Assignment 4

Georgia Koloniari
Salteas-Kalogeras Panagiotis

2. "Remote Procedure calls and Java Remote Method Invocation".

RMI and RPC differ not just in detail but in the very set of assumptions made about the distributed systems in which they operate. These differences lead to differences in the programming model, capabilities, and way the mechanisms interact with the code that implements and built the systems. In particular, while the basic RPC assumption for the underlying systems is heterogeneity, the RMI assumes a homogeneous system where all the machines are running the Java Virtual Machine. RMI takes this homogeneity assumption even further, by assuming that all objects in the system are written in Java, while RPC supports language heterogeneity by a machine neutral interface definition language (IDL), which is responsible for marshaling and unmarshaling data types between different implementation languages and running environments. RPC introduces proxies (stub and skeleton), which are compiled into the calling code and convert any call into the IDL. However, this introduces some limitations in the data types that can be passed around the system. The biggest limitation is the static nature of the data, since RPC cannot support polymorphism. Whereas, RMI can fetch the code for any unknown object at running time dynamically. Thus, it allows the dynamic code load during execution, unlike RPC where all code needed for inter-process communication must be available some time prior to the communication. Another difference is that in RMI stubs are generated on the implementation class of the object to which they refer, rather than reflecting only the declared remote interface as in RPC. These stubs support all the remote methods that the remote object's implementation supports. There is also a difference in the notions of ownership and responsibility. In RPC, the stub code is the responsibility of the calling client and can be linked ahead of time into that client. In RMI the stub for a remote object originates with the object and can be different for any two objects with the same apparent type. The system locates and loads these stubs at runtime, when the system determines what the exact stub type is. So the most important difference is that in RPC, the result of writing an IDL interface is a static wire protocol, which defines the way the stub will interact with the skeleton. In RMI, the interaction point has moved into the address space of the client of a remote object and is defined in terms of a Java interface whose implementation comes from the remote object itself.