A JAVA META-REGISTRY
FOR REMOTE SERVICE OBJECTS

Dott. Marco Bianchi
Netlab, Istituto di Analisi dei Sistemi ed Informatica del C.N.R
Viale Manzoni, 30 – 00185 – Rome
Italy

Dott. Carlo Gaibisso
Istituto di Analisi dei Sistemi ed Informatica del C.N.R
Viale Manzoni, 30 – 00185 – Rome
Italy

Ing. Maurizio Vitale
Servizio Reti di Comunicazione del C.N.R.
P.le Aldo Moro, 7 – 00100 – Rome
Italy

ABSTRACT

In this paper we propose a meta-registry system, in what follows referred to as Netlab Java Meta-Registry (N-JMR), that integrates in a single access point the content of several RMI registries. N-JMR associates the same *N-JMR service name* to the absolute addresses of different instances of the same RMI class. As a consequence N-JMR can select the best remote service, among those associated to the same *N-JMR service name*, according to some evaluation criterion, selected among those provided by the system. In addition it is possible to define and add new evaluation criteria to the system. Furthermore, N-JMR extends the functionality of the RMI naming service by a directory service, thus making it possible to associate any information of interest to *N-JMR services names*. This information can be accessed both at run-time by client applications and by programmers of distributed systems through web applications.

KEY WORDS

IP Networks, Java, Naming and Directory Systems, RMI, Software Agents, System Architecture.

1. INTRODUCTION

The last five years have seen the achievement of object oriented programming languages, Java over all, and of the connectors and components based approach in the design and implementation of distributed systems and applications.

At the same time, thanks to the widespread diffusion of computer networks, the interest of major software technologies producers has been captured by distributed systems potentiality [1]. This interest resulted in the definition of distributed object oriented software architectures, such as CORBA [2], DCOM [3], EJB [4], RMI [5] and RMI/IIOP [6].

As a consequence Web applications rapidly becomes a natural point of convergence for the world of Internet an that of distributed applications, with the aim of making the services provided by these applications available to a large community of users.

With respect to this technological scenario, that of locating remotely executed objects, simply remote objects in what follows, is one of the critical problems to guarantee an effective and widespread adoption of distributed software architecture.

CORBA, DCOM, EJB, RMI and RMI/IIOP make use of special archives, referred to as registries in what follows, implementing a naming service, to access remote objects. Registries maintain for each registered remote object, the association between its registration name and a remote access reference. Despite these technologies solve the problem of locating remote objects in a quite uniform way, the access to registries is strongly related to the particular technology. Furthermore they implement different criteria in selecting a remote object to satisfy a service request and adopts different documentation management systems, if any.

Netlab, the network and multimedia laboratory of IASI, started a research and experimentation activity with the aim of defining the architecture of a meta-registry system for the access and the management of remote objects and the related documentation. This system realises a single point of access to services, thus making it possible to locate the best remote object according to some evaluation criterion.

In this paper, we will restrict our attention to the RMI technology, with the aim of moving a first step toward the implementation of this meta-registry.

The RMI architecture defines a Java distributed object model that integrates naturally into the Java programming language and the local object model, extending the safety and robustness of the Java architecture to the distributed computing world.

A RMI registry runs on each host executing remote service objects and accepts requests for services.
A remote objects is created by the server by locally instantiating an object and by its registration in the RMI registry with the name of the service it implements.
Once the object has been registered RMI creates a listening service that waits for requests of access to the implemented remote service.

The RMI registry is accessed by invocation of the "`lookup()`" method, whose input are an URL specifying an host name and a service name. The method returns an access reference to a remote object implementing the required service. The URL, *RMI absolute address* in what follows, takes the form:

rmi://<host_name>[:<name_service_port>]/<service_name>,

where *host_name* is a name recognised on the local area network or a DNS name on the Internet. The *name_service_port* is required if the naming service is running on a port different from the default one [7].

The main drawbacks of the naming service provided by RMI, with respect to the goals of this paper, are the following:

- each RMI registry has its own naming domain, which is separately managed from other naming domains;

- the naming system provided by RMI only associates a name to a remote access reference, for each registered remote object.

In fact:

- in presence of several registries, each one with its own naming domain, managing the absolute addresses of remote services could result in a very expensive activity and potentially introduce inconsistencies. For example, RMI does not provide any automatic mechanism to register different instances of the same remote service with the same name in different registries;

- RMI technology does not make it possible to associate with each registered service any information except for its registration name. Useful information could be a brief description of the provided functionality, how to contact the service authors, etc.

In this paper we propose a meta-registry system, in what follows referred to as Netlab Java Meta-Registry (N-JMR), that integrates the content of several RMI registries in a single access point by associating the same *N-JMR service name* to the absolute addresses of different instances of the same RMI class. In this way, it is possible for N-JMR to select the best remote service, among those associated to the same *N-JMR service name*, according to some evaluation criterion.

Furthermore, N-JMR extends the functionality of the RMI naming service by a directory service, thus making it possible to associate any additional information to the registered *N-JMR services name*. This information can be accessed both at run-time by client applications and by programmers of distributed systems through web applications.

The paper is organised as follows, section 2 presents the functional features of N-JMR and the advantages introduced for the programmers of distributed applications and systems; section 3 describes the architecture proposed for N-JMR; section 4 provides some implementation details of the N-JMR prototype; section 5 concerns with N-JMR functional correctness; finally, section 6 contains some concluding remarks.

2. N-JMR OPERATION MODEL

As already stated, any application relying on the services offered by a remote object must be aware of its absolute address. By this address it is possible to access the proper RMI registry (step 1 of figure 1) and to obtain the reference (step 2 of figure 1) by which accessing the remote object of interest (step 3 of figure 1).

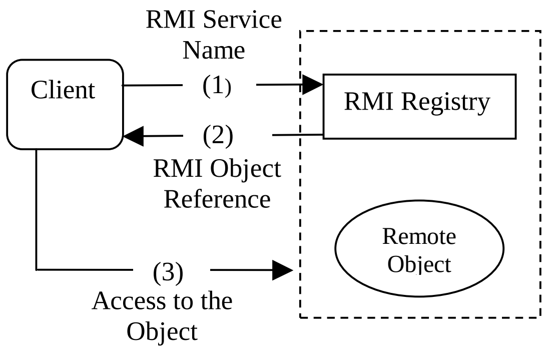

Figure 1

This behaviour is especially suitable for distributed systems with a single RMI registry. In presence of several and not replicated RMI registries, some problems might arise to consistently deal with the corresponding naming domains.

For example, assume a *remote class* named *Calculator* to be available and that *n* different instances of this class have to be executed on *n* different hosts. The *n* instances should be registered with the same *RMI service name*, namely *RemoteCalculator*, into the *n* involved registries. RMI does not provide any automatic mechanism to achieve this goal, and consequently the registration activity can be tedious and time consuming. Let us assume, for example, that *n-1* registrations have been successfully accomplished, but in the last registry the name *RemoteCalculator* has already been used to register an instance of another RMI class. As a consequence the previously accomplished *n-1* registrations have to be cancelled and the registration activity has to be the started from scratch.

Furthermore, the way in which RMI manages its naming domains can be possibly cause of confusion. Among the reasons of the confusion:

- the same RMI service name could be associated to instances of different classes in different registries;

- in order to access a remote object, a system programmer must be aware of its RMI absolute address. At best, all the absolute addresses of interest will be maintained by one ore more configuration files. Whenever a new address of interest is added to a RMI registry or already existing addresses are updated or cancelled, the configuration files have to be manually updated. If the number of the configuration files or the number of addresses they contain is high, this can result in a tedious and time consuming activity;

- there is no way to understand, in presence of different instances of the same class executed by different hosts, which is the best remote service according to some evaluation criterion. For example, few services could have generated a great numbers of remote objects, while other services could be idle.

N-JMR tries to give an answer to the above listed issues placing itself between users and RMI registers, as shown by figure 2.

The main idea underlying N-JMR is that of a registry maintaining the association between a *N-JMR service name*, mainly identifying the service offered by the instances of the same Java remote class, and the RMI absolute addresses of such instances. In this way it centralises the management of multiple RMI registries in a single meta-registry. It is consequently possible for N-JMR to select the best remote object according to some evaluation criterion. This evaluation criterion can be chosen among those provided by the system, or it is possible to define and add new evaluation criteria to the system itself. More in detail, N-JMR and a client interact in the following way:

1. N-JMR is queried by specifying a *N-JMR service name* and by selecting an evaluation criterion (step 1 of figure 2);

2. N-JMR selects and returns the RMI absolute address of the best remote service among all those associated to the specified *N-JMR service name* (step 2 of figure 2) according to the selected evaluation criterion;

3. by this address it is possible to obtain from the proper RMI registry (step 3 of figure 2) the reference to the remote object (step 4 of figure 2) to be accessed (step 5 of figure 2).

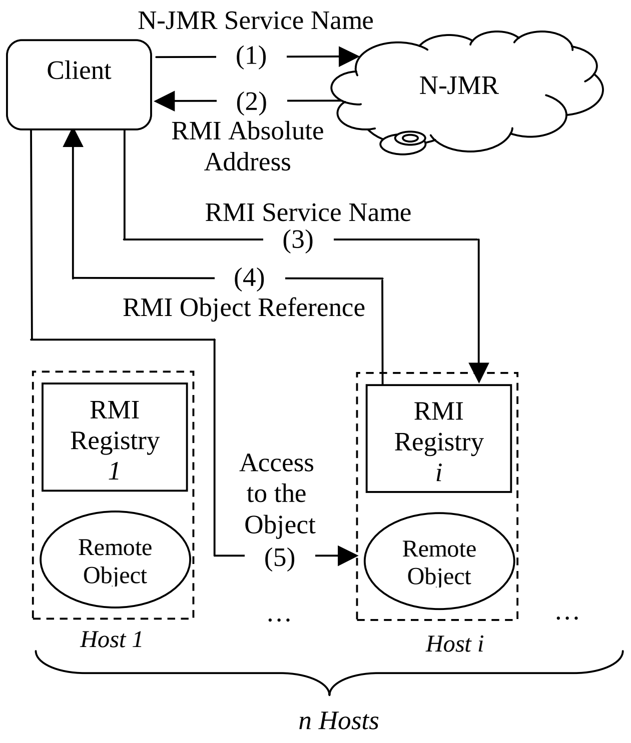

Figure 2

Adopting N-JMR implies at least two advantages, the access to RMI services is:

- *independent from absolute RMI addresses:*

  the RMI absolute address of any remote service of interest has not to be known, all it is needed is the corresponding *N-JMR service*. In this way, the client does not have to concern with the insertion, the updating or the cancellation of RMI absolute addresses from registries;

- *independent from instances:*

  the client has potential access to any of the remote instances providing a certain service, the best one with respect to some evaluation criterion.

Furthermore N-JMR, by extending the functionality of the RMI naming service by a directory service, makes it possible to associate to a N-JMR service name any additional information useful to describe the characteristics of the remote class implementing that service.

3. N-JMR'S ARCHITECTURE

The most interesting aspects of the N-JMR architecture are those concerning with the realisation of the meta-registry. N-JMR is based on the following logic components:

- the *Object Address Provider*: is the system interface toward the client application. The inputs are an *N-JMR service name* and, possibly, the identifiers of an *Object Selector* and of the evaluation criterion to be adopted; the output is an absolute RMI address. Figure 3 depicts the main functionality of the logic component with respect to an hypothetical "`Calculator`" remote class, with an *associated N-JMR service name "Calculator"*;

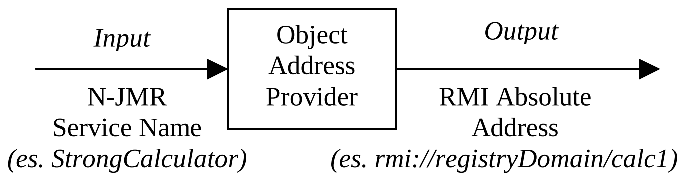

Figure 3

- the *Object Selector*: implements one or more evaluation criteria for the remote objects. The inputs to the component are an absolute RMI remote address and an identifier for the evaluation criterion to be adopted, among those implemented by the *Object Selector* itself. The component is in charge of evaluating the remote services corresponding to the input absolute address with respect to the specified evaluation criterion;

- the *Archive*: maintains all the information required by the system to properly work.

Figure 4 shows the interaction among these logic components:

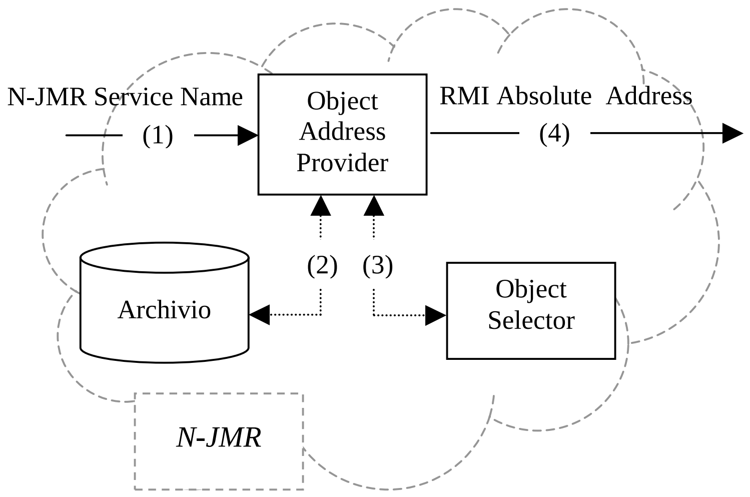

Figure 4

the *Object Address Provider* receives an *N-JMR service name* and, possibly, the identifiers of an *Object Selector* and the evaluation criterion to be adopted (step 1 of figure 4) and retrieves, in the *Archive* content, the RMI absolute addresses of the remote services providing the required service (step 2 of figure 4).

Each of the retrieved addresses is then evaluated by the *Object Selector* according to specified evaluation criterion (step 3 of figure 4).

On the basis of the evaluations returned by the *Object Selector* the *Object Address Provider* returns the *best* absolute RMI address (step 4 of figure 4).

3.1 Objects Selectors Classification

While designing N-JMR we identified two different classes of *Objet Selectors*:

- *Position Indipendent Object Selectors* (*PIOS*): all their evaluations are not dependent by the location on the network of the host executing the *Object Selector*.

- *Position Dipendent Object Selector* (*PDOS)*: otherwise.

In section 5 an example of PDOS will be presented.

3.2 Archive Management

As already stated the *Archive* maintains all the information required by the system to properly work.

Such information must be updated each time an *N-JMR service name* is added, cancelled or updated. This can be achieved in two different ways. The first one relies on a web application especially conceived for the *Archive* management. The web application is in charge of authenticating the user, of collecting all the information required to register the service by a HTML form, of checking for their integrity and, finally, of updating the *Archive* content. This approach makes it possible to easily integrate already available RMI objects into N-JMR.

An alternative to this approach is given by a logic component that makes it possible to update the *Archive* content by software. This component, the *Archive Manager*, provides to RMI objects an API by which they can simply modify the *Archive* content. The *Archive Manager* substantially implements the same functionality provided by the web application.

4. N-JMR IMPLEMENTATION

The implementation activity has been oriented toward the realization of a prototype of the N-JMR system by which testing its correctness from the functional point of view. All adopted solutions have the aim of privileging code reusability and system scalability.

More in details:

- the *Object Address Provider* has been implemented as an RMI object.

  The most significant method signature of the *Object Address Provider* implementation is the following:

  ```
  public String getObjectAddress (
              String NJMRName,
              String objectSelectorName,
              int criterion)
      throws java.rmi.RemoteException
  ```

  Such signature makes it possible for a client to ask for the absolute RMI address of the *best* remote object providing the service specified by `N-JMRName`, according to the evaluation criterion specified by `criterion`, among those implemented by the *Object Selector* specified by `objectSelectorName`;

- the *Object Selector* objects are RMI objects implementing the `ObjectSelector` Java interface. This Java interface defines the methods which are common to all *Object Selector* components. Thus making it possible for an RMI objects programmer to integrate its own *Object Selector* into N-JMR;

- the *Archive* content is distributed in several Lightweight Directory Access Protocol (LDAP) servers [8]. Thus *distributing* the management of the *Archive* content. Furthermore, the definition of the naming domain of N-JMR is done according to the rules of the LDAP naming domain: the entries are organized in a tree-like structure, the Directory Information Tree (DIT). Entries are arranged within the DIT based on their distinguished name (DN). A DN is a unique name that unambiguously identifies a single entry [9]. Thus any *N-JMR Service Name* coincides with a LDAP DN;

- All the web applications, both those managing the *Archive* content and those searching for N-JMR services documentation, have been implemented through Java Servlets [10] and Java Server Pages [11].

5. N-JMR FUNCTIONAL CORRECTNESS

In order to verify the functional correctness of the prototype an *Object Selector*, based relying on the functionality offered by the Simple Network Management Protocol [12], has been implemented. The *Object Selector* implements two different evaluation criteria.

The first one assigns to an RMI absolute address the average percentage of CPU usage of the host executing the corresponding remote service. Clearly, this evaluation criterion is independent by the location on the network of the host executing the *Object Selector*.
The second one assigns to an RMI absolute address the average time spent by an *Hello* message to reach the host executing the corresponding remote service from the host executing the client software. Obviously this evaluation criterion is dependent by the location on the network of the host executing the *Object Selector*. Thus, the implemented *Object Selector* is classified as *PDOS*.

It is worth noticing that this *Object Selector* has been developed as an example, there was no intention to implement an effective selection logic. Nevertheless it can be considered a model to be followed by RMI remote objects programmers.

5. CONCLUSION AND FURTHER WORKS

Netlab, the network and multimedia laboratory of IASI, started a research and experimentation activity with the aim of defining the architecture of, designing and implementing a meta-registry system for the access and the simple management of remote objects and their documentation. Such a system should realize a single access point to services, thus making it possible to locate the best remote object according to some evaluation criteria.

This paper describes the architecture of N-JMR, a meta-registry system filling some gaps of the RMI technology gap, when adopted for distributed systems characterised by the presence of high numbers of RMI registries.

In particular, N-JMR effectively deals with the scalability problems affecting the RMI technology with respect to this particular context, by centralising the management of RMI absolute address inside just one distributed archive. Furthermore, N-JMR makes it possible to select among all the registered remote objects implementing a service of interest, the *best* one with respect to an evaluation criteria provided by an *Object Selector*. The *Object Selector* and the relative evaluation criterion can be selected among those provided by the system, or can be defined, implemented by the RMI object programmer and simply integrated in N-JMR.

Furthermore N-JMR extends the functionality of the RMI naming service by a directory service, thus making it possible to associate any additional information to the registered *N-JMR services name*. Such information can be accessed both at run-time by client application and by programmers of distributed software through web applications searching for available remote services.

N-JMR has been tested from a functional correctness point of view. For such test *Object Selector* based on SNMP has been implemented.

Possibly further works are:

- the integration in N-JMR of RMI/IIOP, EJB, CORBA and DCOM technologies. In this way it will possible associate remote service implemented in different technologies to a N-JMR Service Name. As a consequence a client will also be able to search for technologically different implementations of the same remote service;

- the study and definition of Object Selectors of common interest;

- an evaluation of the N-JMR impact onto the performances of the distributed applications.

REFERENCES

[1] G. Coulouris, J. Dollimore, T. Kindberg, *Distributed Systems Concepts and Design* (2°ed., Addison Wesley)

[2] *The Official CORBA Site* (http://www.corba-org)

[3] *The Official DCOM Site* (http://www.microsoft.com/com/tech/dcom.asp)

[4] *Enterprise JavaBeans Specification, Version 2.0* (http://java.sun.com/products/ejb/)

[5] *Java RMI Specification* (Sun Microsystems Inc., 1999, http://java.sun.com/rmi/)

[6] *RMI-IIOP Programmer's Guide* (Sun Microsystems Inc., 1999, http://java.sun.com/products/rmi-iiop/index.html)

[7] *Fundamentals of RMI (*Short Course by jGuru, 1999 http://java.sun.com/rmi/)

[8] *LightWeight Directory Access Protocol (v2)* (RFC 1777, http://www.ietf.org/rfc/rfc1777.txt)

[9] H.Johner, L.Brown, F.S.Hinner, W.Reis, J.Westman, *Understanding LDAP* (http://www.redbooks.ibm.com)

[10] *Servlet Specifications* (Sun Microsystems Inc., 1999 http://java.sun.com/products/servlet/index.html)

[11] *Java Server Pages Specifications* (Sun Microsystems Inc., 1999 http://java.sun.com/products/jsp/index.html)

[12] *A Simple Network Management Protocol* (RFC 1157, http://www.ietf.org/rfc/rfc1157.txt)