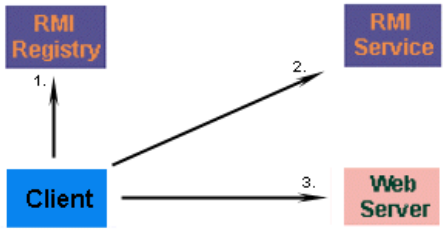
Java RMI & CORBA
A comparison of two competing technologies

With the introduction of CORBA support to Java (as of version 1.2), developers now face the question of whether to continue to use remote method invocation (RMI), or make a move to CORBA. The choice is made more difficult if your applications are already written in RMI - considerable effort might have to be made to convert to CORBA. Is it worth the move? This article discusses the pros and cons, and evaluates the potential of these two technologies.

What is Remote Method Invocation?

Remote method invocation allows Java developers to invoke object methods, and have them execute on remote Java Virtual Machines (JVMs). Under RMI, entire objects can be passed and returned as parameters, unlike many remote procedure call based mechanisms which require parameters to be either primitive data types, or structures composed of primitive data types. That means that any Java object can be passed as a parameter - even new objects whose class has never been encountered before by the remote virtual machine.

This is an exciting property, because it means that new code can be sent across a network and dynamically loaded at run-time by foreign virtual machines. Java developers have a greater freedom when designing distributed systems, and the ability to send and receive new classes is an incredible advantage. Developers don't have to work within a fixed codebase - they can submit new classes to foreign virtual machines and have them perform different tasks. When working with remote services, RMI clients can access new versions of Java services as they are made available - there's no need to distribute code to all the clients that might wish to connect. While code can be accessed from a local or remote file-system, it can also be accessed via a web server, making distribution easier. RMI also supports a registry, which allows clients to perform lookups for a particular service. The following diagram shows the interaction between different components of an RMI system. Clients that know about a service can look up its location from a registry and access the service. If a new class is required, it can be downloaded from a web server.



Client connects to a registry server, accesses a RMI service, and downloads new code as required from a web server.

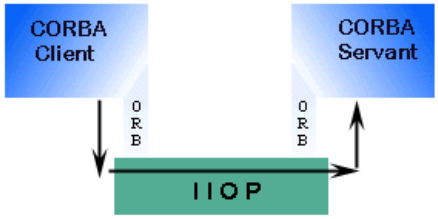
Remote method invocation has a lot of potential, from remote processing and load sharing of CPU's to transport mechanisms for higher level tasks, such as mobile agents which execute on remote machines (Reilly, 1998). Because of the flexibility of remote method invocation, it has become an important tool for Java developers when writing distributed systems. However, there are many legacy systems written in C/C++, Ada, Fortran, Cobol, and other exotic languages. If legacy systems need to interface with your RMI systems, or your RMI systems need to interface with them, problems can occur. RMI is Java specific, and you'll need to write a bridge between older systems. Additionally, if you or your company plans on using other languages in the future, you may also find yourself in a bind because of RMI's tie to Java - one day Java itself may become a legacy platform. Writing interfaces to legacy systems isn't my idea of fun programming!

What is CORBA?

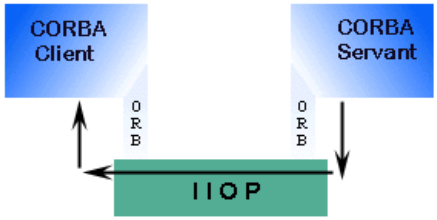
Common Object Request Broker Architecture (CORBA) is a competing distributed systems technology that offers greater portability than remote method invocation. Unlike RMI, CORBA isn't tied to one language, and as such, can integrate with legacy systems of the past written in older languages, as well as future languages that include support for CORBA. CORBA isn't tied to a single platform (a property shared by RMI), and shows great potential for use in the future. That said, for Java developers, CORBA offers less flexibility, because it doesn't allow executable code to be sent to remote systems.

CORBA services are described by an interface, written in the Interface Definition Language (IDL). IDL mappings to most popular languages are available, and mappings can be written for languages written in the future that require CORBA support. CORBA allows objects to make requests of remote objects (invoking methods), and allows data to be passed between two remote systems. Remote method invocation, on the other hand, allows Java objects to be passed and returned as parameters. This allows new classes to be passed across virtual machines for execution (mobile code). CORBA only allows primitive data types, and structures to be passed - not actual code.

Under communication between CORBA clients and CORBA services, method calls are passed to Object Request Brokers (ORBs). These ORBs communicate via the Internet Inter-ORB Protocol (IIOP). IIOP transactions can take place over TCP streams, or via other protocols (such as HTTP), in the event that a client or server is behind a firewall. The following diagram shows a client and a servant communicating.



CORBA client sends a request through its local ORB to a remote ORB's servant



CORBA servant sends back a response to a remote ORB

RMI vs CORBA

Comparing RMI and CORBA doesn't reveal an optimum solution - one is not "better" than the other. The properties of these two technologies lend themselves to different situations. A comparison of RMI and CORBA helps to highlight individual strengths and weaknesses, but the applicability of one technology over the other depends largely on the purposes for which it is to be used, the experience of the developers who will design, implement and maintain the distributed system, and whether non-Java systems are intended to access the system now or in the future.

RMI pros and cons

Remote method invocation has significant features that CORBA doesn't possess - most notably the ability to send new objects (code and data) across a network, and for foreign virtual machines to seamlessly handle the new objects (Reilly, D). Remote method invocation has been available since JDK 1.02, and so many developers are familiar with the way this technology works, and organizations may already have systems using RMI. Its chief limitation, however, is that it is limited to Java Virtual Machines, and cannot interface with other languages.

Remote method invocation

Pros	Cons
Portable across many platforms	Tied only to platforms with Java support
Can introduce new code to foreign JVMs	Security threats with remote code execution, and limitations on functionality enforced by security restrictions
Java developers may already have experience with RMI (available since JDK1.02)	Learning curve for developers that have no RMI experience is comparable with CORBA
Existing systems may already use RMI - the cost and time to convert to a new technology may be prohibitive	Can only operate with Java systems - no support for legacy systems written in C++, Ada, Fortran, Cobol, and others (including future languages).

CORBA pros and cons

CORBA is gaining strong support from developers, because of its ease of use, functionality, and portability across language and platform (Reilly, D). CORBA is particularly important in large organizations, where many systems must interact with each other, and legacy systems can't yet be retired. CORBA provides the connection between one language and platform and another - its only limitation is that a language must have a CORBA implementation written for it. CORBA also appears to have a performance increase over RMI, which makes it an attractive option for systems that are accessed by users who require real-time interaction (Morgan, 1997).

Common Object Request Broker Architecture

Pros	Cons
Services can be written in many different languages, executed on many different platforms, and accessed by any language with an interface definition language (IDL) mapping.	Describing services require the use of an interface definition language (IDL) which must be learned. Implementing or using services require an IDL mapping to your required language - writing one for a language that isn't supported would take a large amount of work.
With IDL, the interface is clearly separated from implementation, and developers can create different implementations based on the same interface.	IDL to language mapping tools create code stubs based on the interface - some tools may not integrate new changes with existing code.
CORBA supports primitive data types, and a wide range of data structures, as parameters	CORBA does not support the transfer of objects, or code.
CORBA is ideally suited to use with legacy systems, and to ensure that applications written now will be accessible in the future.	The future is uncertain - if CORBA fails to achieve sufficient adoption by industry, then CORBA implementations become the legacy systems.
CORBA is an easy way to link objects and systems together	Some training is still required, and CORBA specifications are still in a state of flux.
CORBA systems may offer greater performance	Not all classes of applications need real-time performance, and speed may be traded off against ease of use for pure Java systems.

Summary

An examination of these two technologies shows that, while they do overlap in functionality to some degree, they each possess strengths that outshine the other for particular tasks. A careful evaluation of the intended use of RMI or CORBA is required, to determine which technology is the most appropriate. There aren't any hard and fast rules that can be applied, and there is no clear victor in the battle for the minds and hearts of developers. Time will tell which technology (if any) becomes the more dominant, and in the immediate future both will continue to play a role.