Bridging between CORBA and Jini

J. D. Newmarch, Distributed Systems Technology Centre, Brisbane, and University of Canberra, Canberra, Australia, jan@ise.canberra.edu.au

Abstract

CORBA and Jini are two distributed system architectures. In order for services in one system to be used by another, there must be a bridge between the two. This paper discusses the architectural choices (or design patterns) for bridges between the two types of services.

Keywords: Jini, CORBA, bridge, design patterns

1. Introduction

CORBA is a well-established distributed system architecture which supports communicating objects specified in a language-independent manner by an IDL, with possible implementations in a large variety of languages [3]. CORBA supplies a ‘‘backplane’ Object Request Broker (ORB) whereby these objects can locate and communicate with each other. The requirements of language independence place certain limitations on the mechanisms that can be used in implementation.

Jini is a more recent distributed system technology that is language-centric, based on Java [4]. Jini services must be written in pure Java, and this allows the Jini architecture to exploit certain features of Java. In particular, Jini makes heavy use of mobile Java code, moving ‘‘live’’ objects from one Java Virtual machine (JVM) on a server, to other JVM’s on clients.

Jini servers are restricted to exporting Java objects that do not depend on specific system features. However, they are often ‘‘proxy’’ objects, and the objects they respresent need not be restricted to the pure Java subset. For example, a Jini service may control a piece of hardware, using the Java Native Interface (JNI). The exported Jini proxy does not need to use JNI, but just needs to know how to communicate back to its server which can make non-Java calls.

This paper considers the issue of making CORBA services available to Jini clients as if they were Jini services. This involves building ‘‘bridges’’ between the two types of services. This may be done in a variety of ways, and leads to a set of architectural or design patterns [5]. These patterns are discussed in the following sections. This paper is based on a chapter in a forthcoming book by the author on Jini.

2. Transport Protocols

Most CORBA implementations use IIOP. This is a TCP protocol, and this means that CORBA clients and services may be distributed, located and accessed using TCP mechanisms.

Jini is also meant to be independent of the underlying protocol, requiring only certain transport properties. The only current implementation also uses TCP (with UDP multicast in service discovery). This means that current implementations of CORBA and Jini will most likely both use TCP. This confluence of transport methods allows additional flexibility in bridging architectures that would not be possible across different transport protocols.

3. CORBA IDL

IDL is a language that allows the programmer to specify the interfaces of a distributed object system [3]. The syntax is similar to C++, but does not include any implementation-level constructs. It allows definitions of data types (such as structures and unions), constants, enumerated types, exceptions and interfaces. Within interfaces it allows declaration of attributes and operations (methods).

The book ‘‘Java Programming with CORBA’’ by Andreas Vogel and Keith Duddy [2] contains an example of a room booking service specified in CORBA IDL and implemented in Java. This defines interfaces for `Meeting`, a factory to produce them, `MeetingFactory` and a `Room`. A room may have a number of meetings in slots (hourly throughout the day), and there are support constants, enumerations and typedefs to support this. In addition, exceptions may be thrown under various error conditions. The IDL that follows just shows the interfaces defined. For more details see [1] or [2].

```
module RoomBooking {
    interface Meeting {
        ...
    };

    interface MeetingFactory {
        ...
    };

    interface Room {
        // A Room provides operations to view, make and cancel bookings.
        ...
    };
};
```

4. CORBA Architecture Choices

There are many possible ways of implementing a set of CORBA objects, running from CORBA servers. These are by now well-known in the CORBA community and are discussed in a number of books. We shall only take one choice for this example, since it displays sufficient complexity to allow for the range of Jini choices.

The implementation of the room booking problem in the Vogel and Duddy book runs

1. each room as a separate CORBA object, each with its own server.
2. A meeting factory creates meeting objects which are kept within the factory server, and passed around by reference.

So for a distributed application with ten rooms, there will be eleven CORBA servers running in this design.

5. Jini Architectural Choices

There are several possible ways of bringing this set of objects into the Jini world so that they are accessible to a Jini client. These design pattersn include:

1. A Jini server may exist for each CORBA server
    1. Each Jini server may export fat proxies, which build CORBA references in the same Jini client [see ‘‘many fat proxies’’]
    2. Each Jini server may export a thin proxy, with a CORBA reference held in each of these servers [see ‘‘many thin proxies’’]
2. A single Jini server may be built for the federation of CORBA objects
    1. The single Jini server exports a fat proxy, which builds CORBA references in the Jini client [see ‘‘single fat proxy’’]
    2. The single Jini server exports a thin proxy, with all CORBA references within this single server [see ‘‘single thin proxy’’]

5.1 Many fat proxies

There can be one Jini server for each of the CORBA servers. The Jini servers can be running on the same machines as the CORBA ones, but there is no necessity to do so. Applet security restrictions, for example, may mean that all the Jini servers are running on a single machine, the same one as an applet’s HTTP server.

The Jini proxy objects exported by each Jini server may be fat ones, which connect directly to the CORBA server. Thus each proxy becomes a CORBA client. Within the Jini client, we do not just have one proxy, but many proxies. Because they are all running within the same address space, they can share CORBA references - there is no need to package a CORBA reference as a portable Jini object. In addition, the Jini client can just use all of these CORBA references directly, as instance objects of interfaces. Diagramatically, this appears in Figure 1.

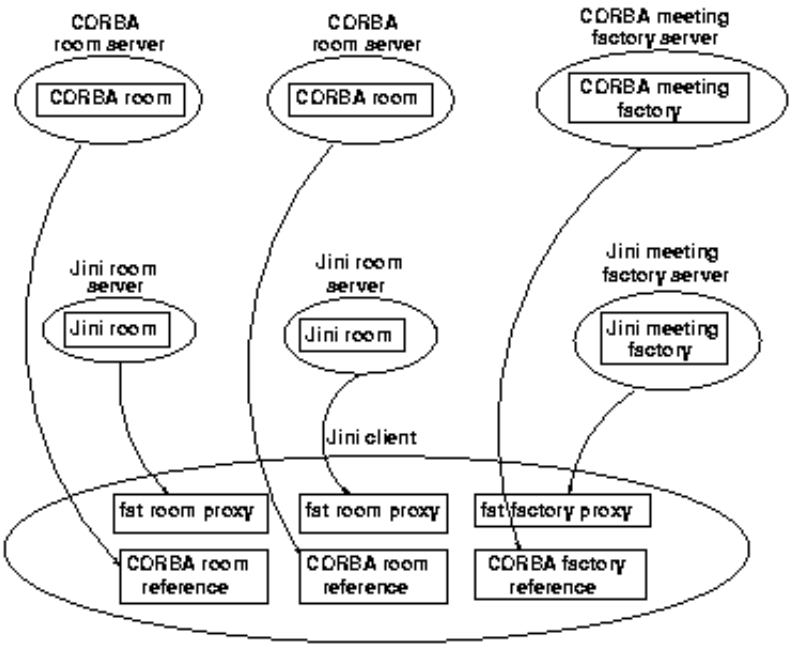

Figure 1: Many Fat Proxies

The CORBA servers are all accessed from within the Jini client. This may rule out this arrangement if the client is an applet, and the servers are on different machines.

5.2 Many thin proxies

The proxies exported can be thin, such as RMI stubs. In this case each Jini server is acting as a CORBA client. This is shown in Figure 2.

If all the Jini servers are collocated on the same machine, then this becomes a possible architecture suitable for applets. The downside of this approach is that all the CORBA references are within different JVMs. In order to move the reference for a meeting from the Jini meeting factory to one of the Jini rooms, it may be necessary to wrap it in a portable Jini object which contains a ''stringified'' version of the CORBA reference. The Jini client will also need to get information about the CORBA objects, which can be gained from these portable Jini objects.

5.3 Single fat proxy

An alternative to Jini servers for each CORBA server is to have a single Jini server into the CORBA federation. This can be a feasible alternative when the set of CORBA objects form a complete system or module, and it makes sense to treat them as a unit. There are then the choices again of where to locate the CORBA references, either in the Jini server or in a proxy. Placing them in a fat proxy looks like Figure 3.

5.4 Single thin proxy

Placing all the CORBA references on the server side of a Jini service means that a Jini client only needs to make one network connection to the service. This is probably the best option from a security viewpoint of a Jini client. This is shown in Figure 4.

6. Exceptions

CORBA methods can throw exceptions of two types: system exceptions and user exceptions. In Java, system exceptions subclass from `RuntimeException`, and so are *unchecked*. These do not need to have explicit `try...catch` clauses around them. If an exception is thrown, it will be caught by the Java runtime, and generally halt the process with an error message. This would result in a CORBA client dying, which would generally be

undesirable. Many of these system exceptions will be caused by the distributed nature of CORBA objects, and probably should be caught explicitly. If they cannot be handled directly, then to bring them into line with the Jini world, they can be wrapped as ‘‘nested exceptions’’ within a `Remote` exception and thrown again.

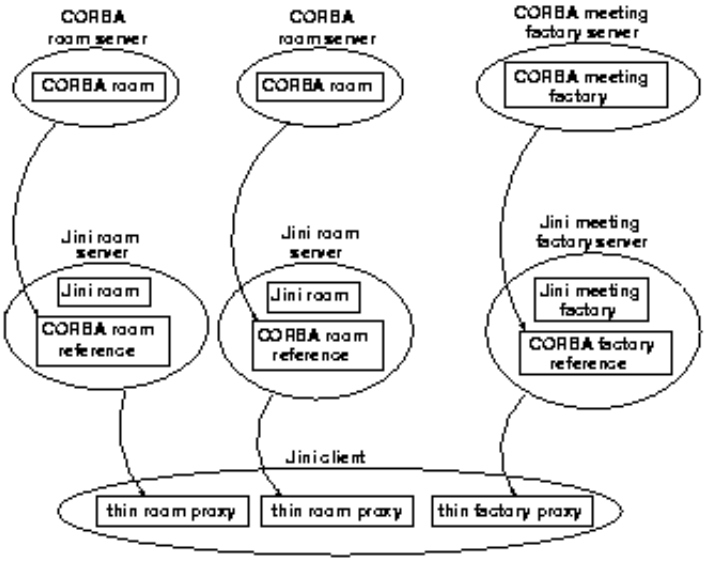

Figure 2: Many Thin Proxies

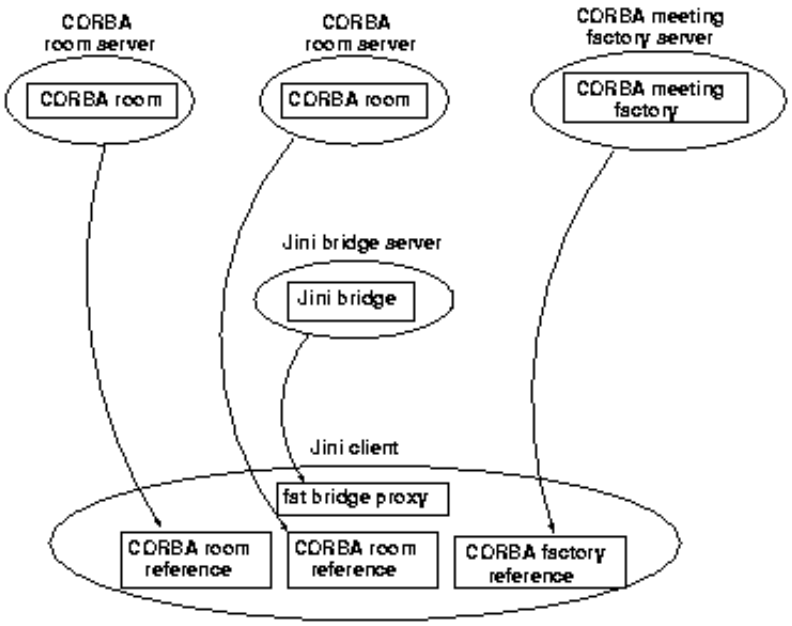

Figure 3: Single Fat Proxy

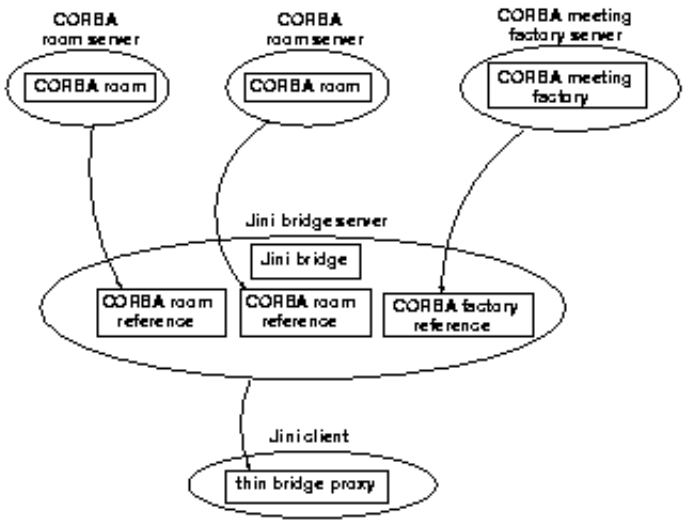

Figure 4: Single Thin proxy

User exceptions are declared in the IDL for the CORBA interfaces and methods. These exceptions are *checked*, and need to be explicitly caught (or re-thrown) by Java methods. If a user exception is thrown, this will be because of some semantic error within one of the objects, and will be unrelated to any networking or remote issues. They should be treated as they are, without wrapping them in `Remote` exceptions.

7. Conclusion

This paper has shown how CORBA services can be made visible to Jini clients as if they were Jini services. This allows existing CORBA services to be used by both CORBA and Jini clients. It has not discussed how to make Jini services appear to CORBA clients, because there are not that many of them yet. It has not shown code implementations due to space reasons, but these can be found in [1].

References

[1] J. Newmarch, *Guide to Jini Technologies*, APress, 2000, http://pandonia.canberra.edu.au/java/jini/Jini.xml
[2] A. Vogel and K. Duddy, *Java Programming with CORBA*, Wiley, 1999, http://www.wiley.com/compbooks/vogel
[3] Object Management Group, *CORBA Specification*http://www.omg.org
[4] K. Arnold et al, *The Jini Specification*, Addison-Wesley, 1999
[5] E. Gamma et al, *Design Patterns: Elements of Reusable Object Software*, Addison-Wesley, 1995