all serializable objects are passed by value while remote objects are passed by reference.

3. What role does the RMI registry plays? Why is there no registry in the other remote communication technologies?

RMI registry is the binder for Java RMI and it is used as a name service. It stores mappings between names and references to remote objects which are hosted on the computer. RMI registry must be run on the server. There are two main methods for registering: bind and rebind and the only difference is that rebind can be called even if there is an object mapped to a concrete name. Clients then need to use the lookup function from Registry so that they would be able to call the right method from the object registered on the RMI registry. Other remote technologies don't use naming registries because they work differently, they don't assume same technology for client and server. For example, gRPC searches for IP addresses.

4. Which concepts and configuration do you find in the WSDL in addition to what you specified in the meals.xsd? What is the SOAP binding style and transport

configuration?

Messages are part of WSDL which are used to define what kind of messages can be received or sent and they consist of different parts where each part is defined by a type. Types can be found in our meals.xsd schema: 1. complexType which contains other complex or primitive types and 2. primitive types like int, string, float... Another XML element that is in WSDL and not in our created schema is portType which defines an interface and that is actually the set of operations offered by one or more endpoints where each operation consists of input/output messages. Binding is used to tell us how the above-mentioned portType is bound to the transport protocol. Bound elements then contain information about which transport protocol is used and additional information that is specific to it. WSDL SOAP binding can be RPC style or document style binding and both can have encoded or literal use. Both communication style models are used to translate WSDL binding to a SOAP message body. With document-style SOAP body can be structured in any way as long as the content of the SOAP message is a valid XML. RPC style implies that the SOAP request body contains the operation name and the set of method parameters. With literal use model body contents should follow what is written in the user-defined schema and in this way we can easily validate the message body and transform the message using e.g XSLT transformation language. Encoded use model allows us to write messages which use XSD datatypes and it's not connected to any user-defined schema. Transport configuration is set to "http://schemas.xmlsoap.org/soap/http" which means that HTTP is used as the transport protocol. Some other URIs can indicate other transports for example SMTP, FTP...

5. Level 3 RESTful API's are hypermedia-driven. How does this affect the evolution of your software, and more specifically your APIs in terms of coupling and future upgrades?

The major advantage is that users don't need to have prior knowledge of the service or hardcode the URI structures for resources because HATEOAS(Hypermedia as the Engine of Application State)) allows the server to make URI changes as the API evolves without breaking the clients. That is only possible with this way of implementation where the server provides links to access the resources and discover available actions dynamically. REST is stateless so in this way we are allowing users to determine the state of the current item that they are on. There are several issues with this approach: slower responses, more data, more work but maybe you can cut some unnecessary calls that are not accessible to a particular item. API becomes more flexible because new features are immediately available to your users. It is also possible to upgrade certain parts (change resources, require some new method parameters) without necessarily breaking backward compatibility but this assumes that the API is correctly implemented by users. To conclude, coupling and future upgrades can be managed in an advanced way if one very important condition is met and that is: users must implement the API correctly.

6. What is the commonality between Java RMI interfaces and gRPC's .proto files? In which aspects are they different? Commonality is that they both have some structure(in Java RMI interface which extends Remote, gRPC has service) in which they specify what are exposed methods and what objects they accept as arguments and return as a result of operation. Now the difference is that this objects are also specified in .proto file but in Java RMI, client usually has jar file with specifications of this objects. That means that this intermediate objects will also be created when using gRPC while in Java RMI they won't be because client already has all necessary information due to the fact that both client and server are using Java technology.

7. What is the advantage of using code generation in combination with an IDL over a language integrated solution such as Java RMI? What are the downsides of using code generation from an implementation perspective? By using IDL we don't restrict ourselves only on one technology(langauge) which means that our service will be available to greater number of customers. If we look from the company's perspective, some other services which need to communicate with this specific service, don't need to use same programming language. From

the other side developing such an application can be quite difficult and messy because instead of focusing only on service development, we also need to represent it in IDL. Problems occur when we need to change behaviour of some object in our service or contract that some method offers us because we also need to change it in our IDL. In the example of gRPC there is also a problem due to the large number of generated classes and it can get very difficult in the sense of having clean and structured code.

8. Compared to SOAP, REST and other text-based protocols, gRPC uses binary protocol buffers for transferring RPC messages. What are the advantages of using a binary format? Which are the downsides?

Since protocol buffers use binary format they are more efficient and processing them is much faster when compared to text-based protocols like SOAP and REST which use XML and JSON. On the other side working with the binary format is hard because you cannot simply read the data. It is also harder to test your implementation, e.g when you are building a REST service you can easily use Postman, Curl, or some other tool to check the response from your route. With protocol buffers, you can't. Working with JSON is easier because there are no strict rules for defining content like in protocol buffers and since there are no schemas it's very hard to decode them.

9. Both SOAP and gRPC use an IDL to specify service interfaces (WSDL and .proto files respectively).What are the major differences between these IDLs, and which one do you prefer and for which reasons? First big difference is that SOAP uses XML for specifying WSDL service interfaces while gRPC uses .proto files. Protobuf IDL is designed to be human readable/writable format while WSDL is intended as a machine readable/writable format. Changing the WSDL service means we need to start service again and regenerate the WSDL file from the server whereas with the .proto files changes can handle most of today's IDE's when the project is rebuilt or even after saving .proto file. Choice between WSDL and .proto files depends on what I need to do. For building web applications I would use SOAP and WSDL because SOAP is language agnostic, it can be implemented over many protocols like HTTP, SMTP and FTP while gRPC assumes that both client and server need to support same Protocol Buffers specification which is very serious requirement. Debugging gRPC based services is also very difficutl because of binary format while WSDL is text based. Another problem is that gRPC uses HTTP/2 but there is very large number of websites which still don't support it. However, for all performance demanding applications I would use gRPC and their .proto files.

10. Suppose that you are tasked with developing the following applications, which of the four remote communication technologies (Java RMI, SOAP, REST, gRPC) would you use to realize them? What is your motivation for choosing a specific technology?

a) I would use REST because it's a remote communication technology that is very easy to use and to develop applications with it. Almost all browsers support working with JSON format which is not true e.g for gRPC and its protocol buffers.

b) Since efficiency is very important, I would use gRPC with protocol buffers. They allow us to write applications in a very efficient manner because of binary format. It also works with HTTP/2 versions that support multiplexing over one TCP connection and dividing the messages into frames in binary format. So in general, HTTP/2 offers us faster and more reliable communication and gRPC uses that. Another feature of gRPC is that serializing and deserializing data structure between various languages is very easy and that is what we need in this case.

c) In this case, we could use several approaches but since everything is written in Java, Java RMI would be a good choice. It allows us to work with remote objects in the same way as if they were local, respecting an object-oriented approach and design. Developing an application is very easy and it's almost the same as if we are creating a simple non-distributed local app. That also means that all JDK features are included like type checking. Of course, there is a chance that the company could add an app that is not written in Java, that would mean Java RMI is not an option and in that case, I would choose REST communication technology.