Report 1

Application of Design by Contract in Distributed Systems

Ashwin Gandhi
axg4794@rit.edu

While implementing a distributed application one has to deal with a number of difficulties in setting up connection, concurrency, communication, and inconsistencies between what the client application and the server application expect from each other. A lot of work has been put in to come up with object oriented solutions for robust and fault-tolerant distributed applications. This project attempts to come up with a solution for handling problems related to invalid and erroneous service calls made by client and also keep a check on the result returned by the server after it has serviced the request in a RMI-based distributed system by using Design by Contract methodology.[1][3] Though this report only deals with the pre-conditions and not about post-conditions. In a Client-Server based distributed system, if the contracts performed a check on the validity of the request before the request reaches the server, then it would reduce a lot of load on the server as it does not have handle erroneous requests.

The Method: Design by Contract

Design by Contract is a programming methodology for designing software which extends the ordinary definitions of abstract data types with pre and post conditions. These conditions are called contracts [2]. Making contracts executable can enhance how components respond to an erroneous and inconsistent state, allowing the developer to better locate and handle faults. Also it is a good design technique because each component has a clearly defined expected behavior in its contracts. This feature reduces the possibility of ambiguities and smoothes out the process of integration of the different components of an application.

What is RMI?

Java RMI allows clients to execute remote function located on the server using the same semantics as local functions calls. It works as explained below:

1) The Server Implements a remote Interface and the client knows about this interface.
2) The server first binds itself to the registry.
3) The client looks up this remote object in the registry which acts as an RMI service proxy.
4) The client program makes method calls on the proxy object, RMI sends the request to the remote JVM, and forwards it to the implementation.
5) Any return values provided by the implementation are sent back to the proxy and then to the client's program

Project Plan:

A “contract.xml” file will be defined at the server which lists out all the specifications and constraints to make a valid call to the server. When a client makes a call to the service, according to usual working of

RMI, it uses the server object (RMI proxy)registered in the registry.Here instead of that, it uses a Server Proxy, which is a proxy of the server object registered in the registry. Proxy objects are usually declared so that the clients have no indication that they have a proxy object instance.A proxy forces object method calls to occur indirectly through the proxy object, which acts as a surrogate or delegate for the underlying object being proxied. In Java, a proxy has an invocationhandler associated with it. This invocation handler has an invoke() method where operations can be performed before going ahead with the function call. So, it basically acts as an interceptor. Here, validity checks could be performed on the method which has been intercepted as specified in the contract.

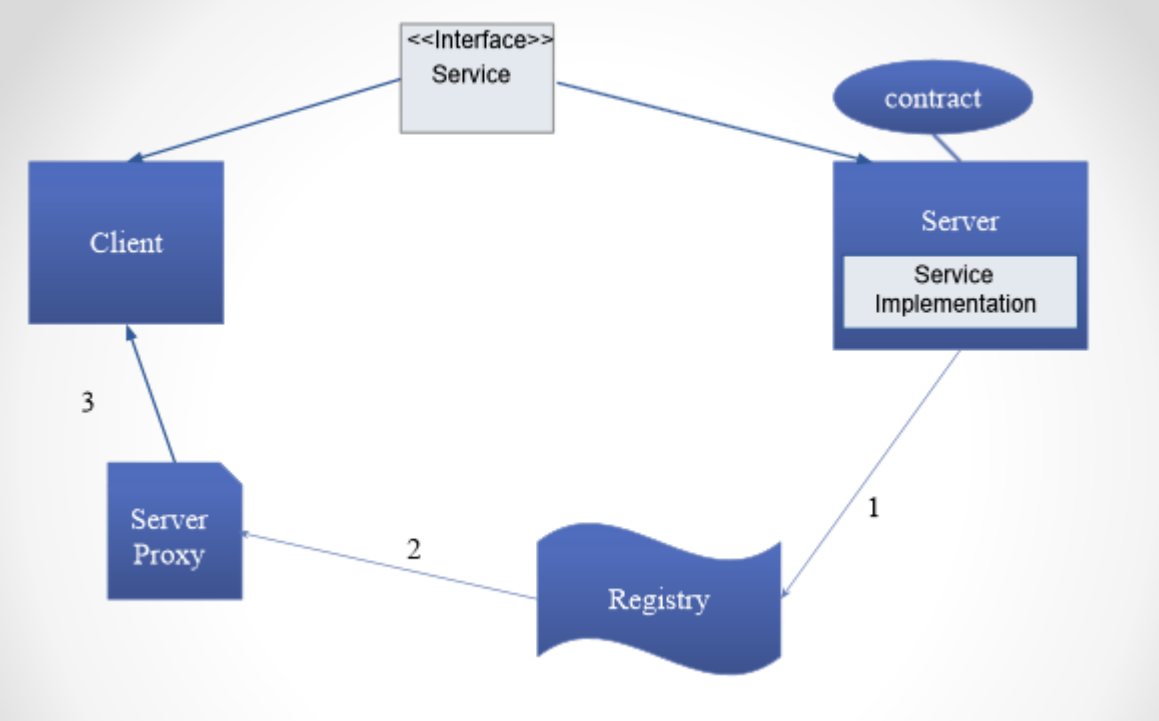

Figure 1: Setting-up RMI environment

1) Server Registers itself in the Registry
2) Server Proxy looks up registered server object in the registry
3) Client initializes a Server Proxy object which contains the object from the registry looked up in step 2

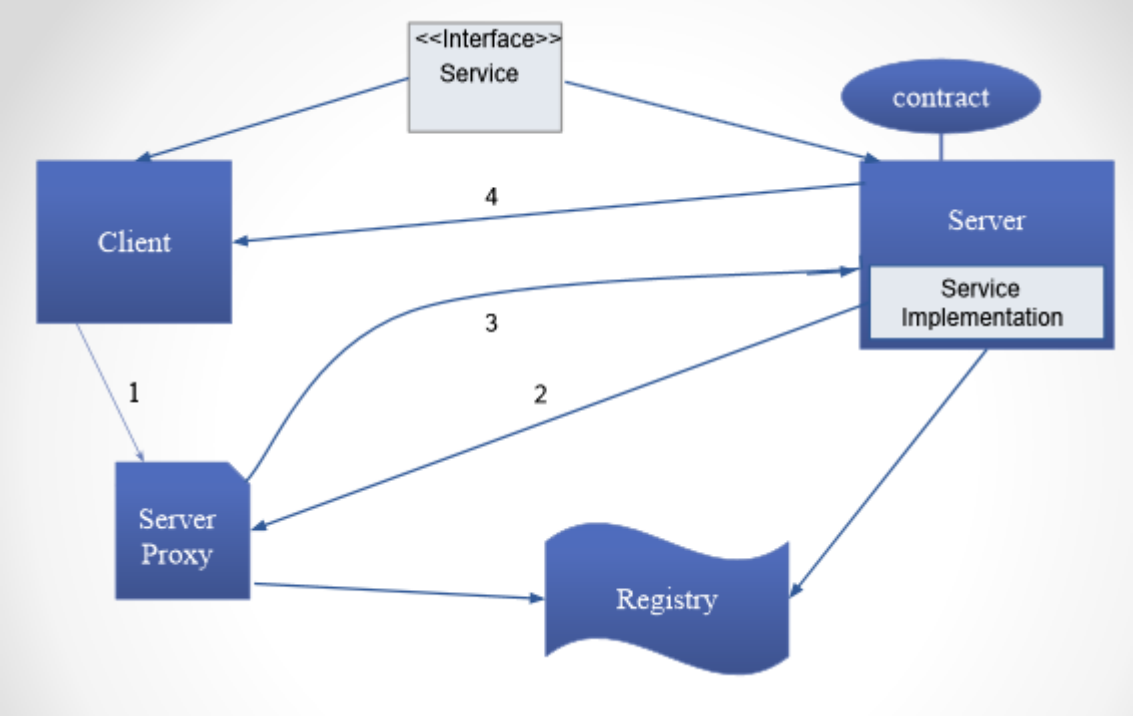

Figure 2: Servicing a Request

1) Client makes a method call using the Server Proxy object which contains the looked up server object from the registry
2) Server Proxy intercepts that call and gets the serialized contract object from the server.
3) The Server Proxy passes the intercepted method to this contract object which checks for its validity and if valid, goes ahead with the call.
4) The server returns with the results to the client

Thus, when the client makes a call for the service using the server proxy, it is intercepted. Inside the invoke() method, the server proxy fetches a serialized contract object generated from contract.xml by the server, which contains a Hash Table and number of functions to perform checks, from the server over RMI.

The server proxy will pass the intercepted method to this contract object, which will do all the described checks and return a boolean value which will advise the server proxy on whether to go ahead with the function call to the service. This check for the validity of the function call, before the call is made to the service, takes the burden of error handling(of this kind) off the server. This would help the server to concentrate its resources on serving other calls.

Problems:

This approach would cause more network traffic.A couple of messages are added, as the client call goes through the server proxy and also, a fetch operation is required to get the contract object.
Also, this approach uses reflection and it is inefficient in regards to time when it comes to using reflection on an object.

Project Outcome:
The outcome of the project will be an approach to partially automate the addition of contracts to Java RMI calls.

References:
1) A Framework Managing Quality of Service Contracts in Distributed Applications by Stéphane Lorcy, Noël Plouzeau and Jean-Marc Jézéquel Year 1998
2) Design by Contract to Improve Software Vigilance by Yves Le Traon, Benoit Baudry, and Jean-Marc Jezequel. Year 2006
3) FORMI: Integrating Adaptive Fragmented Objects into Java RMI by Kapitza R, Domaschka J, Reiser HP, Schmidt H