DIF8901 Object-Oriented Systems

A Comparison of Distributed Object Technologies

Carl-Fredrik Sørensen

The Norwegian University of Science and Technology

Abstract

This essay compares the popular distributed object models/middleware standards; CORBA, DCOM and Java/RMI. These models are to a certain degree competing and have strengths and weaknesses with respect to portability, programming language and hardware support, ease of use and availability. The models are all promoting reuse through component interfaces and distributed object services. All three standards have been extended with component models.

1 Introduction

[CDK01] defines a distributed system as a system in which hardware or software components located at networked computers communicate and co-ordinate their actions only by passing messages. Computing devices may be connected to a wide range of networks as for instance the Internet, mobile phone networks, corporate networks, home networks or to combinations of these. Modern program systems like Internet-based and enterprise applications offer multitiered, component-based architectures that incorporate middleware for distributing components across heterogeneous platforms. The platforms range from mobile devices like personal digital assistants (PDA), laptops and mobile phones; ubiquitous devices like televisions, refrigerators and cars; to different types of computers like mainframes and PCs.

The three most dominating distributed object technologies or middleware are CORBA, DCOM and Java/RMI. These are extensions of traditional object-oriented systems by allowing objects to be distributed across a heterogeneous network. The objects may reside in their own address space outside of an application or on a different computer than the application and still be referenced as being part of the application.

The history and evolution of each of the distributed object technologies are quite different and addresses different requirements for multiple platforms, languages or network protocols.

CORBA – Common Object Request Broker Architecture, is an open distributed object computing infrastructure standardised by the Object Management Group (OMG) and is a specification based on technologies proposed (and partly provided) by the software industry [CORBA]. CORBA is the most used middleware standard in the non-Windows market. The OMG was founded in 1989 to promote the adoption of object-oriented technology and reusable software components.

DCOM – Distributed Component Object Model, is a standard developed by Microsoft and it is a distributed extension of the COM standard [COM]. The COM standard is based on the development of compound document technology to integrate document parts (for instance spread-sheets, pictures, presentations, word processors etc.) created by different Windows applications. COM is the world most widely used component software model and dominates the desktop market.

Java/RMI – Java/Remote Method Invocation, is a standard developed by JavaSoft. Java has grown from a programming language to three basic and completely compatible platforms; J2SE (Java 2 Standard Edition), J2EE (Java 2 Enterprise Edition) and J2ME (Java 2 Micro Edition). J2SE is the foundational programming language and tool-set for coding and component development. J2EE supplements J2SE and is a set of technologies and components for enterprise and Internet development. J2ME is used for creating software for embedded, mobile, consumer and other small devices like Personal Digital Assistants (PDA) and mobile phones.

All three standards have their component model extensions, CORBA with the CORBA Component Model (CCM), COM with the Microsoft Transaction Server (COM+/MTS), and Java/RMI with the JavaBean and Enterprise JavaBeans (EJB) component models. Microsoft is about to deliver a new application development framework called .NET. It is not certain whether .NET will be supported on multiple platforms.

This essay will not provide any code examples in the descriptions and comparisons of the different technologies. Code examples can be found in e.g. [Chung] and [Raj].

2 An Overview of Distributed Object Technologies

This section will give a short technological overview of the different distributed object technologies compared in this essay.

All three distributed object technologies are based on a client/server approach implemented as network calls operating at the level of bits and bytes transported on network protocols like TCP/IP. In order to avoid the hard and error prone implementation of network calls directly in the client and server {objects}, the distributed technology standards address the complex networking interactions by abstract and hide the networking issues and instead let the programmer concentrate on programming the business logic.

The basic idea behind network abstractions like RPC (Remote Procedure Call) is to replace the local (server) and remote (client) end by stubs. This makes it possible for both client and server to strictly use local calling conventions and thereby be unaware of calling a remote implementation or being called remotely. To accomplish this the client call is handled by a client stub (proxy) that marshals the parameters and sends them by invoking a wire protocol like IIOP, ORPC or JRMP, to the remote end where another stub receives the parameters, unmarshals and calls the true server. The marshaling and unmarshaling actions are responsible for converting data values from their local representation to a network format and on to the remote representation [Szyperski98]. Format differences like byte ordering and number representations are bridged this way.
The object services offered by all three approaches are defined through interfaces. The interface is for all defined as a collection of named operations, each with a defined signature and optionally a return type. The interface serves as a contract between the server and client.

2.1 CORBA

This section briefly explores the components defined in the OMG Reference Model Architecture (OMA) as shown in Figure 1 [Schmidt].
Object Services are domain-independent interfaces that are used by many distributed object programs. OMG has specified object services like the naming service that allows clients to find objects based on names.
There are also specifications for lifecycle management, security, transactions, event notification, persistence and object query etc. [OMG]. Vendors have however abandoned some of the services, usually because they were impractical or ineffective.
Common Facilities are oriented towards end-user applications. An example of a common facility is the *Distributed Document Component Facility* (DDCF). DDCF allows for the presentation and interchange of objects based on a document model, for instance facilitating the linking of a spreadsheet object into a text document. The common facilities have mostly been abandoned or taken over by the new vertical industry domain task forces.
Domain Interfaces fill roles similar to Object Services and Common Facilities but are oriented towards specific application domains like manufacturing, telecommunication etc.
Application Interfaces are interfaces developed specifically for a given application.

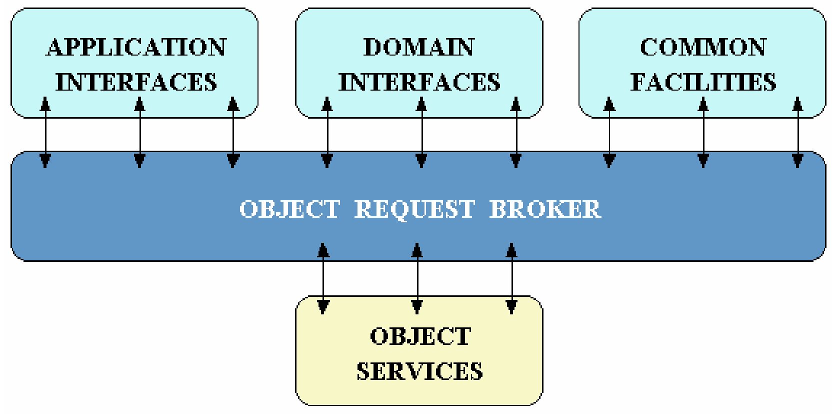

Figure 1. OMG Reference Model Architecture [Vinoski97]

The **object request broker** (ORB) is the core of the CORBA architecture. The ORB is a central object bus where the CORBA objects interact transparently with other CORBA objects located either locally or remotely [Vinoski97]. It provides basic messaging, communication, directory, security and location transparency and insulates applications from details about hardware, networking and system. The ORB simplifies distributed programming by decoupling the client from the details of the method invocations. This makes client requests appear to be local procedure calls. Figure 2 shows the CORBA ORB Architecture.

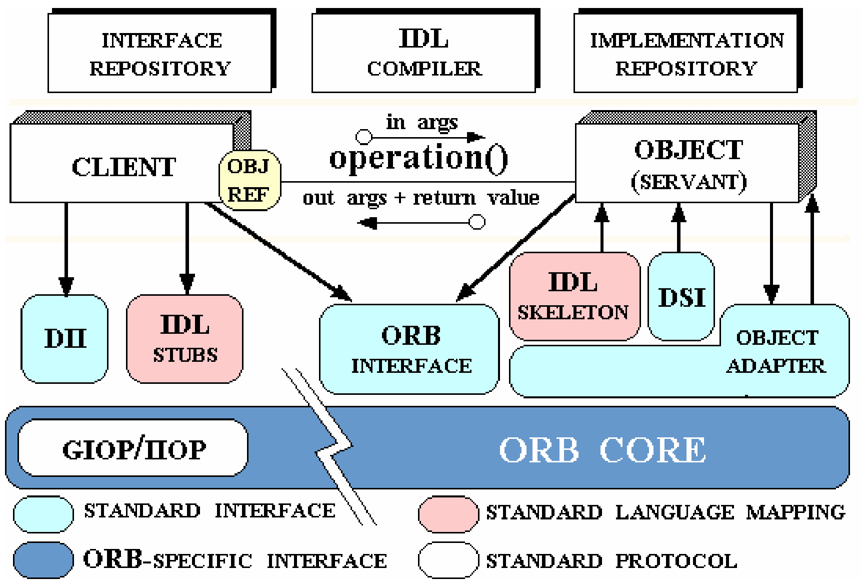

Figure 2. CORBA ORB Architecture [SCHMIDT]

The **Internet Inter-ORB Protocol** (IIOP) was developed in the CORBA 2.0 specification as a protocol for communication between ORBs from different vendors. IIOP is a specialisation of the General Inter-ORB Protocol (GIOP) and runs on top of TCP/IP. CORBA relies on IIOP or other specialisations of GIOP for remoting objects. The CORBA object is a programming entity that consists of an *identity*, an *interface*, and an *implementation*, which is known as a *Servant*. The servant is an implementation programming language entity that defines the operations that support a CORBA IDL interface [Schmidt].
OMG has defined an interface definition language (IDL) that declares the interfaces and methods of a CORBA server object. Every CORBA object must be declared in IDL. The IDL compiler creates stubs and skeletons that serve as the “glue" between the client and server applications, respectively, and the ORB. The CORBA IDL syntax is similar to C++, and includes semantics for multiple inheritance through interfaces, and user-defined exceptions.

When a CORBA client requests a service from a CORBA server object, the ORB is responsible for:

- Find the implementation of the server object.
- Prepare the object for requests.
- Give the object reference back to the client object.
- Communicate the requests to the server object.
- Returning the server result back to the client.

The interactions are done either by the ORB interface or through an Object Adapter (Basic or Portable Object Adapter - POA). The **ORB interface** is an abstract interface for an ORB to decouple applications from implementation details. This interface provides various helper functions such as converting object references to strings and vice versa, and creating argument lists for requests made through the dynamic invocation interface (see section 4, Invocation of objects). The **Object Adapter** assists the ORB with delivering requests to the object and with activating the object. The object adapter associates object implementations with the ORB. Object adapters can be specialised to provide support for certain object implementation styles (such as OODB object adapters for persistence and library object adapters for non-remote objects) [Schmidt].
For further information about CORBA, see e.g. [CORBA] [CDK01] [Orfali96] [Orfali97] [Orfali98] [Vinoski97].

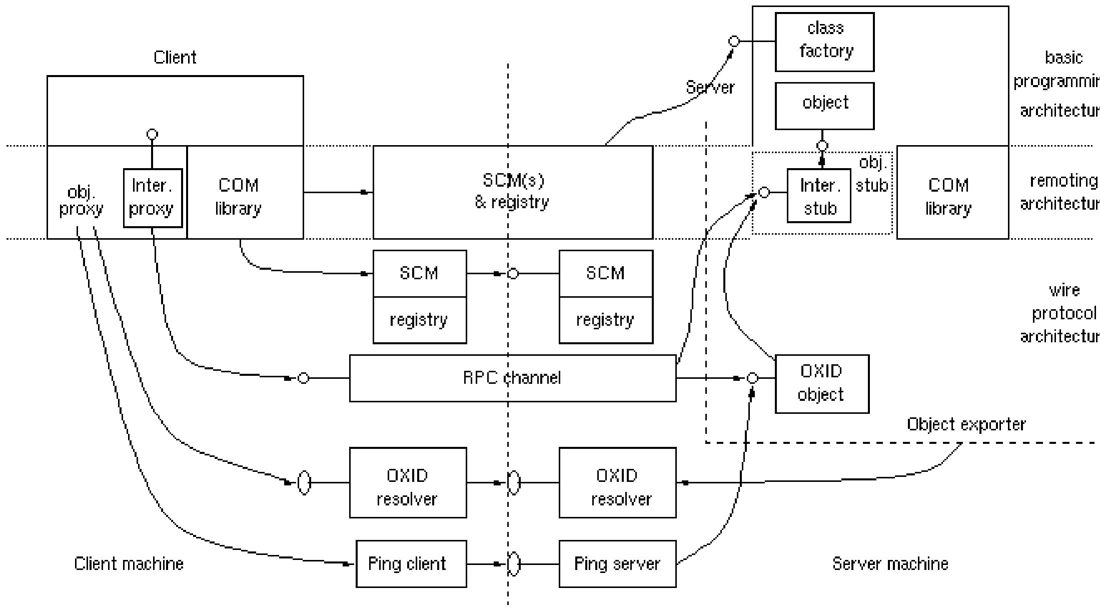

Figure 3: DCOM overall architecture [Chung]

2.2 DCOM

The Distributed Component Object Model (DCOM) supports remote objects by running on a protocol called the Object Remote Procedure Call (ORPC) [Raj]. ORPC is built on top of DCE/RPC and interacts with the run-time services in COM. Figure 3 shows the overall DCOM architecture [Chung].
COM relies on tables of procedure variables (dispatch tables) containing function pointers representing the interfaces. This memory layout conforms to the C++ virtual table layout (vtable) and is used for static invocation of objects. DCOM server objects support multiple interfaces where each represents a different behaviour of the object. A client acquires a pointer to one of the server interfaces and then starts calling the exposed methods of the server through the interface pointer as if the object resided in the client address space.

The Microsoft Interface Definition Language, MIDL, is used to define interfaces for the DCOM (and RPC) objects. MIDL is an extension of DCE (RPC) IDL and is based on C syntax and data types. MIDL supports two different forms of interface descriptions, the basic COM interface `IUnknown`, and the OLE automation interface, `IDispatch`. Every (D)COM object must implement the `IUnknown` interface. The `IDispatch` interface extends the `IUnknown` interface and is a sort of gateway interface to many more interfaces [Grimes97]. MIDL also associates class(es) to

the interfaces. `IUnknown` provides a standard `QueryInterface()` method to navigate among the interfaces. This also introduces the notion of an object proxy/stub dynamically loading multiple interface proxies/stubs in the remoting layer. In addition the `IUnknown` interface includes two methods for reference counting: `AddRef()` and `Release()`.
The MIDL compiler creates **type libraries** (.tlb) that stores the type information of the objects. The type library can be used to dynamically invoke objects implementing the `IDispatch` interface. An object in COM is allowed to implement **dual interfaces**, where an interface is implemented both through `IDispatch` and through the vtable.
A *Unique Universally Unique Identifier* (UUID) uniquely identifies every class (CLSID) and interface (IID) in COM. MIDL does not specify user-defined exceptions. DCOM instead provides standard COM exceptions returned by a standard return code `HRESULT`. The server objects are made available through the system registry by registering the server proxy.
The COM specification is at the binary level. This allows components to be written in a diversity of programming languages. The hardware platform must support COM services in order to provide DCOM.
For further information about COM and DCOM, see e.g. [COM] [Eddon98] [Grimes97] [Pinnock98].

2.3 Java/RMI

RMI supports remote objects by running on a protocol called the Java Remote Method Protocol (JRMP). Object serialisation is heavily used to marshal and unmarshal objects as streams. Both client and server have to be written in Java to be able to use the object serialisation. The Java server object defines interfaces that can be used to access the object outside the current Java virtual machine (JVM) from another JVM on for instance a different machine. A RMI registry on the server machine holds information of the available server objects and provides a naming service for RMI. A client acquires a server object reference through the RMI registry on the server and invokes methods on the server object as if the object resided in the client address space. The server objects are named using URLs and the client acquires the server object reference by specifying the URL.

When a Java/RMI client requests a service from the Java/RMI server, it does the following [Java/RMI]:

- initiates a connection with the remote JVM containing the remote object,
- marshals the parameters to the remote JVM,
- waits for the result of the method invocation,
- unmarshals the return value or exception returned, and
- returns the value to the caller.

By using serialisation of the objects, both data and code can be passed between a server and a client – this allows different instances of an object to run on both client and server machines. To insure that code is downloaded or uploaded safely, RMI provides extra security.
To declare remote access to server objects in Java, every server object must implement the `java.rmi.Remote` interface. `java.rmi.server.RemoteObject` and its subclasses, `java.rmi.server.RemoteServer`, `java.rmi.server.UnicastRemoteObject` and `java.rmi.activation.Activatable` provide RMI server functions. The class `java.rmi.server.RemoteObject` provides implementations for the `java.lang.Object` methods, `hashCode`, `equals`, and `toString` that are sensible for remote objects. The classes `UnicastRemoteObject` and `Activatable` provide the methods needed to create remote objects and make them available to remote clients. The subclasses identify the semantics of the remote reference, for example whether the server is a simple remote object or is an activatable remote object (one that executes when invoked) [Java/RMI]. The `java.rmi.server.UnicastRemoteObject` class defines a singleton (unicast) remote object whose references are valid only while the server process is alive. The class `java.rmi.activation.Activatable` is an abstract class that defines an *activatable* remote object that starts executing when its remote methods are invoked and can shut itself down when necessary [Java/RMI].
Java/RMI can be used on a diversity of platforms and operating systems as long as there is a JVM implementation on the platform. For further information about Java/RMI and distributed computing in Java, see e.g. [Java/RMI] [Mahmoud00] [Orfali98].

3 Component Models

This section provides a brief description of the component models that extends the distributed object technologies presented.

3.1 The CORBA Component Model

The CORBA Component Model (CCM) was introduced to meet the problems with the reuse of the often to fine-grained software objects as building blocks to create components and component frameworks. The CORBA objects both on the client and server-side were difficult to assemble into systems. The CCM is a specification of a component framework that represents both the client and server-side of a distributed architecture. The CCM makes it possible to create tools that can put together secure, reliable, efficient, scalable CORBA systems in much less time than before [Welsh01]. The use of (automated) tools will most likely improve both the quality of CORBA applications, and decrease the time and effort to build CORBA applications.
Like EJB components, CCM components will run in a container. The container consists of one or more specialised Portable Object Adapters (POA) with the Persistence, Transaction, Security and Notification services.
CCM supports the same container types as EJB, in addition is a fourth type, Process, part of the specification. Unlike the Entity container type, the Process type does not expose its primary key. The Process type has else the same characteristics as the Entity type.
Components can be developed in two levels: basic and extended. The basic level is used to make ordinary CORBA objects as components without significant changes of the programming model. The extended level component provides a richer set of functionality than the existing CORBA model [Welsh01].
The CCM can be summarised as a multilanguage form of EJB. In that respect the EJB specification can be regarded as a subset of the CCM specification. This means that an EJB deployed into a CCM container automatically will become a CORBA component; Java clients using the EJB programming model can view CORBA components as EJBs; CORBA clients can view EJBs as they were CORBA components.
See the CORBA Web-site (http://www.corba.org/) for further information of the CORBA Component Model.

3.2 COM+/Microsoft Transaction Server (MTS)

COM+ is a significant upgrade of the original COM by adding transaction processing from Microsoft Transaction Server and Transaction Internet Protocol environments to the distributed component model. The model offers automatic load balancing, object pooling, cached database services, central administration, queued components replacing RPC, and a publish-subscribe event service (like Java and CORBA). MTS simplifies both development and deployment of DCOM components and applications. MTS provides a wrapper for COM components that allows objects to be distributed across a network, running under a specified security context and threading model, without requiring extra code to be written to handle security and threading [Grimes97].
Microsoft also presents the Windows DNA, which is a three-tier architecture based on COM+ and MTS and includes Active Service Pages (ASP) as client technology and SQL-server as backend database. To use DNA, both client and server must be DNA-compliant to be able to communicate.

The .NET architecture is intended to replace both COM+ and DNA as the application and Internet development architecture. The COM runtime is replaced by a common-language runtime (CLR) that supports and integrates components developed in any compliant language. Like CORBA, CLR uses an intermediate language compiler that translates between different languages. The .NET is a loosely coupled architecture for distributed applications. The remote access is based on BizTalk XML and the Simple Object Access Protocol (SOAP) technologies. .NET introduces two new languages where C# is an direct competitor to both C++ and Java, and WSDL (Web Services Description Language) that will be used to create remote XML-based interfaces for .NET components transported via SOAP. SOAP can be used to either invoke RPC on another application or to send the objects as an XML message to a remote location using the Internet HTTP-protocol. SOAP consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data-types, and a convention for representing remote procedure calls and responses.
SOAP can potentially be used in combination with a variety of other protocols [SOAP]
See the Microsoft Web-site (http://www.microsoft.com/) for further information of the Microsoft component models and application development frameworks.

3.3 JavaBeans and Enterprise JavaBeans

The JavaBean component model is the Java graphical component model. The Enterprise JavaBean (EJB) component model is a component model for non-graphical Java components used to packaging distributed middleware services. The EJB model encapsulates business logic on the server side and consists of two primary elements; the Java object or component linked together as an EJB bean, and the EJB container. The container mediates between the bean and its environment. The container provides middleware services like concurrency, persistence, security and multithreading support. The container also provides communication between the environment and the bean and management of the bean components. EJB supports three container types: Entity, Session and Service. EJB is part of J2EE, which includes both Java/RMI and Java-IIOP as middleware options. The use of Java-IIOP guarantees interoperability with CORBA servers written in almost any supported language. J2EE also includes a mandatory RMI-IIOP bridge that makes J2EE application servers able to talk to both CORBA and RMI clients and servers.

Sun has defined the Sun Open Net Environment (Sun ONE) Software Architecture that is a direct competitor to Microsoft .NET. The Sun ONE software architecture addresses issues like privacy, security, and identity. It defines practices and conventions to support situational context, such as client device type and user location. And it supports systems that can be used on many network types like the traditional Web, the wireless Web and the home network. The architecture is designed to ensure that smart Web services, developed using any tool, running on any platform, can seamlessly interoperate. The Sun ONE software architecture is based on XML, Java technology, and LDAP [SunONE].
See the Sun/Java Web-sites (http://java.sun.com/ and http://www.sun.com) for further information of the JavaBean and EJB component models, and the Sun ONE architecture.

4 Comparison of Distributed Object Technologies

This section compares the different distributed technologies with respect to the different areas as listed below. The areas are selected based on where the models have the biggest difference.

➢ Programming languages

CORBA is a specification and can therefore be used on heterogeneous platforms, operating systems and programming languages as long as there is an ORB implementation for the platform and a language mapping for the programming language. DCOM also supports multiple programming languages since it is a binary standard. Java/RMI is only applicable as middleware between clients and servers implemented in the Java language since it relies heavily on Java object serialisation. Java offers an ORB (Java/IIOP) implementation as part of the Java 2 Enterprise Edition (J2EE) that is quite similar in syntax as RMI and that possibly will replace RMI as the preferred Java middleware. Exposing Java objects directly through an ORB or through the Java-IDL mapping makes it possible to invoke Java objects from non-Java environments.

➢ Development support

The system development support of COM in popular languages like Visual Basic and Java (with Microsoft extension) is very good when using Microsoft Visual Studio as the development environment. Objects can easily be registered as COM server objects without any specific coding. COM clients can import the interfaces to the COM server objects through wrappers (of the client proxies) and then used them as ordinary objects in the language. Development support in the form of Integrated Development Environments (IDE), in both CORBA and Java (including Java/RMI), have been quite poor compared to the Microsoft development tools. Especially the developer user interface and debugging facilities have been poorer [MCP98]. This situation is however improving with support of design and development patterns (for instance CORBA design patterns, JavaBeans and EJB patterns) for the different technologies (see for instance http://www.togethersoft.com/), and debugging of remote objects. Especially the Java tools have became mature and supportive in almost all kind of application development. The introduction of the CCM will probably make the development of CORBA objects much easier.

➢ Interface definition

Both DCOM and CORBA offer an interface definition language (IDL) to describe the interfaces for their respective objects. For both cases, the IDL is language neutral and is used to define mappings between different programming languages. It is however some differences in both syntax and semantics between the interface notions. Java/RMI

does not have an own language for interface description but have defined interface declarations as a separate concept in the language and the interfaces are stored as .java files. Java objects that implement the `java.rmi.Remote` interface are accessible by remote Java clients.
The Microsoft Interface Definition Language (MIDL) is not like the CORBA IDL specified by a formal notation (BNF).

➢ Object-oriented support

DCOM and Java/RMI support multiple interfaces for an object (or component), each representing a different view or behaviour of the object. Such a concept did not exist in the CORBA 2.0 specification [Chung], but is included as part of the CORBA 3.0 specification. Multiple interfaces in CORBA are closely associated to the CORBA Component Model (see section 3.1). The mechanism specified is much like the `IUnknown` interface in DCOM, but returns a list of interfaces, whereas the `QueryInterface()` method has to be interrogated separately for each interface [Welsh01].

Java/RMI and CORBA support multiple inheritance at the interface level where DCOM only supports single inheritance[1]. Every CORBA interface inherits from `CORBA.Object`. The constructor to `CORBA.Object` implicitly performs common tasks such as object registration, object reference generation, skeleton instantiation, etc. In DCOM, such tasks are either explicitly performed by the server programs or handled dynamically by the DCOM run-time system [Chung]. In Java/RMI, the tasks are performed by the `java.rmi.server.UnicastRemoteObject.`

A DCOM server object can create several object instances of multiple DCOM object classes depending on the number of interfaces being used (for instance through COM Aggregation or containment). A CORBA or Java/RMI object reference is served by one server object instance. This object instance can however represent many other object instances.
MIDL specifies a class for one or more interfaces, where CORBA IDL only specifies interfaces.

COM provides many of the same services as CORBA and Java, but may be looked upon as a component rather than an object distribution technology due to the missing support of multiple inheritance. The implication of the missing support for inheritance is that COM components can not be modified to create new components. It does however allow some flexible forms of programming [Raj]. COM Aggregation or COM containment may be used to combine COM components (not types) to build new components, but there is no access to source code and the implementations are encapsulated. Many of the details in the DCOM specification are considered as implementation issues and not specified by CORBA [Chung].

➢ Interface identification

Interfaces and classes (implementations) are uniquely defined in COM by use of UUID. The UUIDs are registered in the Windows System registry[3]. UUIDs allow for versioning where a server can extend the functionality by implementing a new interface with a new UUID, while also implementing the old interface [Grimes97]. Older clients request the old interface without knowing about the new, new clients can use both the old and the new interface. CORBA identifies interfaces by the interface name and the implementations by mappings in the *Implementation Repository* (see Figure 2). Java/RMI identifies classes by name and implementations by mappings to an URL in the RMI registry.

➢ Object identification

An interface pointer in COM, object reference in CORBA and ObjIDs (faulting reference) in Java/RMI, provides unique identification of server objects. The interface pointer/object reference serves as object handles at run-time. When creating a DCOM server object, the server can choose to always return the same interface pointer. Different clients can therefore connect to the same object instance with a particular state[2]. Monikers and/or Running Object Table (ROT) can be used to bind an object reference to a named object instance [Chung]. CORBA clients may also connect to the same object instance if there is any existing instance that matches the requested type. Both CORBA and Java/RMI object references normally denote abstract, long-lived objects. The references are immutable in that

[1] Java supports only single implementation inheritance, but the use of marker interfaces (like the Serializable interface) extends the behaviour of a class more than strict single inheritance.

[2] COM separates the object identity from state. This violates the OO principle of encapsulation. COM classes are expected to provide standard factories, but the standard class factory does not provide a way to initialise the object with a particular state. The state must be explicitly loaded into the object instance.

they reliably denote throughout that object's lifetime, regardless of the location or activation state of the object [Rosen98].

➢ Invocation of objects

CORBA and DCOM support both static and dynamic invocations of objects. COM implements the dynamic invocation a bit different from the DII/DSI in CORBA (see below) or the Java Reflection. Normally the MIDL compiler creates the proxy and stub code, which are registered in the Windows system registry[3]. The IDispatch interface is used dynamically to invoke methods on the interface based on a type library-driven marshaling. Instead of using a separate proxy/stub DLL that contains information specific to an interface, a generic marshaler can perform marshaling by reading type library information [Grimes97].
CORBA objects can be invoked by *the Dynamic Invocation Interface* (*DII* in Figure 2) that allows a client to directly access the underlying request mechanisms provided by an ORB. An application uses the DII to dynamically query requests to objects without requiring IDL interface-specific stubs to be linked in. Unlike IDL stubs (which only allow RPC-style requests), the DII also allows clients to make non-blocking *deferred synchronous* (separate send and receive operations) and *one-way* (send-only) calls. The object information must however be in *the Information Repository* (see Figure 2). *The Dynamic Skeleton Interface* (*DSI* in Figure 2) is the server side's analogue to the client side's DII. An ORB can deliver requests to an object implementation. The object implementation does not have compile-time knowledge of the type of the object it is implementing. The client making the request has no idea whether the implementation is using the type-specific IDL skeletons or is using the dynamic skeletons.
Any Java object inherits the `java.lang.Object`, which holds the type information. Reflection and Introspection can be used to query the object type information. The Reflection API (in version 1.2 of the Java language specification) is however not integrated with Java/RMI[4] [Emmerich00].

➢ Parameter passing

The three technologies support parameters passing by either value of by reference. COM and CORBA interfaces are passed as references, Java Interfaces must implement the `java.rmi.Remote` interface to be passed by reference, all other objects and parameters are passed by value. Parameters that are Java objects must implement the `Serializable` interface to be passed by value.
COM parameters can be specified in IDL to either be passed by value or by reference. CORBA parameters that are not specified as CORBA interfaces, including complex data types, are passed by value. The reason for this is that CORBA does not define classes in the IDL (CORBA objects can be implemented by almost any language including languages that are not object-oriented). Object by value (OBV) is a quite new extension to the CORBA standard. OBV means that a CORBA object, in addition to be passed by reference, also can be passed by value over the network. This feature is synergistic with the Java-to-IDL mapping [Welsh01]. But unlike Java it is only the state that can be transmitted, this because of a possible different language at the other end[5].

➢ Garbage collection

Both DCOM and Java attempts to perform garbage collection of remote server objects. CORBA does not attempt to perform general-purpose garbage collection, but some CORBA products do include proprietary services for garbage collection. A COM server object keeps track of the clients by using reference counting as described in 2.2. When no clients reference the COM object instance, it deletes itself. The DCOM method for garbage collection (a pinging mechanism) has been criticised due to increased network traffic. Java objects have a built-in reference counting where a separate process within the JVM, the garbage collector releases instances when no other object has references to the object.

➢ Hardware support

DCOM is mainly supported on the Windows platform, but it has been attempts to provide COM on other hardware platforms without being very successful (for instance SOLARIS); CORBA and Java/RMI are supported on nearly all hardware platforms.

---

[3] Either as part of a server object in the form of a dynamic link library (DLL) form, or as a type library.
[4] It is not possible to pass the meta-data that is needed for a dynamic request across to a remote object, as it does not implement Serializable.
[5] It has been criticised that the value types sent by OBV are not strictly CORBA objects. The present OBV specification has for this reason (and some other) been felt unsatisfactory [Welsh01].

The main advantage with Java middleware is that the same version of an application can with few changes run on any platform with a compatible JVM.

➢ Network protocols

The DCOM network protocol is strongly tied to RPC, the CORBA and Java/RMI network protocols are not [Chung]. The CORBA GIOP (General Inter-ORB Protocol) specification was submitted to provide interoperability between ORBs operating on different network protocols. The IIOP is the Internet specialisation of GIOP providing transport over secure socket layers over TCP/IP. All CORBA 2-compliant ORBs are free to use alternative protocols, as long as they provide IIOP as well [Welsh01].
The network protocols for the three standards are not mainstream protocols for the Internet. DCOM and IIOP will normally not work over firewalls where HTTP is the only usable protocol. Java ORBs can however be downloaded as an applet (ORBlet) simplifying distribution. Java ORBs can also mix and match without recompilation as long as there is a JVM installed.

➢ Security

The security model of DCOM in Windows NT is based mainly on the NT LAN Manager security, while CORBA has defined an own security service to handle authentication of clients. The security model of Windows NT is Windows specific and does not comply with other security standards like Kerberos. This causes complications for the security of DCOM components if other security protocols or LAN's like for instance Novell NetWare are used. DCOM expects the user to be authenticated and authorised as a user in the Windows NT domain to control access to the components. If Novell NetWare takes care of user authentication and authorisation on the client side and the user is not defined as a Windows NT user, this causes authorisation problems on the DCOM server. It is however possible to avoid these problems by not using the built-in DCOM server security, with the possible consequences this may have on malicious use. Another problem with the Windows NT security is scalability and the missing support of delegation. Windows 2000 has implemented the Kerberos security model, which is more scalable, and supports delegation and mutual authentication between the server and the client.

CORBA provides security in two levels where level 1 comprises user authentication, authorisation, data encryption and integrity. Level 1 allows applications that are not security savvy, to relegate themselves to a secure domain. Level 2 provides a stronger security model where applications are security aware.
Java/RMI uses the built-in Java security mechanisms. A separate security manager (`java.rmi.RMISecurityManager`) can be used to rise the security awareness, for instance by using the Secure Socket Layer (SSL). The security manager is not mandatory, but makes RMI clients able to handle serialised objects where the client does not have the corresponding class files in the local class-path.

➢ Component Models

With the convergence of the EJB and CCM component models, it is in reality only two competing component models, Microsoft and the others. While EJB and CCM are fully object-oriented, COM+/MTS is component based. They all provides services like transactions, security, concurrency and multithreading. Scalability is well supported by all models through application server products.
Both EJB and CCM provide a XML descriptors to label components with packaging and deployment information. CCM in addition provides assembly descriptors to show how CORBA components should be interconnected. Registry information and type libraries are used on Windows NT to deploy DCOM components. In Windows 2000, deployment scripts are automatically created for the specific server when a DCOM component is installed. These deployment scripts can either be installed manually, or be downloaded (if the client is an Internet browser).
Most of the application servers products shipped today include support of all three component models, even if the core architecture often is build on top of a CORBA architecture. DCOM components are normally extended with a CORBA bridge when installed on an application server.
The biggest difference between the component models is in the notion of object identity and life cycle where COM normally does not implicitly handle state together with the object identity. The state must be loaded separately after the COM object creation. The lifetime of a COM object is bounded to the lifetime of its clients and ultimately by the lifetime of the process in which it exist [Rosen98]. A COM object in MTS may over its lifetime load and store several distinct, identifiable states, and be used by several clients. Client programs are usually responsible for explicitly managing states and its associations with COM objects. Client programs to CCM (CORBA) and EJB objects do not have any notion of state with a separate identity, apart from the object that encapsulates it.

5 Summary and Conclusions

The distributed object and component models compared in this essay are quite similar with respect to both architecture and supported features. This is also the conclusion of most of the published articles that have compared the technologies the last years (e.g. [Chung] [GR00] [MCP98] [Raj] [Payton] [Welsh01] [Rosen98]). The most significant architectural difference is in the notion of object identity and life cycle.

CORBA has the advantage of being object-oriented, more modern and equipped with more comprehensive features than COM. The CORBA technology has had a great disadvantage that it has been too low-level and complicated, and therefore hard to learn, requiring very skilled developers [Welsh01]. The CORBA objects have therefore been hard to reuse effectively. The CCM is a great step in the direction of defining and grouping CORBA objects to higher abstractions as components. Another advantage with CORBA is that developers can choose almost any language, hardware platform, networking protocol, and persistence technology and still use CORBA.
The Java framework and object middleware is a very good choice for application integration and is supported by almost all application servers, integration brokers, transaction monitors, XML servers, B2B/enterprise application integration products, and data translation engines [GR00]. CORBA complements Java in the integration of applications and legacy systems written in other languages than Java.
Microsoft technologies like DCOM, COM+/MTS are a very good choice for organisations that mainly use Windows technologies to run mission-critical applications. The biggest disadvantage with the Microsoft technologies is the (current?) limitation to the Windows platform.
Interoperability between different object models has been an issue since the middleware war started in the middle of 1990's. The introduction of Java as the Internet language has fired even more under the interoperability issue, especially since most applications were written in other technologies. Java has since then grown from a programming language to a complete application development platform and is supported by most of the application integration systems. The Java standard (J2EE) has been extended mandatory to provide an ORB implementation (Java-IIOP) in addition to the RMI as a distributed technology option. The Java and CORBA architectures are similar enough to interoperate easily, while COM is relatively incompatible with these two. OMG has defined a CORBA-COM bridge to meet the interoperability issues of these two distributed object models (see [Rosen98] for a thorough description of the COM-CORBA interoperability).
XML (eXtensible Mark-up Language) and SOAP have together with application servers shown a great promise for interoperability between the different object models. Microsoft and OMG as well as many other vendors have adopted XML. XML is a true platform independent standard and can be used by any type of application or object model on any hardware platform. XML-files are ASCII-files and can be transported by the HTTP-protocol avoiding problems with for instance firewalls. A disadvantage with XML is to define agreed meta-models between suppliers and consumers of XML messages. In addition is XML quite space-intensive with possible very large overhead of descriptive information. It is however possible to use style-sheet translators as XSLT, to translate a given XML-file from an XML-model to another XML-model. SOAP satisfies the need to exchange structured data by the Web independently of the underlying platforms. SOAP does not, in contrast to CORBA, offer any services or management tools. The firewall argument in favour of SOAP is maybe dubious by making SOAP perform RPC calls through firewalls. This subverts the security regime because the firewall thinks that it is just harmless Web traffic that passes through it. Microsoft's .NET is promised to implement both XML and SOAP giving the Windows applications opportunity easily to interact with non-Windows applications.
Application servers[6] represent a convergence of multiple technologies including object and component models, distributed application processing, and object transaction monitors. Services like transactions, security, persistence, threading, naming and directory service, and distributed processing features, are provided by application servers. The servers most often support all EJB, CORBA and COM, providing interoperability between the component models. Integrated development environment tools (IDE) support the developers to set up and use frameworks and different architectures often based on patterns. The evolution of tool support for development of distributed systems may help to avoid much of the complicated programming of distributed components. Application server and integrated development environments will together most likely make it possible to develop components that uses the distributed technology that best solve the requirements for the distributed application. It is even possible to support interfaces from any of the different distributed technologies based on the type of client requesting services from the component(s).

[6] Maybe with an exception from the Microsoft application server products

References

[Chung]
P.Emerald Chung, Yennun Huang, Shalini Yajnik, Deron Liang, Joanne C Shih, Chung-Yih Wang, Yi-Min Wang: DCOM and CORBA Side by Side, Step by Step, and Layer by Layer.
http://www.cs.wustl.edu/~schmidt/submit/Paper.html

[COM]
Microsoft: The Component Object Model Specification,
http://www.microsoft.com/com/resources/specs.asp

[CORBA]
OMG: The Common Object Request Broker: Architecture and Specification, Revision 2.4.2 February 2001
http://www.omg.org/technology/documents/formal/corbaiiop.htm

[CDK01]
George Coulouris, Jean Dollimore, Tim Kindberg: Distributed Systems, Concepts and Design. Addison-Wesley 2001

[DCE95]
AES/Distributed Computing - Remote Procedure Call, Revision B, Open Software Foundation.
http://www.osf.org/mall/dce/free_dce.htm

[Eddon98]
Guy Eddon, Henry Eddon. Inside Distributed COM. Microsoft Press 1998

[Emmerich00]
Wolfgang Emmerich: Engineering distributed objects. John Wiley & Sons, Inc. 2000

[GR00]
Gartner Research; Distributed Object Middleware: An Introduction.

[Grimes97]
Dr. Richard Grimes: Professional DCOM Programming. WROX Press Ltd. 1997.

[Java/RMI]
RMI Architecture and Functional Specification
http://java.sun.com/j2se/1.3/docs/guide/rmi/spec/rmiTOC.html

[Mahmoud00]
Qusay H. Mahmoud: Distributed Programming with Java. Manning Publications Co. 2000

[MCP98]
Munir Mandviwalla, Paresh Chobhe, Steve Ponomarev: ActiveX/DCOM and Java/CORBA on the Web.
http://joda.cis.temple.edu/ced/rep2/

[OMG]
Comparing ActiveX and CORBA/IIOP.
http://cgi.omg.org/library/activex.html

[Orfali96]
Robert Orfali, Dan Harkey, Jeri Edwards: The Essential Distributed Objects Survival Guide. John Wiley & Sons, Inc., 1996.

[Orfali97]
Robert Orfali, Dan Harkey, Jeri Edwards: Instant CORBA, John Wiley & Sons, 1997

[Orfali98]
Robert Orfali, Dan Harkey: Client/Server Programming with JAVA and CORBA, Second Edition. John Wiley & Sons, Inc., 1998.

[Payton]
Michele Payton: CORBA vs. DCOM: a Comparison
http://www.cs.colorado.edu/~getrich/Classes/csci5817/Term_Papers/payton/

[Pinnock98]
Jonathan Pinnock: Professional DCOM Application Development. WROX Press Ltd. 1998.

[Raj]
Gopalan Suresh Raj: A Detailed Comparison of CORBA, DCOM and Java/RMI.
http://www.execpc.com/~gopalan/misc/compare.html

[Rosen98]
Michael Rosen, David Curtis: Integrating CORBA and COM Applications. John Wiley & Sons, 1998

[Schmidt]
Distributed Object Computing and CORBA Documents http://siesta.cs.wustl.edu/~schmidt/corba.html
[SOAP]
The Simple Object Access Protocol: http://msdn.microsoft.com/xml/general/soapspec.asp
[SunONE]
The Sun ONE architecture: http://www.sun.com/software/sunone/wp-arch/
[Szyperski98]
Clemens Szyperski: Component Software; Beyond Object-Oriented Programming. Addison-Wesley 1998
[Vinoski97]
S. Vinoski, CORBA: Integrating diverse applications within distributed heterogeneous environments, in IEEE Communications, vol. 14, no. 2, Feb. 1997.
http://www.iona.com/hyplan/vinoski/ieee.ps.Z
[Welsh01]
Tom Welsh: OMG, CORBA, and the Whole Nine Yards.
In Distributed Computing Architecture/E-Business Advisory Service.
Publisher: Cutter Consortium 2001

Additional resources

- http://www.corba.org/
- http://www.microsoft.com/
- http://www.microsoft.com/com/
- http://java.sun.com/
- http://java.sun.com/j2se/1.3/docs/guide/rmi/
- http://java.sun.com/j2se/1.3/docs/
- http://www.w3.org/
- http://www.omg.org/
- http://www.biztalk.org/