UPEC
EdiCom

member of the Frontec group

Distributed components in a Microsoft environment

Thesis project

1999-04-29

1 ABSTRACT

Distribution over a network may sound a bit old-fashioned; documents, files and programs have been distributed over networks for quite a long time. The difference with the new distributed architectures is that they have extended the term distribution to include applications consisting of scattered parts, i.e. distributed components. A distributed environment has many advantages, for example, parts of applications can be distributed to the computers best meeting their demands.

The two main competitors in the area of distributed architecture are Microsoft and the Object Management Group (OMG). Microsoft is fighting for its Distributed Component Object Model (DCOM) and OMG for its Common Object Request Broker Architecture (CORBA).

Covered in this report is DCOM, with purpose to examine what demands a distributed component should meet, and what a distributed component's targeted system should look like.

DCOM was found to provide rich support for distributed computing in a Windows environment. DCOM components are easily assembled into fully adequate applications. DCOM has, since it was first shipped in 1996 matured quickly in the Windows environment. This especially holds for the easy management of DCOM applications.

Today, businesses ranging from middle sized to large, are the ones best suited for DCOM implementations. This will however change when DCOM over the Internet reaches a more mature level, and DCOM applications will then span all PC-computer markets.

Final Thesis report at KTH.
Author: John Sjöberg
Examiner: Björn Lisper, KTH
Academic supervisor: Niclas Finne, SICS
Company supervisor: Jan Bäckström, Upec EdiCom
Performed at: Upec EdiCom

April 29, 1999
Stockholm, Sweden

DISTRIBUTED COMPONENTS IN A MICROSOFT ENVIRONMENT

Contents

2 ACKNOWLEDGE

2.1 Goals

The goals of the project, as stated in the project specification are:

1. Gain competence in Microsoft's architecture for distributed components, on behalf of Upec EdiCom.
2. Gain knowledge in which criteria that should be fulfilled for:
   a) A component to be able to provide a value-added functionality/service in a distributed architecture.
   b) A target system to be able to benefit of a distributed component.

2.2 Layout

The focus of the report lies in the understanding of concepts behind Microsoft's distributed architecture. How do the different parts of it work? How do they work in conjunction with each other? What possibilities are there with DCOM?

The report begins with a discussion on components, followed by another discussion on distributed systems. The reader is then confronted with Microsoft's different technologies that its distributed architecture is based on. Then a brief analysis of DCOM is presented followed by a discussion on how, who, where and when DCOM is/could be/should be used and who is using DCOM, as well as some objective conclusions on DCOM. The forthcoming evolution of Microsoft's distributed architecture is explained, and the thesis is eventually brought to an end with a comparison of DCOM and CORBA.

3 COMPONENTS

3.1 A Component

Shortly described, a component can be said to be an encapsulated piece of software that is only accessible via its interfaces – as depicted in [1]. A component should further be designed for easy composition to other components.

A component can be viewed as a coarser grained object that can be divided into three separated parts:

- **Specification**
- **Design**
- **Executable**

The **specification** of a component is a description of the semantics. The specification contains information about how the component can be used and what result the component delivers. This is in the case of COM/DCOM handled with the IDL - Interface Definition Language, which describes the interfaces of a component. The interfaces must declare what methods are available in a component, how those methods are called (with what parameters), and what their return values are. It is very important that the specification is correct, as a component making use of another component only accesses the other component through the other one's interface. Apart from interfaces, the specification will also include some kind of describing text that can be either read by a human directly or through some kind of "translator" tool. This is important for "assemblers" - persons that will use components to put applications together.

The **design** is the actual implementation of a component, i.e. the code that the component is based upon.

The **executable** is the platform dependent data (machinecode/bytecode) which delivers the component's capability on a designated platform.

A component is often referred to as a service. This is due to the fact that a component will fulfill some kind of contract between the component and the components consumer. A contract is based on the specifications of components. One component declares that it provides one kind of service and another component, the consumer, makes use of that one.

How or in which programming language the component is constructed doesn't matter, it is the functionality described through the interface that is important. This will open up a market for plug-and-play components. A new/upgraded component is inserted and the application dynamically adapts to it.

A component can be regarded as the practical implementation of the object oriented model, focusing on coarser grained objects or business components.

Example:
A spell-checker is used as an example of a component in [2]. A spell-checker can be implemented as a component and incorporated into an editor. The company creating and updating the editor (company E) might have bought the spell-checker component from company A. But then company B comes up with a component that is twice as fast as company A's. As E would like to stay competitive on the market with the best editor, it buys B's spell-checker component. As the spell-checker is built as a component, it is not a big issue for E to distribute this new spell-checker. It is just to install the new component, and the next time someone want to use the spell-checker B's component is launched.

3.2 Why components?

Components and the concept of standalone objects that can plug-and-play across networks, applications, languages, tools and operating systems are quite new to the software industry. For components to be able to support that concept, a new infrastructure is needed. Such architectures are however being developed, and the development of components and architectures are tightly integrated.

3.2.1 Pros and cons of components

Pros:

- Easy enhancements.
- Simpler and faster upgrades of applications.
- Adaptability.
- More effective as different parts of a development team easily can work on their own specific components.
- Legacy applications and systems can be wrapped as components and thus used with new software too.

Cons:

- Time-consuming development.
- Assemblers possibly needed.
- Component-based development is quite complex at the moment.

Time-consuming:
Development of components requires more modeling than traditional development if the benefits of using components will be achieved. One reason to that is that an interface never changes. That is one of the most important rules within component-based development, the implementation of an interface however could change. Anyway, because interfaces are persistent, they must be designed with some caution in mind:

- Is it flexible enough? (Can it be used within future applications as well as other applications of today?)
- Is it too flexible? (Does the flexibility deteriorate the performance of the component?)

These questions should be answered before the implementation of the component starts. If a development project is divided up between different groups, then well described behaviors and interfaces are needed so that the developers know what their guidelines are. If not, the resulting pieces of the different groups would probably not fit together.

Assemblers possibly needed
Assemblers will be needed to assemble components into applications. Assemblers will be the ones understanding how the components can be linked together for maximum performance and maximum flexibility. Assemblers will constitute a new kind of developer that is needed, and it will take some time to educate them.

Component-based development is quite complex at the moment:
The most common component models today require a lot more knowledge from the developer than programming in pure C++ do for example. Furthermore, there are not many tools available, which facilitate the design and implementation of components (such as the tools from Rational[1] for different programming languages).

3.3 Components vs. Objects

There is no standard definition of what a component actually is, but there has been a lot of discussion on this topic. The diffuse description of a component, given by Heinz-W. Schmidt in [3] is typical – "Components are large grain objects. They comprise other auxiliary objects and are subject to interoperability requirements […] components combine distributed objects with contracts and substitutability". It seems like there are no easy explanations to what a component really is, and if there are any differences between a component and an object. They both aim for the same goals, but they intent to score in different manners.

The "definition" of a component and an object does not differ very much, so rather than examining their "definitions", they should be examined by the way they are incorporated in practical use.

A high-level description of a component that could also be applied to an object is presented in [1], where it is said that a component should:

[1] Rational (www.rational.com) has tools for design and reengineering. The designer use graphical tools for designing of classes, the classes' methods and the relationships between the classes. It is then possible to get a skeleton of code out from the tool and the reverse is then also possible (get the design out of the code).

- Offer operations.
- Be interchangeable with other components offering the same specification.
- Be encapsulated.
- Have the possibility to be tested, standardized and reused.

In addition, a component may also depend on other components for its behavior.

I will not step into a deep discussion about the difference between an object and a component, but the major differences are shown in table 1, below.

	OBJECT	**COMPONENT**
1.Access/Reuse	In the code level – necessary to compile every change.	In the specification level. Independent of different languages or environments.
2. Incorporation	Through compiling and replacement of the old code.	Dynamic incorporation at run-time.
3. Encapsulation	Programmer driven.	Interface enforced.
4. Orientation	Technology-oriented.	Business-oriented[1]
5. Granularity	Fine-grained.	Coarse-grained.
6. Development	Standards-based.	Language-based.
7. Standards	Open standards.	Proprietary standards.

Footnote: It should be noticed that the first point, that of reuse in the case of a component isn't accurate for the moment, as there are not any components, yet, that support every environment and language.

Table 1. *Comparison of objects and components.*

[1] "Components are business-oriented" simply means that components are described by their functionality and not by how they perform a specific task. Components provide services, as opposed to objects, which provide operations. Components should be on a level so that "everyone" can understand them (such as a "spell-checker component").

4 CBD - COMPONENT-BASED DEVELOPMENT

Component-based development has gained a lot of interest ultimately. A lot of people believe that components are going to be the new "era" in software development. When Butler Group [1] discusses component-based development it is called "the industrialization of software delivery". It is a strong expression, but if you look at the possibilities of components, it is with no doubt quite suitable. Components open up a brand new market to the development industry. Business services can be designed as components and thus they are inherently reusable, as opposed to being designed for obsolescence. Developers can create components with certain functionality and sell them as plug-ins to customers. The customers can either be companies using the components for their own applications or they can be end users adding them into their existing applications as upgrades or extensions, which is similar to how plug-ins for web browsers work.

As component-based development is a quite new phenomenon, organizations with influence in this area are trying to introduce new concepts to the development industry. Most of these concepts should be appropriate for the object-oriented and generic development as well.

4.1 Steps to CBD

Important steps when developing components:

- **Modeling**
- **Acquisition**
- **Build**
- **Assembly**
- **Wrapping**
- **Execution**

4.1.1 Modeling

The needs for modeling are: to understand which components to build, to divide up a large complex project and ensure the resulting pieces will fit together again, to ensure clear communications between a component supplier and consumer.

It is important that the problems are broken down to correct useful parts (components). To ensure that the application will feature long term adaptability, widespread reuse and physical deployment, it is very important that the components developed are well modeled.

Consider an application development team that is divided up into smaller groups, where each group develops their own components, to finally join all the components to assemble the application. The modeling must be deep enough to provide the groups with sufficient information about the components of the other groups, if the components are going to interoperate. Therefore it should be ensured that all the groups follow the specification for the components, acquired in the modeling process.

A standard way of communicating the behavior of software, independent of how it is implemented would be needed in the modeling. A solution to this problem could be underway through UML – Unified Modeling Language. According to [1], UML will be used in every new object-oriented software project after 1999.

4.1.2 Acquisition

The acquisition process spans from identification of appropriate components to obtainment of those components. Included in this process are requirements definition, searching, evaluation and selection, together with management aspects which will include legal, economic, organizational and measurement issues.

Although definitions of application requirements have been used by companies for a long time, the process of defining requirements will not be the same with components. When dealing with components it is important to take into account the current as well as the planned interface architecture. Finding the right components is not easy, but there are several catalogs available through the Internet, where searches can be made:

- http://browserwatch.internet.com/activex.html - ActiveX controls
- http://www.javasoft.com/beans/index.html - JavaBeans
- http://www.software.ibm.com/ad/visage/rc/ - Component catalog for IBM VisualAge (incl. Java, C++, Smalltalk and COBOL)

4.1.3 Build

The primary target is to deliver an "encapsulated piece of software" that is only accessible through its interfaces. The components should be built for reuse, independent of specific applications. Built for reuse means that the component should be sufficiently generic to facilitate its reuse in future projects.

The authors of [1] points out that there are some generalizations that make a component more reusable. The two most appropriate ones are:

- **Widening** – Extending the requirements of a component makes it more probable that the component can be reused in the future.
- **Configurability** – Building smaller set of components instead of one big that satisfies all requirements. The components can then be combined in different ways to meet various demands.

4.1.4 Assembly

The assembly process can be thought of as a process equal to the one constructing a car of different parts, from different suppliers.

The assembler (the person working with the assembly of components) will have to be well informed about what components are available, and how to acquire, customize and extend them.

Conceivable actions to be performed in the assembly process are:

- **UI design** – Components may come with general UI elements and must in that case be customized.
- **Workflow and process control** – Concatenation of components and design of the manner of execution.
- **Testing and debugging** – Testing of the application during the construction. The participating components should be tested individually, before the start of the assembly.

4.1.5 Wrapping

A wrapper is a layer of software that provides alternative interfaces around the code that is wrapped. This could be used when dealing with legacy applications or existing applications that need to be ported to different environments. Wrapping can often be considered the best choice if comparing the effort required to wrap an application and building the ideal solution from scratch. Reasons for wrapping instead of building that are given in [1] are:

- Other existing systems are dependent on the application. Ensuring that a new component still feeds these dependent systems could take considerable effort.
- Time and effort taken to wrap can be considerably less than building a new component. The more complex the application being wrapped, the greater the saving.
- Skills to build a new replacement may be in short supply.

4.1.6 Execution

When it comes to execution the environment must be considered. Will the application run on Windows, or UNIX?

The architectures (goes along with the environments) that most IS (Information Services) managers consider are Microsoft or non-Microsoft architectures. A Microsoft architecture consists of COM/DCOM[1] with MTS[2] and MSMQ[3]. Non-Microsoft architectures are mainly different vendor implementations of CORBA[1]. The Microsoft architecture is covered in-depth in this report with some relations to CORBA taken into account (see chapter 16).

[1] COM/DCOM are abbreviations for Component Object Model/Distributed COM.
[2] MTS is the abbreviation for Microsoft Transaction Server.
[3] MSMQ is the abbreviation for Microsoft Message Queue.

4.2 How can CBD improve the software application?

A general discussion of component-based development and its fulfilling of the ISO 9126, which is a generic definition of software quality, has been performed by Richard Veryard in [4]. In the following part, he evaluates the, in ISO 9126, six desirable characteristics used to measure quality of software.

Functionality Use of pre-existing components allows faster delivery of greater functionality.

Maintainability The modular structure of a component-based solution allows individual components to be replaced easily.

Usability Use of standard components supports commonality of GUI. CBD also supports desktop integration, which gives the user a single view of heterogeneous data.

Efficiency Performance bottlenecks can be identified, and the need for performance tuning can then usually be localized in a small number of performance-critical components. Components can be internally optimized to improve performance, without affecting their specification; components can be moved between platforms to improve performance, without affecting the functionality or usability of the application.

Reliability Given a formal and complete specification of a component, the reliability of a component comes down to the simple question: does the component provide a correct and complete implementation of its specification. The reliability of the application as a whole is a separate issue, but is clearly enhanced when the application is constructed from reliable components.

Portability The specification of a component is platform-independent. A component can be quickly rebuilt for a new platform, without affecting any other component. (With some tools, this will often require no alteration to the internal design, merely a regeneration of the executable portion.)

[1] CORBA is the abbreviation for Common Object Request Broker Architecture. CORBA is developed by the Object Management Group (OMG).

5 DISTRIBUTED SYSTEMS

A distributed system is in [5] defined:

A distributed system is a collection of independent computers that appear to the users of the system as a single computer.

5.1 Distributed systems vs. Centralized systems

The reasons why anyone would like to use a distributed system in regard to a centralized system are mainly:

1. **Performance vs. costs**
2. **Reliability**
3. **Scalability**

1. A big mainframe computer is more expensive than a collection of smaller computers, even if the small computers together have the same computing power as the mainframe.

2. If a system is based on a mainframe computer and that one goes down, the whole system will be down. If, on the other hand the system is based on many small computers the loss of one or a few of those computers is not necessarily critical for the functionality of the system.

3. If more computer power is needed, another computer (or computers) can easily be added to the distributed system, which wouldn't be possible in a centralized system. Also, imagine a big central computer situated in New York, handling all the computing in the USA. Even if it was possible to create a computer that could work that fast, the network that connected to this mainframe wouldn't be able to prevent congestion.

A distributed system is, if correctly made, superior to a centralized one, but a distributed system is much harder to create and maintain. There are today no implementations of a distributed system that provide all the requirements of a good distributed system (a system with answers to all issues described below in 5.2). Even if not all goals are reached, distributed systems are mostly the best choice.

To maintain the single system image[1] there should be a single, global interprocess communication mechanism allowing any process to talk to any other process. Furthermore, the security scheme should be global, not a mix of access control lists, capabilities, or UNIX protection bits, as that would hardly export the image of single system. The same global requirement goes for process management. Finally, the file system should look the same everywhere, as having different file systems on different machines would raise problems with naming and the global hierarchical file system.

5.2 Tasks to deal with for a Distributed System

Five important issues to be solved in a distributed system are **Interoperability**, **Versioning**, **Language independence, Fault tolerance** and **Scalability**.

5.2.1 Interoperability

Interoperable systems are in [3] defined as systems that are composed from autonomous, locally managed, heterogeneous components, which are required to cooperate to provide complex services.

This means that:
1. The components of such a system should be able to operate independently of the other components in its environment.
2. It should be possible to manage the components locally, even if they are physically distributed.

[1] Single system image means that the system is transparent to the users and no matter where the system is accessed it always looks the same.

3. The components may be implemented using different programming languages and they achieve similar goals in different ways.
4. The components in the system should be able to interact (efficiently) to achieve shared goals.

5.2.2 Versioning

The problem with ***versioning*** is that components relying on some interfaces must be able to trust that the interfaces functionality are not changed when their versions are changed. If not, the components implementing a changed interface might not produce the same results as earlier. This cannot of course be accepted as the whole idea of components and their reusability would vanish.

Consider an example:
A component used to calculate the number of users logged on to a company's network could be used by a diverse number of applications. One application that uses the component to calculate the mean number of employees logged on to the network, wouldn't be very pleased if an interface (that the component implemented) was changed so that the calculation now considered the number of free hosts on the network.

5.2.3 Language independence

Language independence simply means that it shouldn't matter in what language or how a component has been implemented. Independently of how it was created, it will be able to communicate with other components implemented differently.

5.2.4 Fault tolerance

Leslie Lamport once defined a distributed system as *"one on which I cannot get any work done because some machine I have never heard of has crashed".* This has often been true and still often is. In a distributed system it cannot be accepted that if one machine (although one important) goes down it removes the availability of the rest of the system. If that happens, the system does not achieve ***Fault tolerance****.* A system that is fault tolerant masks failures, hiding them from the users so that they never get the feeling of that the system is having problems.

5.2.5 Scalability

The ability of a system to scale, its ***scalability*** is indeed very important to a distributed system. Users (i.e. hosts), servers and tools are likely to be added to the system over time. If introducing a new user to the system requires a day of work to complete it isn't very flexible. If the network traffic on the distributed system is slowed down (distinguishable by users) just because there are 100 users today instead of 10 which were the case when the system was started up, it doesn't scale well. There is a guideline covering centralization that should be considered when designing distributed systems.

- Avoid centralized components, tables and algorithms.

Centralized components and centralized tables stick much together. They are not used that much today as the old architecture with one mainframe and a set of dumb terminals has been removed from most systems. Anyway, centralized components do still exist in form of only one central mail-server for many networks. The biggest problem with having centralized components or tables is not the performance of the lookups etc., but the network connections to it. Furthermore, if a centralized component goes down, the services it normally provides can no longer be provided (implies lack of fault tolerance).

To achieve decentralized algorithms the following characteristics should apply to the algorithms:

- No machine has complete information about the state of the system.
- Machines make decisions based only on local information.
- Failure of one machine does not ruin the algorithm.

This applies to how routing of data is performed in a network.

6 MDCA – MICROSOFT'S DISTRIBUTED COMPONENT ARCHITECTURE

The term Microsoft's Distributed Component Architecture (MDCA) was stated by Roger Sessions and is not an official Microsoft term. Though, the term describes quite well Microsoft's architecture of today. When Roger Sessions coined the expression, he chose it as a better name of Microsoft's DNA (Distributed interNet Architecture). That is not how the term DNA is used by Microsoft or me. How DNA officially is described is concluded in chapter 11 of this paper.

6.1 Overview

In [6], Roger Sessions states that MDCA consist of these six bricks:

- COM (Component Object Model), which defines the basic component/object model.
- DCOM (Distributed COM), which allows components to be accessed across a network.
- MTS (Microsoft Transaction Server), which among other things provides a component runtime environment for the middle tier.
- DTC (Distributed Transaction Coordinator), which coordinates the distributed transactions.
- MSMQ (Microsoft Message Queue), which provides asynchronous communications.
- MSCS (Microsoft Cluster Server), which allows multiple servers to work as one

They are all, except for MSCS, described in this text. MSCS does currently only support two servers to act as one.

6.2 MDCA - DNA

Microsoft's Distributed Component Architecture is what many people believe to be the DNA. As outlined in chapter 11, this is not true. DNA will consist of COM+, Forms+, and Storage+ (which will be described in the DNA and the COM+ sections). Forms+ and Storage+ deal with presentation and storage, whereas COM+ will probably incorporate all the features of MDCA. The pieces that are going to be the foundation of DNA, i.e. COM+, Forms+, and Storage+, are not yet available. COM+ will be released about a month after the introduction of Windows 2000, but it will take a little longer until the solutions for Forms+ and Storage+ eventually ship.

7 COM/DCOM

COM (Component Object Model) and DCOM (Distributed COM) are very closely related and will therefore be described together in this chapter. Everything that is true for the COM structure holds for DCOM too, the difference is that DCOM just reaches a little bit further than COM.

7.1 Introduction

As COM in many ways is working in a similar manner to C++, the introduction to COM will be done using C++. Traditionally C++ libraries have been distributed in source code form. The users of a library would be expected to add the implementation source files to their make systems and recompile the library sources locally using their C++ compiler. Assuming the library adhered to a commonly supported subset of the C++ programming language, this was a perfectly workable approach.

Sharing a class or some code between different applications could not be achieved. So even if there where 10 applications that made use of one class they all had to put it into their own code and compile it independently of the others.

A solution to this problem was then to package the class as a Dynamic Link Library (DLL). In that way the class could be used by every application on the user's hard disk that needed it. The operating system's loader would, hopefully be smart enough to share the physical memory pages containing the class' read-only executable code between all client programs making use of it. However, one problem exists with this model. Classes compiled using a different compiler than the class they would like to link to would not be able to do so. This is due to different compilers using different schemes for the composition of class import libraries, and classes compiled with different compilers would therefore not be able to link with each others import library. The only way to get across this problem, according to Don Box in [9], would be to produce, for each DLL, a custom import library for each compiler. That should be quite tedious.

This approach, which is further described in Essential COM [9] by Don Box, could eventually when still only using C++ end up with a solution similar to COM.

7.2 The background

In the beginning of the Microsoft era, there was no method for having two applications interoperate. For example, the only way to embed an image into a text written in a text editor was to print them both and then use scissors and glue to join them, as Richard Grimes so concrete describe it in [11]. To simplify interoperation, Microsoft came up with the **Windows clipboard**, which let users share data among applications. It was then possible to copy something from one application and paste it into another one.

After the clipboard came the **DDE** (**Dynamic Data Exchange**). With this protocol, two applications could link pieces of data together dynamically, which means that if the data changes in the source document, the linked item in the destination document changes too.

OLE 1.0 (**Object Linking and Embedding 1.0**) was introduced almost at the same time as DDE. OLE 1.0 used DDE as it's interprocess communications mechanism. OLE 1.0 introduced the association of applications with extensions of the name of data (e.g. A file name with the extension "doc" (filename.doc) indicates that the data is a Word document). Data (objects) from different applications could also be put together and double-clicking on an object in a thus compound document would bring that object up in it's associated application. The possibility to activate objects was the biggest advantage with OLE 1.0.

Next came **16-bit OLE 2.0**. It was far more sophisticated than the first version of OLE, and it meant more than just object linking and embedding. The use of DDE for interprocess communication was gone, as well as a lot of other fundamental things as the programming model. OLE 2.0 came with **COM (Component Object Model)** as its core. COM was designed with the knowledge that distribution of objects would probably be coming some day, which would eventually result in DCOM (Distributed COM, see below). COM can be regarded as the mechanism that makes the features of OLE possible. COM will be outlined in detail later in this chapter. OLE defined compound files, an implementation of structured storage. Furthermore, inter-application drag and drop was introduced along with in-place editing and OLE Automation[1].

Then a new **32-bit OLE** came. From now on no version numbers were used for OLE. The idea of COM was that if something new was desired it could be added to the old architecture. Thus no new "versions" of OLE were to be released. With 32-bit OLE, applications could share data with other processes.

The infrastructure of COM was put completely on top of MS-RPC (Microsoft's Remote Procedure Call) system. Any cross-process call was delivered using MS-RPC. MS-RPC is based on and compliant with DCE-RPC (Distributed Computing Environment – RPC), defined by OSF (the Open Software Foundation).

A distributed OLE was the next development for Windows. It was first named **Networked OLE**, but it later changed to **DCOM (Distributed COM)**. DCOM is based on COM, but whereas COM addresses the issues of developing components that can be plugged into various applications located on single platforms, DCOM extends this with provisions for distributed shared memory management, network interoperability and transparency, and dynamic management for component references.

DCOM communication is based on OSF's DCE RPC, which defines an API to remote function calls from a client to a server. DCOM introduced some new concepts into the COM world, such as security. Before DCOM, security in COM was not considered important at all. This was based on thoughts that users of COM applications had to be locally logged on to the machine the application was running on if they would be able to run it. This is true in most cases, but not all, for example when a web browser causes a web server to launch a CGI application that in turn activates a COM object. Anyway, introducing DCOM made the need for security obvious.

After the release of DCOM the latest shipping from Microsoft is **ActiveX**. Microsoft first coined the term ActiveX at the Internet Professional Developers Conference (Internet PDC) in March 1996. ActiveX referred to the conference slogan ”Activate the Internet” and was more a "call-to-arms" than a technology or architecture for developing applications. ActiveX came out as an expression but has now come to signify a lot more and it means a lot more than just "Activate the Internet". In [13] Jerry Anderson says: "ActiveX and OLE have become synonymous. What people once referred to as OLE Controls (OCXs) are now referred to as ActiveX Controls. OLE DocObjects are now ActiveX Documents. In some cases, entire documents on how to implement OLE technologies have been updated to be ActiveX technologies, and the only thing changed was the term OLE, which now reads as ActiveX."

Due to the same Jerry Anderson, ActiveX spans the whole area from Web pages to OLE Controls and furthermore it represents Internet and applications integration strategies. Anderson also establish that: "It [ActiveX] has come to signify small, reusable components that can get hooked into all the latest technologies coming out of Microsoft and the Internet".

In the Microsoft family, ActiveX is the presentation tier, whereas COM and DCOM, which both can act as both clients and servers, are independent of the tier the user is working on. ActiveX allows visual components to plug in together on the client.

7.3 How COM/DCOM works

COM is a binary standard of how communication with an object is handled and how that object handles its own lifetime and how it tells the world what it can do. The COM programming model is based on one simple idea: program against abstract interfaces, not concrete implementations. COM is language independent to the extent that the language supports pointers, which is required for the functionality of COM (see 7.3.1, page 18).

[1] OLE Automation is now just called Automation. Automation is the possibility to access COM objects from typeless languages like Visual Basic.

COM treats interfaces, implementations and classes as three distinct concepts. An interface is basically an abstract protocol for communication with an object. Implementations are concrete data types that expose one or more interfaces through precise semantic interpretations of each of the interface's abstract operations. Classes are named implementations that represent concrete instantiable types.

7.3.1 Interfaces

In the COM specification it is said: "An interface is an immutable contract that implies concrete syntax and loose abstract semantics". This means that an interface is a contract, because when an object implements an interface it promises to provide code for every member function of the interface implemented. It is only through interfaces that a client and an object communicate. In [2] it is stated out "…an interface is a semantically related group of member functions that carry no implementation (nor any data members)…[The interfaces] act as a single entity and essentially represents a "feature" or a "design pattern". Objects then expose their features through one or more interfaces, …". An interface is immutable to stay consistent forever. This consistency is conceived with GUIDs (Globally Unique Identifiers, which are described later in this chapter). Thus, if an interface is once defined and registered within an application, that interface must never ever change.

An interface can be described using COM's interface definition language (IDL). A compiler, the Microsoft IDL compiler (MIDL) is then used to compile an object's IDL to produce a type library, a proxy and a stub. This is further described later in this chapter.

The interface that is the root of all COM interfaces and that is the only interface that does not derive from another COM interface is the IUnknown interface. Every other COM interface must either derive directly from IUnknown or through another COM interface which in turn must itself derive from IUnknown or another COM interface. There is no limitation imposed from COM's side on the depth of the interface hierarchy, as long as the base interface is IUnknown. One limitation though is that COM interfaces cannot derive directly from more than one interface. One of the reasons for this is due to the close relationship between COM and DCE RPC. By limiting interface inheritance to only one base interface, the mapping between COM interfaces and DCE RPC interface vectors is straightforward. However, any implementation can chose to support as many interfaces as it desires (i.e. multiple inheritance).

Calls to interface member functions imposes that the address of the call itself is discovered at run time using the value of the interface pointer, which is a value known only at run time. This means that when the kernel loads a client application, there are no "import records" to map to absolute addresses. Instead, those addresses are computed at run time. This makes the use of interfaces a true form of dynamic linking to a component's functionality. This means plug-in features, as components can be inserted at run-time.

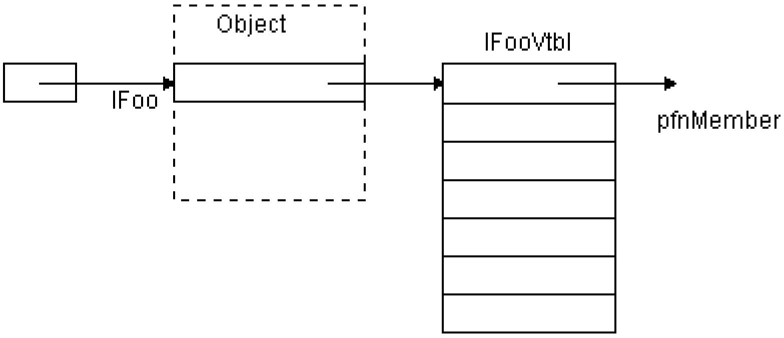


Figure 1.

At run time, an interface is always seen as a pointer typed with an IID (Interface identifier). The pointer itself points to another pointer that points to a table[1] that holds the addresses of the implementation of each member function in the interface (see figure 1). This binary structure is a core standard of COM, and COM depend upon this standard for interoperability between software components written in arbitrary languages. As long as a compiler can reduce language structures down to this binary standard, it does not matter how the component is implemented – the point of contact is a run-time binary standard. Some languages that support this and thus can be used for writing COM components are C, C++, Java, Smalltalk, Ada, Object Pascal, Delphi, BASIC and Visual Basic.

[1] An interface is actually a pointer to a vtable. Vtables are usually shared by multiple instances, so the methods need a different pointer to be able to find the object that the interface is attached to. This is the interface pointer, and the vtable pointer is the only thing that is accessible from it by clients of the interface. By design, this arrangement matches the virtual method-calling convention of C++ classes, so a COM interface is binary-compatible with a C++ abstract class.

7.3.2 The object reference model

COM doesn't provide an explicit delete operation neither does it assume the presence of an automatic garbage collector. Instead, COM relies on reference counting to determine when an object can safely unload itself. A COM object maintains a count of how many clients hold references to its interfaces. When the object hands out a pointer to one of its interfaces, it adds one to the interface's reference count. If the client acquires an interface pointer to an object from a source other than the object itself, that client must call the object's AddRef method, which also causes the object to increment its reference count by one. Whenever a client is finished using an interface pointer, it calls Release on that pointer and the object subtracts one from the reference count. When all clients have finished using all of an object's interfaces (when the reference count falls to zero) the object commits suicide, freeing any resources it has been consuming.

With DCOM there are some concerns with reference counting. One problem is that if a client calls AddRef and Release often (which is typically the case), a lot of network traffic will be generated. Another problem is that when a client goes down the remote objects, that the client holds, must in some way be released. The first problem is solved by keeping an internal, per host reference count. It doesn't matter how much the count is as long as it is more than zero. So the only time when a reference call is sent out is when the count goes down to zero. The second problem is solved using pinging. The client sends a "still alive" packet to the server every two minutes. If the server doesn't receive such "still alive" ping for three periods (3 * 2 min) in a row, the client is considered dead. If that is the case, the object decreases the reference count with as many Releases as needed.

Pinging every object the client holds a reference to would put a high and unnecessary load on the network. Therefore, the DCOM infrastructure automatically sets up ping sets consisting of every client/referenced object combination on each pair of machines. The whole ping set is then kept alive with a single packet sent every two minutes between those two machines.

7.3.3 Local and remote transparency

When writing code for a client, one does not have to consider whether the object to call is in-process, local out-of-process or remote. This is due to that the client sees all objects as in-process objects. How is this achieved?

When an in-process object is involved, COM can simply pass the pointer directly from the object to the client. Calls through that pointer end up directly in the object code. Thus, the in-process case is a fast calling model – the time of an in-process call is the same as if raw DLLs were used.

Passing the object's exact pointer value to *other processes* when local or remote objects are involved is not possible. To still achieve transparency of location, marshalling is used. Marshalling forms the necessary structures for interprocess communication. Marshaling is the process of packaging up object method requests into a transmittable packet and then sending it to the server. The location transparency is achieved using proxies (residing in the client's process) and stubs (residing in the server's process). Requests for remote invocations on objects are marshaled to the proxy, which in turn forwards them to the corresponding stub, which then invokes the method on the requested object. Returning of results are performed the same way. The proxy and the stub handle the information of how the communication between them is handled for an interface (e.g. if big endian or little endian is used).

COM creates the stub in the server's/object's process and has the stub manage the "real" interface pointer. COM then creates the proxy in the client's process and connects it to the stub (see SCM – the Service Control Manager below).

This proxy/stub setup as being the same architecture as the one used in raw RPC with the difference that both the proxy and the stub are called stubs. In Java they are called stub and skeleton, respectively.

A proxy does not contain the actual implementation of an interface. Each member function in the proxy packages the arguments it receives from the client into a packet and passes that packet to the stub using RPCs. The stub unpacks these arguments, pushes them on the stack and invokes the call on the real object (using the interface pointer the stub is managing). The object executes the function and returns its output. The output is packaged up and sent back to the proxy, which unpacks output and returns it to the client. Thus the client only sees in-processes and transparency is achieved.

Proxies and the stubs are automatically created when the MIDL compiler is used to compile IDL interfaces (see section 7.4).

7.3.4 SCM – the Service Control Manager

The responsibility of SCM is to find and activate objects. In the activation process, it binds the initial interface pointer and hands it over to the client. After this initial work is performed, SCM fades away and rests back, it is not involved in any communication between the client and the server.

7.3.4.1 The work of SCM

When a client requests an object to be created, COM contacts the SCM on the local machine. The SCM then looks into its locally held database, the system registry, for information about the object server. It will then launch the server, obtain an interface from the server and pass it back to the client.

If the object is an in-process server, the SCM does not need to activate the object server, as it is run under another executable. Thus, the only favor required from the SCM is that it obtains the path to the server, which is then used by the requesting process.

If the object is a local server (not in-process), the SCM also activates the object (the executable).

In case the server is on a remote machine, the local SCM will have to contact the SCM on the remote machine that has the object. The remote SCM will then launch the server. Depending on whether the server is in-process or out-of-process (a DLL or an EXE), the launching is performed differently. In case of an out-of-process server, the server is just launched, nothing more. But, if the server is in-process, then a surrogate executable must be started by the remote SCM. This surrogate then loads the DLL.

7.3.5 GUIDs – Globally Unique Identifiers

To maintain uniqueness to static entities in COM, such as implementations or interfaces, GUIDs are used. A GUID is exactly the same as to what OSF defined a UUID (Universally UID) to be, the only difference is that Microsoft did not want to go as far as to declare it “Universal”. A GUID is a 128-bit long identifier declared as being unique in both time and space. It is generated using an algorithm that uses a machine’s (the machine at which the GUID is calculated) hard-coded network card value (if one is installed, and if not, a random number is generated) and the current time when the algorithm is run. Thus, it can be said to be both unique in both space and time as network card numbers are unique.

The possibility that two GUIDs are the same is minimal. Kraig Brockschmidt gives a somewhat concrete example of the number of different GUIDs that can be created in [2]. He states that with 128-bits there can be $3.4*10^{38}$ different combinations, and that it would take 196 million years to run out of GUIDs if every one in the world generated ten trillion GUIDs per second. Even though it is not a calculation of the probability that two GUIDs ends up the same, the use of GUIDs can be thought of as quite safe.

7.3.6 Libraries

For the development of COM, a new library, the Active Template Library (ATL), has been introduced. Before the introduction of ATL, the only available help for development of COM in C++ was MFC (Microsoft Foundation Classes). However, the Standard Template Library (STL) could be used for raw COM programming in C++, but that means a lot of unnecessary and hard work, which in ATL and MFC is performed by “Wizards”. It should also be mentioned that both ATL and MFC are C++ specific.

ATL was created to answer the need for lightweight and fast ActiveX COM components. ATL consists of a set of template classes. MFC works great for providing an Automation interface on applications, i.e. the IDispatch interface (which is used if programming in a typeless language as Visual Basic) is automatically included, without requirements of knowledge of it from the developer. MFC is enhanced for user interfaces, thus if writing a UI server, MFC is considered the best choice. But a problem with MFC is that it is hard to extend the number of supported interfaces. If custom COM interfaces should be added into the class problems arise. The reason is that MFC C++ manipulates the underlying structure (the vtables) and sometimes turn into assembler in trying to make programming easier for programmers. This could be a problem if not only the standard interfaces are going to be used. MFC also brings in a huge overhead (in form of code for unnecessary classes).

ATL on the other hand is a template library, and as such the developer can chose to use as much or as little as is needed. With ATL, small, lightweight servers can be created; servers that are easy to use and modify. ATL was only designed for writing COM servers. It is however also possible to integrate ATL with MFC. This is quite important as ATL is limited to the creation of COM-based objects only and lacks in generic and utility class support. Thus, combined with MFC and the Standard Template Library (STL), ATL is a powerful framework for ActiveX development as it silently brings the features of the IDispatch interface. If writing core COM components without GUIs, ATL is the best choice.

7.4 IDL and the MIDL compiler

COM IDL is based on the DCE-RPC IDL. DCE IDL allows remote procedure calls to be described in a language neutral manner that also enables an IDL compiler to generate networking code that transparently remotes the described operations over a variety of network transport protocols. COM IDL simply adds a few COM-specific extensions to DCE IDL to support the object-oriented nature of COM (e.g. inheritance and polymorphism). An IDL compiler called MIDL parses COM IDL files and generates several useful items. One of the things that MIDL generates is source code that allows the interface to be used across thread, process and host boundaries. MIDL generates C code but the MIDL can also generate a binary file that allows other COM-aware environments to produce language mappings for the interfaces defined in the original IDL file. This binary file is called a type library and contains tokenized IDL in a parsed form (that is supposed to be very efficient). Type libraries are typically distributed as part of the implementation's executable and allow pointer aware languages (described under 7.3.1 Interfaces) to use the interfaces that are exposed by the implementations.

An example of a declaration of a method in an IDL file:

```
HRESULT Method([in] short arg1,
               [out] long *arg2,
               [in,out] wchar_t **arg3);
```

Actually, there are some attributes that need to be inserted above the code and that is required for the definition of the interface of which "`Method`" is a member. For example, such a required attribute is the declaration of the interface's GUID.

The [in]s and [out]s declare if the argument is an in-argument or if it's an out-argument. The COM methods should always return a HRESULT from a call, which contains information if something went wrong. To allow languages like Java and Visual Basic that does not use pointers to make use of COM another tag, [retval], is used to declare that the value is returned. In Java, the HRESULT is not used as the return value, rather is an exception thrown if the method fails.

7.5 Security

Important questions to answer, when designing a distributed application are: Who can access which objects? Which operations are objects allowed to perform? How can administrators manage secure access to objects? How secure does the content of a message need to be as it travels over the network?

Different platforms use different security providers and many platforms even support multiple security providers for different usage scenarios or for interoperability with other platforms. DCOM and RPC are architected in such a way that multiple security providers can be simultaneously accommodated.

Common to all these security providers is that they provide a way to identify a security principal (typically a user account), a way of authenticating a security principal (typically through a password or private key), and a central authority that manages security principals and their keys. If a client wants to access a secured resource, it passes its security identity and some form of authenticating data to the resource and the resource asks the security provider to authenticate the client. Security providers typically use low-level custom protocols to interact with clients and protected resources.

7.5.1 Security policies

DCOM distinguishes between four aspects of security:

- **Access security -** Which security principals are allowed to call an object?
- **Launch security -** Which security principals are allowed to create a new object in a new process?
- **Identity -** What is the security principal of the object itself?
- **Connection policy –** Integrity: can messages be altered? Privacy: can messages be intercepted by others? Authentication: can the object find out or even assume the identity of the caller?

7.5.1.1 Protecting the object: Access security

The most obvious security requirement on distributed applications is the need to protect objects against unauthorized access. Sometimes only authorized users are supposed to be able to connect to an object. In other cases, nonauthenticated or unauthorized users might be allowed to connect to an object, but must be limited to certain areas of functionality.

DCOM provides declarative access control on a per-process level. Existing components can be securely integrated into a distributed application by simply configuring their security policy as appropriate with the DCOMcnfg tool[1]. New components can be developed without explicit security awareness yet still run as part of a completely secure distributed application.

If an application requires more flexibility, objects can programmatically perform arbitrary validations, be it on a per-object basis, per-method basis, or even per-method parameter basis. Objects might also want to perform different actions depending on who the caller is, what specific access rights the caller has, or which user group the caller belongs to.

7.5.1.2 Protecting the server machine: Launch security

Another related requirement on a distributed infrastructure is to maintain control of object creation. Since all COM objects on a machine are potentially accessible via DCOM, it is critical to prevent unauthorized users from creating instances of these objects. This protection has to be performed without any programmatic involvement of the object itself, since even launching the server process could be considered a security breach and would open the server to denial of service attacks.

The COM libraries thus perform special security validations on object activation. If a new instance of an object is to be created, COM validates that the caller has the privileges required to perform this operation. The privilege information is configured in the registry, external to the object.

7.5.1.3 Controlling the object: Security identity

Another aspect of distributed security is that of controlling the object. Since an object performs operations on behalf of arbitrary callers, it is often necessary to limit the capabilities of the object. One obvious approach is that of making the object assume the identity of the caller. Whatever action the object performs: file access, network access, registry access, and so on, is limited by the caller’s privileges. This approach works well for objects that are used exclusively by one caller since the security identity is established once at object creation time. The approach can also be used for shared objects if the object performs an explicit action on each method call.

However, for applications with a large number of users, the approach of making the object assume the identity of the caller can impose problems. All resources that are potentially used by an object need to be configured to have exactly the right set of privileges. If the privileges are too restrictive, some operations on the object will fail. If the privileges are too generous (i.e. there is write access to some files where only read access is desired), security violations might be possible. Although managing access can be simplified by defining user groups, it might often be

[1] The DCOMcnfg tool is shipped with Windows 9x and NT.

simpler to have the object run under a dedicated security identity, independent of the security identity of the current caller.

Other applications may not even be able to determine the security identity of the caller. Many Internet applications, for example, do not assign a dedicated user account for every user. Any user can use the application and yet the objects still need to be secure when accessing resources. By assigning objects a security identity of their own makes this kind of application manageable in terms of security.

7.5.1.4 Protecting the data: Connection policy

As the "wire" between callers and objects become longer, the possibility of data that is being transported as part of method invocations being altered or intercepted by third parties increases. DCOM gives both callers and objects a range of choices to determine how the data on the connection is to be secured. The overhead in terms of machine and network resources gets bigger and bigger with the level of security. DCOM therefore lets applications dynamically choose the level of security they require.

Physical data integrity is usually guaranteed by the low-level network transport. If a network error alters the data, the transport automatically detects this and retransmits the data.

However, for secure distributed applications, data integrity really means being able to determine if the data actually originated from a legitimate caller and if it has been logically altered by anyone. The process of authenticating the caller can be relatively expensive, depending on the security provider and the security protocol it implements. DCOM applications can then choose if and how often this authentication occurs (see section 7.5.1.5 below for details).

DCOM currently offers two fundamental choices with regard to data protection: integrity and privacy. Clients or objects can request that data be transferred with additional information that ensures data integrity. If any portion of the data is altered on its way between the client and the object, DCOM will detect this and automatically reject the call. Data integrity implies that each data packet contains authentication information.

However, data integrity does not imply that no one can intercept and read the data being transferred. Clients or objects can request that all data be encrypted. Encryption implies an integrity check as well as per-packet authentication information, and privacy is thus achieved.

7.5.1.5 Identifying the caller: Authentication

The above mechanisms for access control, launch permission control, and data protection require some mechanism for determining the security identity of the client. This client authentication is performed by one of the security providers, which returns unique session tokens that are used for ongoing authentication once the initial connection has been established.

Keith Brown, [7], explains how the Windows NT authentication (NTLM – NT LAN Manager) works giving an example:

Windows NT 4 uses a mechanism known as NTLM authentication. NTLM is a simple challenge/response protocol that avoids sending a user's password across the wire. One caller (**A**lice) contacts an object (**B**en) and indicates that she wants to be authenticated via NTLM. **B** sends a random 8-byte number (the challenge) back to the caller, asking her to prove her identity. **A** uses this challenge and her password to create a 24-byte response (via various cryptographic functions), and send this back to **B** along with her name (principal) and the name of her domain controller (authority). **B** presents all this information to the Local Security Authority (LSA) on his machine and asks for proof of **A**'s identity in the form of a security token. The LSA routes the information to the correct domain controller, via trust relationships. The domain controller eventually performs the same calculation as **A** (it is the only other entity that knows **A**'s password, or more precisely an MD4 hash of the password) and either verifies **A**'s identity or tells **B** that the caller is a fraud. If **A** is verified, the NTLM security provider generates a unique access token, which it returns to **A** for future use. For future authentication, the client can simply pass in the token, and the NTLM security provider would not perform the extra round trips for the "challenge/response" protocol.

DCOM uses the access token to speed up security checks on calls. However, to avoid the additional overhead of passing the access token on every call, DCOM by default only requires authentication when the initial connection between two machines is established. It then caches the access token on the server side and uses it automatically whenever it detects a call from the same client.

For many applications this level of authentication is a good compromise between performance and security. However, some applications may require additional authentication on the calls. Often, certain methods in an object are more sensitive than others. An online shopping mall might for example only require authentication on connection establishment as long as the client is only calling methods for "browsing" the shopping mall. When the client eventually orders goods and passes in credit card information, the object might require calls to be individually authenticated.

Depending on the transport used and the size of the method data to be transmitted, a method invocation can require multiple data packets on the network. DCOM lets applications choose whether only the first packet of a method invocation should contain authentication information or if each packet should be individually authenticated.

As discussed in the previous section, authentication, integrity, and privacy are tightly related. For this reason DCOM defines a single sets of constants that convey the level of authentication and privacy. These constants are the same constants as defined for DCE RPC and are listed in table 2.

Authentication Level	Description
None	Does not perform any checking or verification of the caller’s identity.
Default	Uses the Windows NT default authentication level, which is currently *Connect*.
Connect	Checks and verifies the caller’s identity on only the first call, when a connection is made to the server.
Call	Checks security on every call.
Packet	Encrypts the caller’s identity to guarantee its authenticity. Encryption prevents a user from successfully impersonating another.
Packet Integrity	Similar to *Packet*, but checks the network packets to provide the highest possible level of security.
Packet Privacy	Authenticates all previous levels and encrypts the argument values of each remote procedure call.

Table 2. ***The Windows NT authentication levels.***

Since DCOM enforces the authentication policy on behalf of the object, the object needs to indicate the authentication level it is willing to accept. This can be done either programmatically or via external configuration.

7.5.1.6 Protecting the caller: Impersonation levels

A more subtle implication of security in distributed applications is the issue of protecting callers from malicious objects. For Internet applications in particular, this is a critical concern. Since DCOM allows objects to impersonate callers, objects can actually perform operations they do not have sufficient privileges to perform alone. To prevent malicious objects from using the caller's credentials, the caller can indicate what it wants to allow objects to do with the security token it obtains. The following options are currently defined:

- **Anonymous** - the object is not allowed to obtain the identity of the caller. This is the safest setting for the client but the least powerful for the object.
- **Identify** - the object is only able to detect the security identity of the caller (that is, the user name), but can not impersonate the caller. This call is still safe for the client in that the object will not be able to perform operations using the security credentials of the caller. However, the client's user name will be disclosed to the object.
- **Impersonate** - the object can impersonate and perform local operations, but it can not call other objects on behalf of the caller. This mode is possibly a bit unsecure for the caller, since it allows the object to use the client's security credential to perform arbitrary operations on the machine where the object is running.
- **Delegate** - the object can impersonate the caller and, in addition, it can perform other method invocations using the security identity of the caller. In this mode, the caller essentially delegates ownership of its security identity

to the object so that the object can perform arbitrary—including remote—operations using the caller's security identity. In Windows NT 4.0, no security provider supports delegation.

These options are defined as part of the Windows NT security infrastructure.

DCOM allows these settings to be both programmatically controlled and externally configured.

7.5.2 Windows NT security infrastructure

As described previously, DCOM ties into the extensible Windows NT security infrastructure. This section describes some aspects of the Windows NT security infrastructure.

Whenever DCOM needs to know the identity of a caller, it asks the security provider to authenticate the caller. The security provider returns an access token, which contains a security identifier (SID) that uniquely identifies the authenticated security principal.

DCOM then uses the security APIs to find the caller's security token in a list of security principals. This list of security principals is called a Discretionary Access Control List (DACL). Access control lists consist of multiple Access Control Entries (ACE), which correspond to individual security principals. Each access control entry can indicate that the corresponding security principal is to be allowed access or that it is to be denied access. The security APIs traverse the list of access control entries in the access control list. If a security principal matches an access-denying ACE, the caller is assumed not to have access. If a security principal matches an access-allowing entry, the caller is assumed to have access. Access-denying entries are usually placed before access-allowing entries. The DACL is combined with another SID that indicates the owner of the list, and with another list of ACEs. This other list of ACEs is called the System Access Control List (SACL). Its purpose is to indicate whether access to the list should trigger an audit event. The current implementations of DCOM do not implement auditing and simply ignore the SACL.

The combination of owner (owning SID), list of principals (DACL), and list of principals to audit (SACL), is called a security descriptor (SD): a security descriptor completely describes the security policy in terms of object owner, access, and auditing.

The security descriptor (in its self-relative form) is a memory structure that references its elements (DACL, SACL, and so on) using offsets instead of pointers. This self-relative security descriptor can thus simply be written into a registry key and safely retrieved.

7.5.3 Designing for security

DCOM provides multiple choices to secure applications. On one end of the spectrum, DCOM can enforce security without any cooperation on behalf of the object or the object's caller— the security settings for an object can be externally configured and DCOM enforces them automatically. On the other end of the spectrum, DCOM exposes its entire security infrastructure to the developer so that both clients and objects can obtain complete programmatic control over their security policies.

Application designers can choose whichever mechanism is most appropriate for the specific application they are designing. Both approaches have specific advantages. Keeping the security policy externally configurable provides more flexibility at deployment time, because the same binary component can be used in different applications or in different environments that require different security policies. However, the security configuration needs to be provided at deployment time and has to be correct for the application to work properly. Programmatically controlling security policies provides additional flexibility to the developer, but hard-codes certain security decisions into the components or their clients.

Except for the ones already mentioned, valuable discussions on the COM/DCOM topic and useful resources can be found in [2], [8], [9], [10], [11], [12], [13] and [14].

8 MTS – MICROSOFT TRANSACTION SERVER

MTS provides an infrastructure for component based distributed applications, using DCOM as its communication mechanism. The primary goal of MTS is to let developers write COM-based servers that are still powerful and scalable. Benefits that MTS provides to component developers are:

- Provides support for transactions.
- Eases threading concerns.
- Simplifies security.
- Simplifies the deployment of client/server systems.
- Provides scalability through JIT (Just-In-Time) activation and object and resource pooling.

8.1 Transactions

A transaction is a logical unit of work that is either performed in its entirety or not performed at all. When talking about MTS, transactions always involve databases and changes to them. Transactions have four essential properties which are:

- **A**tomic: The transaction happens indivisibly to the outside world (either commit or abort).
- **C**onsistent: The transaction does not violate the system invariants (that shall/can not be changed).
- **I**solated: Concurrent transactions do not interfere with each other.
- **D**urable: When a transaction commits, the changes become permanent.

These properties are called the **ACID** properties.

A transaction should function as a consistent and reliable unit of work for the database system and must guarantee the ACID properties when executing in a consistent database.

8.2 Threading

Multiple threads per process are used to achieve high levels of scalability. This approach enables a single server to satisfy several simultaneous client requests. The MTS runtime system employs and handles a pool of threads for the convenience of the developer. The developer is discouraged to create threads within components designed for MTS. The primary task for MTS threading, is to let the developer delegate that kind of work to MTS. This means that the developer only have to write the components as though they were single threaded and then let MTS handle the threading (which depends on the threading model of the affected component). If an object does thread, MTS will suspend that thread and any advantages of creating that addition will be lost.

Each object created within MTS is created in an activity. An activity is a distributed logical thread. This means that although MTS objects may be running on different machines, and thus in different threads, MTS ensures that they behave as if they are running in a single thread. Any object created within the activity will be synchronized with all the other objects within the activity, regardless of what actual machine it is running on. MTS does allow for there to be more than one activity on a single thread to cut down on the number of threads created. However, in this case calls into an activity are still serialized.

COM supports three different threading-models for servers today:

- Single threading
- Apartment threading
- Free threading

In addition to this one more model is defined, but not yet implemented: the Rental or Hotel threading-model.

8.2.1 Single threading

Let the program execute as a single thread that services all component instances contained in the server.

8.2.2 Apartment threading

Use more than one thread for individual component instances. However, a single thread must still service all requests for an object.

8.2.3 Free threading

Any number of threads can execute any object’s methods at any time in any order. Multiple threads can execute within the same method at the same time.

8.2.4 Rental/Hotel threading

Rental/Hotel threaded servers enable multiple threads to run within a single component instance, but only one thread executes within any given method.

The free threaded model is the most flexible and scalable of them all, but it is difficult to program with it. The rental/hotel model is not as flexible as the free threaded model, but it should be easier to write programs with rental/hotel threading compared to free threading. Apartment threading is the default threading model for COM. It is not very flexible, but it is easy. Single threading can be used when it is of highest importance that a thread is only serving one object at a time.

8.3 Security

Security is simplified with MTS as it uses the concept of *roles* and *packages*. A role defines a logical group of users and separates the application’s security requirements from the operating system’s underlying security model. Even though Microsoft did not design roles as being the same as users (in the Win NT environment), the use of roles makes it easy to administer access to an application and use programmatic security. A package is a collection of components that make up an application. The security can then be defined for the whole package rather than its individual components.

8.4 Deployment

MTS provides, via the Transaction Server Explorer, a concept of bundling components together into what are called packages. The nice feature of packages is that the properties of all the members of a package can be declared for the whole package as one (see **Security** above), instead of having to declare the properties for each component in the package. Another nice feature is that a package is operating in its own protected address space in memory, apart from other packages, thus providing an additional level of failure isolation and data protection. Unreliable components (provided by for example different companies) that are assembled together with secure components can be divided into two different packages. In that way the whole application does not need to go down if any of the unreliable components brings down its process.

When a package is deployed, all the components residing in the package are deployed together. Packages make installation easy. Once the components to reside in a certain package are defined, the components can be exported as a whole to a package file so that it can also be imported to another Transaction Server. When this is done, the necessary server DLLs, type-libraries and proxy-stubs will be copied to the export directory. The security for the package need not be configured in advance of the insertion of components into the package. However, it is wise to ensure that the components will require the same kind of security, as that is one requirement of a package.

8.5 Just-In-Time (JIT) and object and resource pooling

8.5.1 Object pooling

Although it is described here in the text, it should be said that MTS does not support object pooling today. This is due to thread affinity of the most current COM objects. Object pooling will however be supported some day in the future; when is not declared, but it will probably be together with COM+.

With JIT (Just-In-Time) activation, client throughput is optimized by ensuring that multiple clients can use middle-tier (where the business logic resides) component instances. It is done by coordinating the instances so as soon as one client's work is completed its instances are freed up to work for another client.

Just-in-time activation is used by MTS. It allows clients to hold references to MTS components for long periods with limited consumption of server resources. The client will think that it holds a reference to the actual object, but it does not. Instead, MTS puts the object in an object instance pool after the transaction is committed. If the client then invokes another method on the object (the client doesn't know that there is no ”its” object), MTS just creates or calls another instance of the object (if there are any such objects in the object instance pool present), then passes the client's call to that instance. It is clear that object pooling prevents the MTS object of keeping its state. The MTS objects should thus be stateless, even though the client issuing the MTS object can have state.

Object pooling increases the efficiency of activating and deactivating objects, as opposed to creating and deleting objects each time they are needed to perform some work. In [19], Don Box states out: Object pooling is based on spending bits to save cycles; it assumes that the memory consumed by an object in the pool is less costly than the cycles required to create a new object from scratch.

8.5.2 Resource pooling

MTS provides pooling and automatic management of two critical server resources: threads and database connections. A pool of threads is employed and handled by MTS, to make the components respond to client requests quickly. Then MTS delegates free threads to clients when needed.

A database connection is a representation of a communication link between a client application and a database. The pooling of database connections is very important to achieve scalability, but this requires that the components used are stateless. The need for stateless components is probably best explained given some numbers:

A connection to a Microsoft SQL Server database (the database most often used with MTS) takes about one second or two to set up. One connection consumes about 50K of memory on the database server. With stateful components, each component will have to be connected for every use (initially) and one connection is required for each component. If 1.000 persons are using one component that needs to be connected to a database, 50MB of RAM is required to maintain the connections.

Thus, stateful components do not scale very well, and they should therefore be avoided when MTS is concerned.

The pooling of database connections that is managed by MTS enables many users to access a database with only a small set of database connections. However, to reach highest availability on the database connections the connections must be released immediately when there is no use of them. The responsibility of releasing connections lies on the client.

The database connection pooling can be used to let more users utilize a database than there are licenses for database connections available.

8.6 How MTS works

8.6.1 The context object

A client creates a component and MTS creates a context object for this component. This context object has information about the object security, threading model and its transaction. The client will never have to worry about the context object as MTS handles everything about it. There is one context object per interface of an object in MTS. The context wrapper is used to receive and dispatch calls. Since MTS always sits between the client and its objects, MTS can enable transactions and to optimize these calls by intercepting the calls (see chapter 15). It is through the context object the client aborts or commits transactions.

One responsibility of the context object is to cache type libraries. This means that when an object is deactivated and reactivated at a later stage the context will still have the type library, which leads to a performance gain, especially if there are many client calls on the same object.

8.6.2 MTS components

MTS components are COM components. Even so, there are a few restrictions on a MTS component. The most important is that the component can only be in-process, i.e. a DLL. This is due to that in-process components make it possible for MTS to load it in the appropriate server context. Therefore the components that are to be run by MTS have to reside on the same physical machine as the MTS server.

8.6.3 MTS Transactions

MTS isn't really what it's name indicates since much of what MTS provides has nothing to do with transactions. Instead, a large part of the MTS functionality focuses on making it easier to build scalable COM servers. However, support for transactions is a requirement for many kinds of services, and it is certainly an important part of what MTS offers.

There are two types of MTS-based objects: transactional and non-transactional. A transactional object is an object that takes part in a transaction, whereas a non-transactional object is not taking part in a transaction.

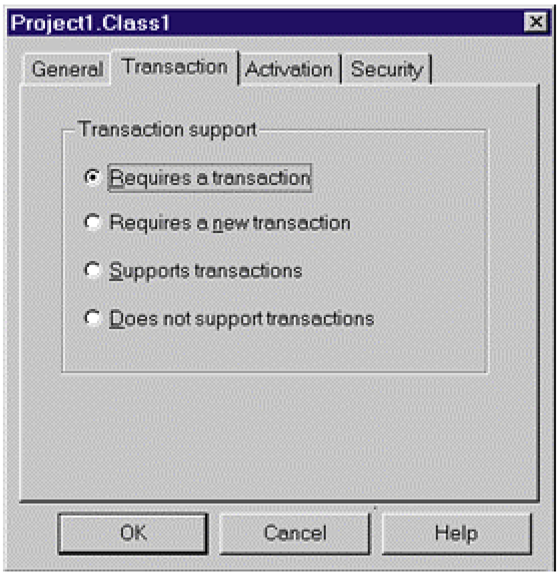
The transaction models of classes can either be set in the IDL or administered with a MTS administrator tool named MTS Explorer (Figure 2). The possible transaction models are:

- Does not support transactions
- Supports transactions
- Requires a transaction
- Requires a new transaction

Classes marked *"Does not support transactions"* always produce non-transactional objects. The object is not run within the context of a transaction, but it does have a context object.

Classes marked *"Supports transactions"* produce transactional objects if they are created by a transactional object, but produce non-transactional objects when created by a non-transactional object. The object is run within the context of the client's transactions.

Classes marked *"Requires a transaction"* or *"Requires a new transaction"* always produce transactional objects. *"Requires a transaction"* imposes that the object must execute in the context of a transaction, and if the client does not have a transaction, one will be created. *"Requires a new transaction"* means that the object must execute in the context of its own transaction, therefore a new transaction context will always be created.



Transaction support

● Requires a transaction
○ Requires a new transaction
○ Supports transactions
○ Does not support transactions

Figure 2.

Why let a class that does not support transactions make use of MTS then? Well, as stated earlier, support for transactions is only one benefit from MTS, other benefits are concurrency (automatic threading), security control and object dispensers.

8.7 The usage of MTS

The mission of Microsoft Transaction Server as stated by Microsoft in [15], "A Guide to Reviewing and Evaluating Microsoft Transactions Server", is explained below:

"The mission of Microsoft Transaction Server is to provide solution developers with all of the following application infrastructure services: receiver, queue, connections, context, security, thread pool, synchronization, management, and configuration, so that developers only have to concentrate on client logic, server logic, and setting up the database. This significantly lowers the cost and complexity of deploying server applications."

MTS provides a built-in architecture for three-tiered applications on Windows NT Server. MTS is a middle-tier platform for running business logic. In a three-tiered application, business-process logic is separated from presentation services (GUIs) and data services (databases). The reason to build three-tiered solutions lies in the belief that faster development, lower deployment costs, and increased flexibility can be achieved when applications are well partitioned. This should be seen as opposed to monolithic and client/server designs, where presentation, logic, and data service are quite tightly dependent on each other.

8.7.1 Hardware requirements

- A Windows NT x86 compatible computer.
- 30 Megabytes of space available on the computers hard disk.
- A minimum of 32 megabytes of RAM.

8.7.2 Software requirements

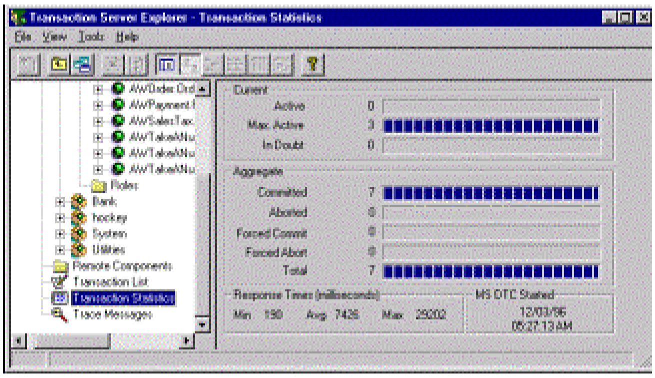
No other software requirements are needed except for MTS itself. However, if the transactional support given by MTS is going to be profitable, then an ODBC-compatible database server and possibly an Internet Information Server (for web-integration) should be considered. If not, what could be used of MTS is its support for threading, just-in-time activation (which is yet to come) and its support for management of packages and security.

8.7.3 Performance

According to Vlad Kroutik in [16], MTS does not allow applications scaling past 50 to 100 concurrent clients. However, more than one MTS can run on a server, thus multiplying the possible number of concurrent clients by the number of MTSs. This may seem like a big bottleneck to scalability, but if the clients are written in a MTS manner they should release their objects (i.e. call SetComplete) immediately after the tasks have been performed, thus freeing space for other clients. Since it is possible that clients will be written in a non-MTS manner (i.e. not committing after each fulfilled request), it is hazardous to rely on the clients to do this after every task has been performed. As MTS is used for a wide variety of tasks, not only for transactions, the committing of the client is in many situations likely not to be completed when it should, thus resulting in a decreasing number of possible concurrent clients. This is a big drawback of MTS.

8.8 Transaction Server Explorer

A graphical tool called the Transaction Server Explorer (applied in the MMC – Microsoft Management Console, see figure 3) comes along with the installation of MTS. This graphical tool can be used for a vide diversity of things such as:



Current
Active 0
Max. Active 3
In Doubt 0

Aggregate
Committed 7
Aborted 0
Forced Commit 0
Forced Abort 0
Total 7

Response Times (milliseconds)
Min 190 Avg 7426 Max 29202

MS DTC Started
12/03/96
05:27:13 AM

Figure 3. ***The transaction Server console.***

Deployment, with the ability to:

- Create packages
- Install components
- Install packages
- Export packages across MTSs
- Configure transaction support
- Configure security

Management, with the ability to:

- Manage transactions
- Monitor transactions
- Resolve transactions
- View transaction statistics
- View trace messages
- Manage components

8.9 Merging transactions

Until today, most business solutions were developed as monolithic applications by teams. The team members all worked together. It was relatively easy for them to integrate the transactional ACID properties into an application because they had a common design and could be tested through all of the different failure scenarios together.

With component development, solutions will be assembled from prebuilt components developed by different companies, it would be costly to test and link various failure scenarios together.

Using MTS makes transaction management transparent to component developers. Developers do not have to care about how the ACID properties are actually maintained, they only write their components and then use the Transaction Server Explorer declaring the components to be transactional.

To let a component participate in the transaction of the component calling it (its client component), the component need only be declared as “Requires a transaction” or “Supports transactions”. This can be done either programmatically or by using the Transaction Server Explorer. In that way the component calling them will be responsible of the transaction and many components possibly running on different systems can act as a single, atomic unit.

8.9.1 Requirements for transaction merging

To merge transactions the following is required:

- A database server with a resource manager that knows how to roll back and commit a transaction.
- A database server with a resource dispenser that knows how to talk ODBC.
- MTS must be installed on the computers containing components that are going to take part of the transaction.
- The components of the caller as well as the receiver must be members of packages (could be different packages).

8.10 MTS conclusions

It is easy to benefit from MTS, especially for the application developer. Threading, transactions, security and bringing together components into deployment packages do not have to be considered during development, but can instead be taken care of when the components are already created.

MTS applications can generally be considered appropriate when:

- Transactions are required.
- Applications and parts of applications need easily be distributed over a network (from one central location).
- A distribution of resources (performance, failure recovery, etc.) is favorable.

Furthermore, if MTS is already in use by other applications on a network, then MTS should always be considered the right way to go.

Information about Microsoft Transaction Server and how it works can be obtained from these sources: [5],[11],[17],[18],[19],[20],[21],[22],[23].

9 MS DTC – MICROSOFT DISTRIBUTED TRANSACTION COORDINATOR

To get an overview of transactions, refer to the previous chapter on MTS (chapter 8).

If a transaction spans more than just one database it is necessary to have something that ensures that the laws of a transaction are not violated. The coordinator ensuring this in the Microsoft distributed environment is the MS DTC (Microsoft Distributed Transaction Coordinator). The DTC allows applications to update multiple databases in a distributed environment while providing transaction management. Through DTC, a data modification is guaranteed to run to completion or the modification is rolled back in its entirety. For example, DTC makes sure that if a modification updates data in two servers and the second server crashes during the update, the entire transaction is rolled back from both servers. The DTC acts as an overseeing layer that is added to monitor the distributed transactions.

9.1 Resource provision

MS DTC coordinates transactions but relies on resource providers to understand particular types of data and to know how to commit or rollback changes on those resources. MST distinguishes between two different resource providers, *resource managers* and *resource dispensers*.

9.1.1 Resource Manager

A resource manager handles durable data and is a database that knows how to participate with the DTC in the distributed transaction protocol, the two-phase commit protocol [10]. The two-phase commit protocol is used to get all the enlisted resource managers to commit. Two-phase means that either all or none of the resource managers commit. In the first phase DTC sends a message to each resource manager to see if they are prepared to commit. If they are, then DTC goes into the second phase where it tells each one to commit. If any part of this process fails then the total transaction has failed. To comply with the two-phase commit a resource manager should know how to treat commit and rollback requests. Furthermore, as the data is durable and that implies that no data should be lost due to system failure, a resource manager has to make sure that data is not lost.

The durability is achieved through DTC keeping a log of all transactions and persists this on disk, recording the start of a transaction, the resource managers that have enlisted and any decisions made on the transaction. If the MTS-DTC machine does go down, then when it restarts, it can read the log to obtain the state of every transaction that it is monitoring, thus the transaction is durable.

The most popular resource managers that can converse with MS DTC are Oracle’s relational databases, MS SQL Server, IBM DB2, and the MSMQ (Microsoft Message Queue).

9.1.2 Resource Dispenser

A resource dispenser is software that manages non-durable data. The big difference between a resource manager and a resource dispenser is that in case of failure to the system, the state of the dispenser is not maintained. Thus, there are no guarantees of durability (commit and rollback). The primary goal with a resource dispenser is to share existing data dynamically.

Two resource dispensers ship with MTS. The ODBC Resource Dispenser, and the Shared Property Manager.

9.1.2.1 ODBC Resource Dispenser

The ODBC resource dispenser is responsible for managing pools of database connections for Transaction Server components. Using these pools remove the overhead of opening a connection when the client wants access to a database. Then, when a client has made it’s call, the ODBC automatically reclaim the connection from the client for use by other clients. This allows a much larger number of clients to access a given database.

9.1.2.2 Shared Property Manager

Shared properties are properties global to the server. They could be some item of data or shared object. To control the access to the shared properties, some kind of synchronization is needed. That is exactly the job of the Shared Property Manager, it manages the synchronized access to shared properties. That means it manages locks and concurrency, which enables different components to access the same data with optimal results.

The Shared Property Manager is a global name/value dictionary that allows sharing of global attributes of basic types: int, char, long, float, array (see Vlad Kroutiks article, [16]). While the Shared Property Manager is a convenient way to share one or two attributes, it might not scale if the entire application's state will be stored in the name/value dictionary. The Shared Property Manager is also not distributed across different servers. Therefore, if an application's components are distributed across multiple servers, the Shared Property Manager will not be able to share data across those servers. Additionally, users of the Shared Property Manager must convert everything into generic name/value pairs.

These sources were used as "information bases" for the MS DTC section: [10], [11], [17], [23], [24]

10 MSMQ - MICROSOFT MESSAGE QUEUE

The fundamental problem in building a distributed application is providing some way for the various parts of the application to talk to each other. One option is to use a standard Web browser client and use HTTP to move information between it and a server. Another option is to use a RPC protocol (as does DCOM). With MSMQ developers have the another option, applications that communicate through messages, thus achieving a simplified asynchronous communication model.

MSMQ also offers built-in transactional support. This means messages or message queues can participate in MTS transactions. With this support, asynchronous operations can be rolled into an existing transaction or into a new transaction that is created. Hence, the developer will both be able to offer flexibility of asynchronous messages and an insurance that the data will not be compromised by inconsistent state.

10.1 How MSMQ works

The idea of MSMQ is that an application builds a message and sends it to a queue. Another part of the application or even other applications can then read the message from the queue. To understand it, one can think of how email work. Email protocols is a way of letting people send messages to each other, whereas MSMQ is designed to send messages between applications.

Messages are sent to and received from queues. Those queues are handled by queue managers. Queue managers may communicate with one another to send messages from one queue to another.

David Chappell claims in his article [25], that the performance difference between MSMQ and pure RPC is insignificant.

10.1.1 MSMQ Clients and Server

In MSMQ there are two types of clients and one type of server.

- Servers
- Independent clients
- Dependent clients

The *server* acts as a message router. The clients rely on the server to expose the messages they put on the queues to other clients. The clients should also be able to pick up messages from the queues (if they are authorized to do so).

An *independent client* has a queue manager with its own queues. Applications running on an independent client are free to send messages at will since they can be stored in the client’s queues. The client need not be connected to a network, and if not, the messages sent by the application are stored in the client’s queues. When an independent client connects to a network, its queue manager will automatically detect this and forward those messages onto an MSMQ server. If the client is connected to a network when the application sends the message, the message is immediately forwarded to the MSMQ server. An independent client could be used on a laptop that connects to a network occasionally.

Dependent clients can like independent ones only support applications that send and receive messages. There is anyway a difference in the limitations of a dependent and an independent client. The dependent client have no queues of its own, and hence must be have online access to a MSMQ server to work. Dependent clients are intended for systems with permanent connections to a network, such as a desktop machine connected to a LAN.

10.2 Asynchronous Transactions

MSMQ combined with MTS means that transactions can be applied even though the client is disconnected from the network. This could be interesting for salespeople that work ”on the road” selling products and then updating databases via transactions when they connect to the network.

Sending and receiving messages if using MSMQ with transactions most typically involves four operations:

1. A client adds a message to a queue, perhaps updating a local database. The first transaction.
2. A queue manager moves the message from the client computer to the server queue and the queue manager gets exactly one copy of the message.
3. A server that gets the client's message from the queue processes the request and adds a message to a client response queue. The second transaction.
4. The client retrieves its response message. Third transaction.

It is not forbidden to put all the steps (1-4) into one transaction, but it is not recommended, as it would mean that if step 3 or 4 fails, then the client's message waiting on the queue would also disappear.

As pointed out in the MS DTC section, MSMQ can also participate as a resource manager in existing MTS transactions managed by MS DTC.

10.3 When to use MSMQ

As RPC would probably be easier to use most of the time, David Chappell [25], says that the approach of using MSMQ to communicate is only appropriate if:

- The sender need not wait for a result but can usefully do something else in the meantime.
- The sending and receiving applications run at different times.
- A sender isn't sending to a specific receiver, but instead to any one of a group of receivers. As once a message has been placed in a queue it is potentially available to any application that can read from that queue.
- Requests need to be logged and possibly reprocessed to recover from failures.
- There is a need to let third party clients respond to messages. An example is if A can send a message to B which sends one to C and D, C and D can both send response messages back to A.

Articles found valuable on the MSMQ topic were [26] and [27].

11 GENERAL ANALYSIS ON DCOM

11.1 Platforms supported

DCOM was only available for Windows NT 3.51 in the early era of DCOM, but a couple of months afterwards, Windows 95 was supported as well. As Microsoft is quite focused on the PC market (i.e. Windows 95/98 and Windows NT), they have not themselves ported DCOM to other platforms than Windows. However, Microsoft understood that for DCOM to be very successful it would have to be available for other operating systems as well. Thus, they made DCOM rely on the general DCE RPC and started to work with others that wanted to port DCOM to different operating systems. That way Microsoft did not have to drag focus from its primary goal to have Windows operating on as many computers as possible. As there are always companies seeking prosperity, it did not take long before DCOM was ported to the first Unix OS. Today all the major platforms have their own DCOM implementation (Macintosh, Linux, MVS, Open VMS, Solaris, HP-UX, IMB OS/390, etc.).

11.2 Requirements

DCOM is, loosely spoken, about *remote activation*. That is, a client using DCOM to connect to a remote object will try to activate the object if it is not already active (i.e. someone else recently activated the object). There are some limitations to the remote activation, and these limitations concern Windows 95/98. To be able to activate a remote object, the remote object must reside at a machine with Windows NT. Windows 95/98 will not do since they do not support security, which is considered important when processes are launched on remote machines[1]. Remote activation can be achieved on some Unix-machines too, through the use of “DCOM-daemons” provided by a couple of vendors, such as Software AG Systems [29] and Bristol Technology [28].

According to Software AG [29], their port consists of all of the non-GUI portions of COM/OLE. Thus, there is no implementation of drag-and-drop, inplace activation, or other such functions. The basic functionality of COM for instantiating and using objects as well as Structured Storage, monikers, Automation (non-GUI parts), Custom Marshalling, Connectable Objects and Uniform Data Transfer are included in the port. MSRPC, Microsoft’s implementation of DCE RPC is also included in the port.

Software AG also claim to have support for:

- regedit (registry editor) motif application that is used to edit the registry.
- irotview (running object table viewer) similar to its namesake on NT.
- stgview (structured storage viewer) for displaying the contents of a structured storage .doc file.
- regsvr (register server) same as regsvr32 on NT.
- olecnfg (OLE configure) similar to DCOMCNFG on NT.

These points are roughly the basic requirements of a DCOM-supporting system.

11.3 Applications in which to use DCOM

DCOM is not the solution to every problem in development. Many applications do not aim at providing support for distributed applications neither need they, but in my opinion that kind of applications will decrease in number.

So, when should one choose DCOM over the other two most popular choices: CORBA and Java/RMI (Remote Method Invocation)?

- If the application is to run in a pure Microsoft environment (i.e. only Windows 95/98/NT is used) then DCOM should be the best choice.
- If the environment will consist of pure Unix machines, then CORBA would probably be the most fit.
- If the environment will be a mixture of Windows and Unix machines, then the alternatives must be balanced against each other and Java/RMI could be the most appropriate one.

[1] The work-around for the problem with not being able to launch Windows 95/98 servers is simple, but not very pretty. Let the server be manually started before any client connects to it, because once the server is launched, clients will connect normally.

11.3.1 Pure Windows environment

If Windows is used as the environment for which an application will be built then it is almost impossible to avoid using COM in the development. In these situations, COM should be used, mainly because the code would be more flexible as it can easily (if designed for change) be upgraded, extended, etc.

I believe that within 2 years, almost everyone creating new applications for Windows will do it with COM. One important reason for that would be that the development tools are more or less invisibly incorporating COM into the developers code. One advantage with using COM in Windows development as opposed to not using COM is mainly the flexibility COM brings into the development.

11.3.2 Primarily support for Windows

However, there are occasions when COM would not be that appropriate, obviously this could be the case if an application was written to run on both Windows and other platforms (e.g. Unix). This holds for many legacy applications that were initially written for Unix, VMS, etc and later on ported to Windows. What could now be the case is that the main market for the application has turned out to be the Windows environment. However, the application is still likely to support the environment(s) it was once developed for, as some of the application’s customers probably still reside on that environment(s) (and supporting other platforms than just Windows could mean market advantages).

If the application will continue to develop and its developers feel that they cannot provide the same features that competing applications have incorporated, turning to COM development could be a good idea. My suggestion would be to wrap the legacy code into COM components to be used in Windows environments, but still continue to produce raw code (no COM) when the enhancements can be used under all the platforms supported. This raw code can then afterwards be wrapped to COM.

11.3.3 Support for Windows as well as other environments

The above described is applicable if Windows has emerged to be the primary environment supported, where other environments are mainly supported to benefit the application’s market. When the application’s market cannot be distinguished to be mainly the Windows environment, then other environments cannot solely be put aside.

In this case it is hard to pick one typical approach. Java could be a good choice if it is an application that will be built from scratch. If, however, we are talking about a legacy application to which we would like to add features (e.g. distribution) the application is most probably not written in such a recently emerged language as Java.

It could be possible to wrap everything to COM, but as that requires the non-Windows environments to be extended in some way to simulate a COM/Windows environment, it would probably not be a very good choice. Although there are vendors of such “COM/Windows-daemons”, I would not recommend using that for an application that does not interoperate with a Windows environment as there are then other solutions at hand such as CORBA and Java/RMI (Remote Method Invocation). The only one that would benefit from DCOM with a non-Windows environment would be the developer, who only needs to develop one version of the application. The “daemons” are primarily developed to enable the DCOM protocol for utilization across environment boundaries and surely they do slow things down as they are basically simulating (part of) the Windows environment (I have not tested any such daemon though, and neither have I seen any report on the subject).

The best solution might then be to design and (re)develop the code by dividing up the code in parts, based on points of interconnection. This would result in the code being well-defined and thus easy to port to COM to be used on Windows as well as using it on other environments. GUIs anyway (if needed), will also in the future have to be developed on a per environment basis.

11.3.4 Support for a mixture of Windows and other environments

When the application is to be deployed on a mixture of Windows and other environments the answer to what technology to use is not straightforward.

A feasible approach can be to use the Java and it's RMI (Remote Method Invocation) architecture. Java works on every environment with a JVM (Java Virtual Machine) installed, thus using Java indicate a simple and straightforward solution, as no more than one architecture will be involved. If the application concerned is a legacy one this approach will not work very well. JavaBeans cannot easily wrap code written in another language than Java. Thus, a legacy application would have to be developed from scratch or another approach is needed.

Another approach would be to use CORBA for the non-Windows environment, COM for the Windows environment and then use some bridging software to glue them together. For this solution to work three different systems must be used: CORBA, DCOM and the bridging software which probably equals problems.

Yet another approach would be to use DCOM for both the environments. The advantage of this approach would be the same as for using Java, i.e. it is an unanimous solution.

12 THE USAGE OF DCOM

12.1 How

Up until today it has not been very common that program developing companies create general-purpose applications built on DCOM. A more common area for the DCOM technology has been specialized solutions, developed for company internal purposes, often with some kind of connection to the Internet and databases. Those Internet-solutions mainly utilize DCOM to act as the remoting layer between the web-server and the database. Examples of two companies showing where DCOM typically has been operating are SAS (Scandinavian Airline Systems) and JABR Technology. SAS has used DCOM, combined with IIS (Internet Information Server), MTS, and SQL database, as the connection protocol for their new services on Internet. JABR Technology's application Synapse[1], make heavy use of DCOM as the underlying structure. These two examples describe how DCOM has been used: as an internal connection working behind the scenes to serve Internet clients and as a special-purpose product.

DCOM can be used to build solutions spanning the Internet with HTTP-tunneling, but the support for such development is not very rich. Microsoft, or rather some of it's developers, have promised better support for DCOM over the Internet in the near future. The problems mainly concern firewalls[2] [30] and authentication/security.

The trend will most certainly be for DCOM to move into the general-purpose area of development, benefiting the "ordinary" user. Possibly, many of those solutions will be using DCOM without really bringing any new outrageous features that would not have been possible to deliver before, as almost every feature of DCOM has been available to achieve using only legacy techniques. DCOM brings faster development, and (hopefully) less "buggy" applications. Even more important is that a big part of those solutions would have been so difficult and time-consuming to develop that they would never have seen the day without DCOM. It will consequently profit the "ordinary" users as the programs they are using are going to be developed to be able to take advantage of the good that comes with distribution.

As web-enabled applications will be increasingly based on components, the division between development for browsers and workstations will vanish. COM/DCOM will find a strong market in this field.

DCOM is, together with Java/RMI, showing the way into a new era in programming. Earlier it was almost solely big and expensive applications that could use distributed programming techniques, but with DCOM and Java/RMI distribution is being/will also be brought to the smaller applications.

There should also be something said about CORBA, as it is a widely spread specification. CORBA is not competing with DCOM and Java/RMI in the small application market. CORBA traditionally run on mainframe machines and not PCs[3], which makes applications based on CORBA not very well fit to reach the "ordinary user".

12.2 Who

The first DCOM specification came in 1996. It has since then, mainly due to the introduction of ATL (Active Template Library), come to be significantly easier to develop solutions based on COM/DCOM. This especially holds for the release of ATL 3.0 and Visual C++ 6.0 and also Visual J++ 6.0. As COM/DCOM is built upon C++, the greatest part of the business tier development has been in C++ with the client part mainly written in Visual Basic. This however is changing as Java is being used more widely in COM/DCOM development too.

[1] Synapse let images be sent over the Internet annotated with text and voice and was developed for doctors (by a doctor!) to let experts handle complicated task "en remote". Information can be found at http://jabr.ne.mediaone.net/.

[2] DCOM dynamically allocate the ports it is using. Thus, when firewalls grant an application communication through a port and that port is then not used by both outgoing and incoming traffic the firewall complains.

[3] There are CORBA implementations for Linux, and thus CORBA spans the PC market. Though, it is not very common today that "ordinary" users have Linux as their OS, but if this changes, I think CORBA could gain significant market shares in the small application area. The Linux versions for CORBA and DCOM are both available for free.

For an experienced C++ developer it takes about 2-3 months to be productive in programming COM/DCOM in C++, but a C++ novice (with prior knowledge in programming) need about 6 months to achieve the same level of productivity. Java development for COM/DCOM on the other hand (as well as Visual Basic) is basically equal to pure Java development, except that Microsoft's Visual J++ is needed for the development.

When COM+ is released (which will happen within a month of the release of Windows 2000[1], according to Microsoft) it will probably not mean that it will be easier to develop COM in C++. Rather, the focus will lie in more advanced code, achieved with less programming effort. The ordinary C++ developers will not need to dig into new COM specifications except for gaining knowledge of how metadata- and attribute-driven programming works (which will be core concepts in COM+ programming). Java and Visual Basic would be the languages that probably benefit most out of using metadata and attributes as that makes it possible to their programmers to make a more extensible use of COM and it's peculiarities.

The most popular languages used for writing COM/DCOM code are compared in table 3, below:

Task	Programming language		
	C++	**Java**	**Visual Basic**
Program speed	Very fast.	Slower than C++. The speed will increase with the evolvement of Java Virtual Machines (JVMs).	Slow.
GUI development (Client)	Quite easy with MFC. Results in a somewhat "messy" code.	Quite easy. With the introduction of Swing the GUIs look the same on different platforms too, which was not the situation before Swing.	Very Easy. VB is well suited for graphical design.
Calculations (Server – business tier)	Fast. Well suited for heavy calculations.	Middle.	Slow.
Data types	Yes.	Yes, but more limited than C++.	No, VB is a typeless language.
Support for COM	COM is designed outgoing from C++.	Support for C-specific types such as structs are missing.	The same as for Java.
COM interoperability	Yes.	Yes. Method return-value is limited to one parameter (limited in parameter passing).	Yes. The same as for Java applies to VB, plus that the interface IDispatch must be supported due to VB being a typeless language.
Difference between code for COM compared to "pure" code	Quite big. With the help of development tools (Visual C++) a lot of the plumbing is automatically handled.	Hardly any difference.	Hardly any difference.
Learning curve	High.	Quite low.	Low.
COM learning curve (with earlier experience in the "pure" language)	High. 2-3 months for an experienced C++ developer to achieve productivity.	Low. Almost the same as pure Java programming. As Visual J++ is required extra time to learn it is needed.	Low. The same as pure VB programming.
Development time	Long.	Quite short.	Short.

[1] The release date of Windows 2000 is postponed all the time. The latest report (1999-04-25) says that Windows will be released in August 1999.

Frequency of bugs	High.	Middle.	Low.
Availability of developers	Low.	Middle. The major part graduating from technical schools today know how to program in Java.	High. VB is easy to learn and taught by many institutions.
When to use in COM development?	At the business tier, when performance is important.	At the business tier, when performance is of minor importance.	At the client tier, when not to complicated calculations are needed.
Foresights **(as a COM development language)**	Will continue to be used for sophisticated COM-solutions for at least 4-5 years. Java will all the time steal more and more attention from C++ as the COM-developing language.	Will gain more and more popularity in COM development. Will finally take over from C++ to be used for sophisticated solutions.	Will continue to be used for much of the client tier development. Programmers will be able to perform sophisticated development, using metadata.

Table 3. ***Comparison of programming languages in relation to COM.***

12.3 Where - DCOM development

DCOM component development is similar to the development of any kind of component. Maturity required by businesses developing components are the same no matter if the component is a COM component or a CORBA component. Therefore the following section will focus on component-based development rather than COM/DCOM development.

Early forms of components have mainly taken the form of graphical (GUI-based) entities, which typically reside on a client. In the last years, components have begun to "grow up", not only in terms of their functionality but also in their appearance on the server side. The evolution of the component market has begun a progression that will take it from basic GUI controls to the following:

- The implementation of more sophisticated business functions.
- Server-side component "wrappers" for legacy applications and data.
- Frameworks that provide the means of easily building domain-specific applications or portions of applications.
- The componentization of entire applications to ensure modularity and easy interoperability with other applications.

Software components can form the basis for a fully distributed computing environment. In this environment, elements of the application can be updated, invoked, and executed, regardless of where they reside.

Common for businesses that develop solutions built on components is that they must somehow adjust to this new way of development. That indicates they must put effort in analyzing what components they need and will need in the future, so that they can benefit of reusing components. “Adjust” also means that they should build a repository for storing and easy access of components. The repository is especially important to larger businesses as they will have more people working with the components that will have to be able to locate the components needed, no matter of where they are physically located. The component-based development maturity is essential for these businesses.

12.3.1 Developing companies maturity requirements

Of the greatest importance to IS organizations are the economics of using components to develop software applications. The benefits of using prebuilt, tested software components to build new applications are intuitively obvious. Time and money are saved, in terms of both building and maintaining application software. Both savings can be turned into competitive advantage.

Component-based development is seen as the industrialization of the software market. Analogues to the automobile industry are common. However, the techniques of industrialization could not be applied without the factory and management infrastructures to support them. This will probably prove to be as true for software as for automobiles.

12.3.1.1 Building for reuse

There will be a need for an infrastructure to support the reuse of components and it is based on the following observations:

- Software developers have not generally been open to the idea of developing or supporting reusable components. Some of this comes from a personal desire to "own" what one develops and to use only what one creates. More of this is based on the culture of optimizing software for a particular use.
- Historically, reuse of components (and objects) has been limited to specific projects or, in some limited cases, individual groups. A means to distribute and manage components throughout a multidisciplinary, geographically separated organization is essential to extending component reuse.
- True reuse requires that the developers take a wider view of the development process when designing components. No longer can a developer consider only a particular project in determining the requirements for a piece of software.

Organizations adapting to component-based development must contain personnel dedicated to the success of components, throughout the organization. These could include:

- Reuse "evangelists" who understand and can effectively communicate to technical and upper management the benefits of reuse
- Technical managers who are "on board" and can further the cause to the key corporate decision makers
- Reuse managers whose responsibility it is to manage the reuse process, assets, and tools and to accumulate data and perform analyses to determine and monitor cost/benefit tradeoffs as they pertain to component reuse

To achieve the needs described above, a number of types of expertise, as well as software tools, are required to manage the development of applications using components. Not only do the components themselves need to be designed for reusability, but the availability of components and information relevant to their use must be made a high priority.

12.3.1.2 Tools needed

Tools are needed for the component-based development to take form and develop. These tools will be:

- ***Analysis, modeling, and design tools***. These tools allow for the modeling of component-based applications and data using visual techniques.
- ***Component construction and assembly tools***. These tools are used by developers to build software components and/or assemble these components into applications.
- ***Software configuration management tools*** (SCM tools). An important aspect of the component-based development infrastructure is the ability to manage component assets as well as components in development. Therefore, many of the capabilities of SCM tools, including versioning, will be necessary.
- ***Automated software quality tools*** (ASQ tools). As components appear on the scene, they will need to be tested. ASQ tools will be enhanced to support the testing of components, including the ability to work with applications containing components that reside on different systems on a network.

In addition to the organizational changes and additions that need to be made to support the effort from a personnel standpoint, software tools that will support component-based development, according to IDC [31], need to be procured and deployed. These tools will ensure the availability of both the components themselves and the information required to use them to all developers and managers who may need them.

12.3.2 COM development in smaller companies

Smaller companies are taking the first steps and they are already into COM development. They must be flexible in adjusting themselves to new technologies and markets and they are not as tight limited to their legacy applications, as are the bigger companies.

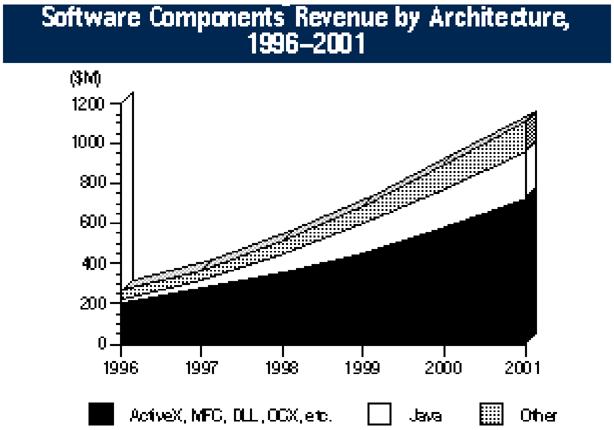
Smaller companies developing solutions built on COM/DCOM will design and develop their own components and they will not make use of much components provided by third parties.

A new market will open up for companies developing components for the COM/DCOM market and this market will be quite fast growing.

IDC's forecast for the market for software components is shown in figure 4.

The total software component market in 1996 was measured at $267 million, and it reached about $365 million in 1997 according to IDC [31]. These numbers came almost entirely from COM/DCOM components, but the fastest growing architecture is and will be JavaBeans.

By the year of 2001 the COM/DCOM component market will reach about $700 million and the market for JavaBeans about $250 million, according to IDC.



Source: International Data Corporation, © 1998

Figure 4.

12.3.3 Larger companies developing for DCOM

Larger companies are opening their eyes for development with COM/DCOM. As there is a lack of professionals skilled in programming in C++ and Java for COM larger companies will mainly have to find their COM-staff being hired consultants.

Larger companies will probably not rely on COM as much as smaller companies will, since their core architecture is not COM, but rather CORBA, or just a pure implementation. They have their legacy applications that must remain intact in the future too. However, it is possible that many of these companies will wrap their legacy Windows code to COM and their Unix (or possibly other) code to CORBA. They will then need a bridge between the COM/DCOM and the CORBA implementations.

In this sense it does not matter that much to these companies if they are relying on consultants to deliver their wrapping and bridging as it does not directly concern their core business.

Larger companies will invest in education of acquiring and assembly personnel. The primary task of these persons will be to find components to be used and then assemble the components (externally and/or internally developed) into the companies applications.

12.3.4 Consulting firms

Consulting companies are the ones that will be fastest in the acquisition of people with knowledge in COM/DCOM development. COM/DCOM developers will by a big part consist of consultants and as there will probably be a lack in personnel with this knowledge these firms will be profitable.

12.4 Where - DCOM usage

Today, solutions built on DCOM are mainly used in high-end companies, because applications with support for DCOM have primarily been built for specialized purposes.

DCOM is an architecture for businesses of varying sizes, but there are not many mission-critical applications delivered by Microsoft technology today. When Microsoft get their cluster technique (MSCS) ready, and at the same time can provide a more stable operating system than today, Microsoft will gain new customers. These customers will provide/use business critical applications with demands that cannot be saturated by Microsoft technology today. The architecture for these applications can be replaced by the cheaper and the more general technology based on DCOM, MTS, MSMQ and SQL databases. This as opposed to using highly specialized solutions built on expensive hardware, which has mainly been the only choice earlier. Applications that fit into this category are for example stock exchange systems.

12.4.1 Private user maturity

To be able to use applications built on DCOM a certain level of maturity is required by the users. Security is a bigger issue with applications based on DCOM compared to legacy applications. Legacy applications aimed at private users have not been involving Internet and technologies that potentially can screw up their PCs as easy and much as DCOM applications over the Internet will be able to do.

Private users will have to have or achieve:

- Knowledge of how to adjust security settings of DCOM-based applications.
- Common PC user knowledge.

A person could still use DCOM applications without the first point's knowledge-requirement, but sometimes there is central to the application as well as to the user that the security settings of the application be altered.

12.4.2 Business maturity

Companies have usually been involved in issues concerning security before, so that will not be anything new to them, as was the case with private users. Companies with network administrators can let them handle everything related to DCOM applications, and the employees using the applications do not have to learn anything new. Some carefulness and understanding of security related to DCOM will however be useful.

Companies need:

- Knowledge in how to adjust security settings of DCOM- and MTS-based applications.
- Knowledge in how to maintain and update databases.
- Knowledge in how to administer MTS and MSMQ.
- Knowledge in how to configure firewalls to support DCOM solutions operating over the Internet.
- Common knowledge in how to administer the network.

The points are appropriate when the companies applications make use of the technologies mentioned.

To set up applications making use of the above mentioned points consultants could be used, but after installation it will be quite practical to have employees skilled in how to administer those technologies.

12.4.3 Internet/intranet solutions

Larger companies will still have a big demand of people developing and maintaining their Internet/intranet solution(s). This work will probably also be handled by consultants or it will be outsourced to external companies. Nevertheless, this business is not central to these companies and thus, hiring consultants for it will not affect the business too much.

The grand part of companies putting effort in providing a useful (and efficient) solution for their web will stick to DCOM, MTS, MSMQ and SQL databases as their architecture, for the sake of ease and efficiency. These solutions will mainly be bought by "Internet solution companies" or consultant firms.

12.4.4 Business solutions for private users

As more and more small applications built on DCOM will evolve, buyers of those applications will automatically start to utilize DCOM.

Many of the applications with the market mainly consisting of the private user who is only using the computer for amuse and some private purposes will turn into DCOM over HTTP when support for it will become better. The network these users will connect to is the Internet. The big possible limitation to that kind of application will be security concerns as surely malicious programs and components on the Internet will frighten users.

12.4.5 Business solutions for intranets

The market for business solutions for intranets will be the fastest growing part of the two kind of business solutions. This is due to that support for DCOM on intranets is working well already, as opposed to the support for DCOM over HTTP which is not yet good.

This kind of applications will be used for a variety of things. People on the intranet will start to interoperate more and the applications will be making use of the distribution now laying at their hands. Over time, an increasingly number of sophisticated solutions to applications will be adopted.

I believe there will be an evolvement of applications built on a conjunction of MTS, MSMQ and databases, that are not solely built for the web. The target group for this kind of applications will primarily be the somewhat bigger companies. Especially those companies that have employees spread over a bit wider areas where the staff depends on the same applications (which possibly include transactions) will find this kind of applications usable. This evolution will further enhance as development for the web and workstations merge into one.

Smaller companies, although utilizing networks will, as I see it, not jump on the DCOM-ship until easy DCOM solutions for the Internet are possible. These smaller companies will reject DCOM applications in the beginning, because they will mostly be a bit more expensive than single-workstation/user solutions. When the distributed solutions have gained popularity and strength, the prices will most likely fall and the smaller companies will thus enter DCOM-wave. The reason to why these companies wait, apart from higher prices, is that they cannot see the benefit by the distributed application, as they neither can make fully use of it. This is due to that distribution is by large part used to simplify cooperation between persons, and in a smaller company the staff is often located close to each other.

12.5 When

DCOM business applications can generally be said to be appropriate when:

- More than one user of the same (kind of) application, where every user's input affects the others.
- When a distribution of resources (computer power, computers, etc.) is favorable.

Suitable models in which to use DCOM are:

- Traditional client/server applications.
- Three-tiered applications.
- Distributed applications.

Where the last two points are tightly coupled.

More and more businesses and software companies are turning to a three-tiered application model for distributed solutions. In a three-tiered application, as described in [32], business-process logic is separated from presentation services and data services. The appeal behind three-tiered solutions lies in the belief that businesses can achieve faster development, lower deployment costs, and increased flexibility when applications are neatly partitioned, as opposed to monolithic and client/server designs, where presentation, logic, and data services are tightly dependent on each other. See figure 5.

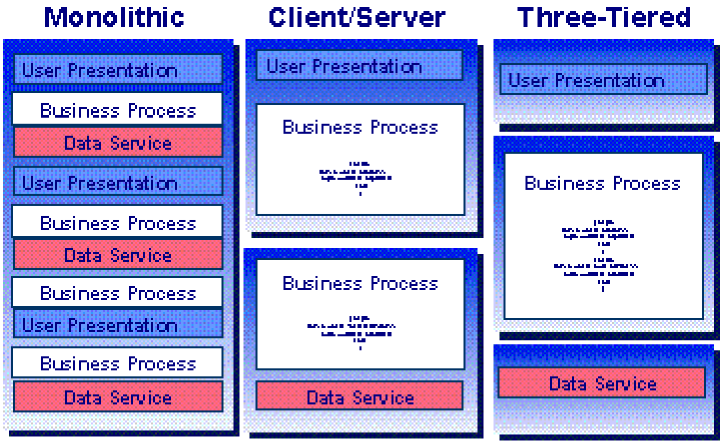


Figure 5. Evolution of distributed application models from monolithic to three-tiered

Today, applications making use of services on the Internet are limited to reside in the client tier, but there will most certainly be an extension to that in the future. In such future solutions, networks distributed over the Internet will be possible.

13 CONCLUSIONS ON DCOM

13.1 Objective advantages

13.1.1 Support

Microsoft has said that DCOM is the new way of programming and that it will support that belief to the extent that it will come true. Nevertheless, Microsoft has a great impact on the software industry and it is therefore quite possible that they are right in this statement. It is just like in the financial market with George Soros. He might not always be correct in his judgements, but he invests in such a way to make them come true. Brokers know that, and they base their investments on his statements, which in turn means that his statement finally comes true.

As mentioned above, Microsoft has said that they support this quite new way of programming (COM and DCOM), and it thus means that it will become true. Developers know that Microsoft will invest heavily in making it come true and therefore not hesitate before entering Microsoft's ship. Microsoft will continue to focus its attentions on real scalability, manageability, and low cost in the NT environment. This will benefit not only Microsoft but also many development companies sticking to Microsoft.

Another good thing about COM/DCOM is that those architectures do not have to change to enable the possibility of better solutions and extension of their use. This can instead be reached with improvements in other supporting technologies such as MTS, MSMQ etc.

13.1.2 A hyped market

DCOM is subject to become a hyped marketing argument and thus developing applications with DCOM will look good in the marketing.

13.1.3 Interoperability

Many enterprises deploy applications based on different object models just as they deploy applications on computers running different operating systems. An object model that offers interoperability with other models reduces the barriers between applications and the duplication often required to implement the same functionality in those other models. Interoperability also allows the features of one object model to fill the gaps in another model, such as distribution or platform support.

A nice feature with using DCOM is that the interoperability problems will probably always be taken care of, because DCOM is the standard for distributed applications on PCs and other architectures will be able to provide some kind of bridging to DCOM. If the ones behind the other architectures do not perform the bridging, then there will always be others throwing themselves into it.

13.2 Objective disadvantages

13.2.1 Backward compatibility

Even though Microsoft always claim that its new releases are backward compatible it does often show up to be incorrect or false. Many bugs have shown up when users have upgraded from one environment to another. Thus, developing COM components for Windows NT could mean that they are not working on Windows 2000, when it is released.

Microsoft often releases its products, even though the products are not complete. There have for example been 5 different service packs released for Windows NT4.0. Hence, there could be a problem if a program is developed on a machine with the latest service pack and then sold to someone not using that latest service pack, as the application might not work then.

When Windows 2000 and COM+ will come, legacy COM components will need to be converted to COM+. This will probably be done using COM+ converters built into COM+ development tools. The first release of such a tool was handed out to the attendees of the PDC 98 (Professional Developers Conference), which was held by Microsoft. As the tool has not been distributed outside the PDC, I have not tried it, but developers using the converter reports that it is working okay, although not perfect. In time for the release of COM+ there will probably be a more complete converter. Microsoft says legacy COM components will work just fine on Windows 2000, but this will probably not be totally correct. It might be true for most of the components but there will certainly be components that need to be redesigned a bit to fit Windows 2000.

13.2.2 Interoperability

Although Microsoft is working together with a number of companies to port DCOM to other platforms than Windows, the result is not as compelling as IS managers would like it to be. DCOM on Windows works excellent for most kind of applications or at least will probably work fine for them in the future when further enhancements are delivered by Microsoft.

A drawback for DCOM is the immature support for other environments than Windows. Microsoft does not care too much about non-Windows environments when developing DCOM. As a proof to that, support for MTS is still not provided on other platforms than Windows. Though, Software AG's EntireX [29] provides support for MSMQ.

14 DNA - DISTRIBUTED INTERNET ARCHITECTURE

Microsoft's Distributed interNet Architecture, DNA, is the evolution of Microsoft's current Win32 programming interfaces model, which is at the heart of Windows today. DNA will be based on three cores: COM+, Storage+ and Forms+. DNA is Microsoft's system for the future.

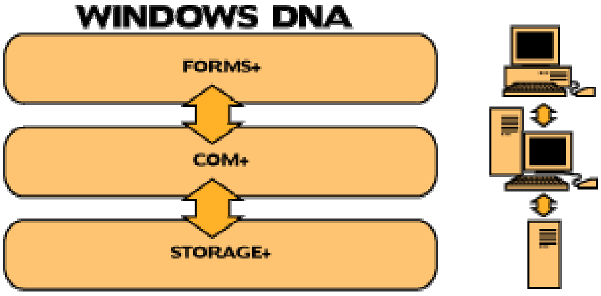


Figure 6.

With figure 6, Tom Armstrong [33] depicts the long-term goal of Windows DNA: Forms+ at the GUI level, COM+ at the middle tier, and Storage+ on the data tier.

14.1 COM+

COM+ is the successor to COM/DCOM, MTS, MSMQ and MS DTC and is discussed more in detail in the next section (COM+).

14.2 Forms+

Forms+ is another name for intelligent DHTML pages with scalable user interfaces. Forms+ will enable developers to create applications without first having to know the type of client i.e. if the user is using a browser or if it is sitting in a desktop Win32 environment (thin and rich clients).

DHTML was first supported in Internet Explorer 4.0 and is basically HTML + CCS (cascade style sheet) + client-side scripting language (e.g. VBScript or JavaScript). CSS allows authors and readers to attach a style to Web pages that defines the presentation of content by modifying style attributes of corresponding HTML elements. Applying CSS attributes to forms allows authors to make communications more accessible (e.g. for disabled persons) by using positioning (the ability to dynamically position elements on a page) and dynamic styles (such as colors, typefaces, spacing etc.) to increase visibility and ensure access to controls easier.

The next step in the forms+ evolution will take place in the DHTML engine of Internet Explorer 5.0, which is at this moment out in beta and will be included in Windows 2000. Internet Explorer 5.0 includes two key methods that enable the meshing of Win32 and Web APIs. The first one is called DHTML Behaviors and it enables developers to embed a Web component within a Win32 application in the same way as they would have added a COM object. Second, Microsoft will enable DHTML Behaviors to be scripted together with the rest of the code using scripting tools, such as Visual Basic and Visual J++ 6.0.

14.3 Storage+

Storage+ will be an integrated storage system that unifies the storage architecture for SQL Server, Exchange Server and individual machines' file systems. Like Forms+, Storage+ is not a product, but rather Microsoft's plan for how it would like its underlying storage technologies to evolve. But unlike Forms+, a vision for which Microsoft has created a prototype and a plan for marrying existing Win32 and HTML technologies, Storage+ is a lot further out on the horizon. Microsoft officials have not set any kind of timetable for delivery of the Storage+ components, thus it probably lies quite far in the future.

Storage+ will consist of a series of technologies layered on top of the NT file system. A common log, a transactional coordinator and a caching and replication mechanism will probably be featured. A record store and a transaction file system will presumably sit above these, both of which will be accessible via a common indexing mechanism. Atop this indexing mechanism there will most likely be two storage models: the societal model, which will be the repository for people, places, times, addresses and the like, and a more traditional operating system model. Some type of query processor will enable access to all the information stored in the Storage+ framework.

Storage+ will allow developers to create a structure that will use a common interface to store and access disparate data, such as Web pages, database tables, configuration information and components, from an array of stores. That is to say, it provides seamless access to all types of information via a unified storage schema.

The information for this section, were primarily obtained from these resources: [34], [35], [36], [37], [38], [39], [40].

15 COM+

COM+ is the evolution of COM. It is not yet released, but it is going to be shipped with the Beta3 of Windows 2000, formerly known as Windows NT 5. COM+ will be compatible with the old COM and DCOM will not change. The compatibility of COM+ with COM is going to be solved with tools from Microsoft (i.e. a future version of Visual C++). These tools will be used to wrap COM+ interfaces around the COM components.

15.1 Basic concepts of COM+

The primary motivation for COM+ is to achieve ease of use. Many COM developers have claimed that it is difficult to use COM in development, especially if using C++. A rare fact with COM is that there are big differences between the insight needed of COM to be able to develop COM components. C++ programmer are exposed to all the details of COM, while on the other hand developers in Java or Visual Basic easily can use COM, almost without any special knowledge of COM at all. This is due to built in support for COM in Visual Basic and the Java Virtual Machine (the one provided by Microsoft). In COM, the objects and their clients make calls on a standard COM library. In languages like Visual Basic and Java some or all of these calls are hidden, but C++ programmers use this library directly.

COM+ aims at removing the high knowledge barrier for C++ programmers by making every language tool do more work. However, it should be noted that development of COM components in C++ is not only wasted time (in comparison to Java or VB). The possibilities are greater with C++, e.g. the threading-model option given in C++ is not accessible from Java or VB.

How COM+ will facilitate the software technology is making tools do more, thus liberating the developer from a big burden. The most significant change will be within the C++ compiler that Microsoft will ship along with COM+ (or more exactly at most one month after COM+'s introduction).

15.2 Services and Interceptors

With COM+, developers are no longer required to define interfaces using IDL, The syntax of the programming language's that they are using will instead be used to define the object's interfaces. This is done through the language's compiler, which is required to work with the COM+ library to generate metadata for the object. The metadata will be used for a new concept called interceptors (also named services).

In COM today, the COM library is not directly involved when a client calls a method in a COM object. Instead, the call goes directly from the client to the object. This will no longer be true in COM+. In COM+, all method calls will pass through the COM+ library.

COM+ services are basically a move to incorporate the current technologies in MTS and MSMQ into the object programming model itself, effectively rendering them redundant. But even more services than those offered by MTS and MSMQ today are going to be present in COM+. Some services that probably are going to be used in COM+ are:

- **Event Services** – Publish and subscribe (in COM this equals connection points which imposes many roundtrips).
- **Load Balancing** – Dynamic routing of component processing.
- **Security** – Role-based security model (MTS).
- **Queued Request** – Asynchronous Services queuing services (MSMQ).
- **Transactions** – Consistent data transfer (MS-DTC).
- **Object pooling** – Efficient use of object instances (MTS).
- **Asynchronous Transactions** - Consistent asynchronous data transfer (MTS + MS DTC + MSMQ)

Interceptors will make object calls slower, so why would Microsoft like to introduce it? First, Microsoft claims the overhead that it incurs will be negligible. Second, since COM+ can be involved in every method call, it is possible to insert other objects in that path and there are likely to be many situations where the convenience and flexibility outweigh the performance decrease that they bring.

David Chappell’s definition of an interceptor is in [41] an object that is automatically invoked during access of another COM+ object. One or more interceptors can wait along the path. An interceptor might for example perform a security check on the client, then cause the method call to fail if there are not necessary permissions in place. Doing this allows keeping the authorization logic separate from the business logic in the component the client is using. An interceptor could also be used to perform load balancing, transparently distributing method calls to various equivalent objects.

Further discussions on COM+ can be found in [42], [43], [44], [45], [46].

16 DCOM VS. CORBA

DCOM, or rather MDCA described in chapter 6, is a Microsoft architecture that exists as a single implementation. CORBA, on the other hand was defined by a consortium of vendors called the Object Management Group (OMG). Many of the members of OMG have their own implementations of CORBA.

There have been some comparisons made between DCOM and CORBA. Most of them focus on which one of them are going to be the *de facto* standard, and not on how they actually perform. Since the biggest concern for development companies (which is what the papers most of the time, are written for) is what they are going to be able to sell and how much of it, this outcome is quite obvious. However, many of the writers that are concerned with magazines seem to raise DCOM (or rather Microsoft) to the skies. On the other hand, those writing scientific papers often have another opinion, claiming that the underlying structure of CORBA is better than that of DCOM, but that CORBA is a more complex system to manage.

My intention is to present an approach, which differ a bit from the usual comparisons between DCOM and CORBA. Afterwards a presentation of pros and cons of them both will be given. The work describes how DCOM and CORBA solve distributed systems problems, without declaring the “winner”. The work was mainly carried out by Imran Sayeed [47].

16.1 DCOM and CORBA solving distributed problems

DCOM and CORBA both provide a framework for creating and using components that can interface with each other – as well as with other applications, libraries, system software and networks – in a standard, well defined manner. The fundamental difference between the two approaches is that CORBA was designed to be a distributed system and support components that could exist anywhere on the network and DCOM (read COM) was not.

CORBA typically uses the Internet Inter-ORB Protocol (IIOP) for communication between components, but it is possible to use raw DCE RPC as well. IIOP is a quite new standard for CORBA that was created to remove interoperability problems between vendor specific communications. Before IIOP, it was up to the vendor to choose the communication protocol. DCOM however, uses DCE RPC for its communication across machine boundaries.

An essential part of both frameworks is the value-added services they provide to components. These services are: naming, security, persistence and data access, events and transactions.

16.2 DCOM

The value-added services associated with DCOM are MTS, MSMQ, MCS and MMC. All of them are shortly described in the MDCA section except for MMC. The Microsoft Management Console. Together they provide a set of products, libraries and SDKs (Software Development Kits) for developing business-critical applications. The following section examines how scalability, reliability, security and manageability are supported.

16.2.1 Scalability

Microsoft Transaction Server provides a set of interfaces and libraries that allow applications to easily scale as the number of users and user data increase. MTS provides automated thread management, which allows individual components to spawn multiple threads to handle increased demand without requiring developers to write tedious thread code. It also offers resource pooling and database connection multiplexing to more efficiently distribute user and data load across multiple replicated application components. This load balancing is limited to a single machine.

DCOM does not provide a centralized naming service. This essential part of a scalable architecture lets users locate a particular application or component no matter where it resides in the enterprise. This allows the application to handle an increasing number of geographically dispersed users.

DCOM will probably get the possibility to gain the Active Directory Service Interface (ADSI) with the release of Windows 2000. ADSI allows components to seamlessly use a variety of existing naming services such as NetWare Directory Service (NDS), Lightweight Directory Access Protocol (LDAP) and the Windows registry.

16.2.2 Reliability
Microsoft Message Queue Server (MSMQ), Microsoft Cluster Server (MSCS) and Microsoft Transaction Server (MTS) all help develop and maintain reliable applications. MSMQ and MTS are described in their own sections of this paper. MSCS organizes data stored in databases so that recovery from system failures is fast. MSCS automatically restarts applications and provides automated failover between server machines.

16.2.3 Security
NT LAN Manager (NTLM) and Microsoft Transaction Server authenticate users and authorize checking via Discretionary access control lists (DACLs).

There are seven levels of authentication that are used to identifying the caller: none, default, connect, call packet, packet integrity and packet privacy (refer to section 7.5.1.5 for a description of them).

The MS Crypto API provides data encryption and integrity to prevent eavesdropping and tampering while the Authenticode SDK uses digital signatures to provide non-repudiation. Other features such as Kerberos will be supported with Windows 2000.

16.2.4 Manageability
Microsoft Management Console (MMC) provides a unified graphical user interface for managing MTS and MSMQ based components. Features include centralized configuration and administration as well as remote deployment of components. MTS gathers various statistics about applications under its control. This information can be viewed in the MMC.

According to Sayeed, there are two significant drawbacks to the DCOM technology:

1. First, DCOM is essentially a Windows-only solution
2. Second, he points out that many of the value-added services are very new and are still maturing in the enterprise space.

16.3 CORBA
The value-added services provided by CORBA are collectively known as CORBA Services. Currently defined services in CORBA 2.0 include Naming, Events, Life Cycle, Persistence, Relationship, Externalization, Transaction, Concurrency, Property, Licensing, Time, Trader and Security. Of these, Naming, Events, Transaction, Trader, Security, Life Cycle, Concurrency and Externalization services are available from different CORBA vendors.

How those services assist with scalability, reliability, manageability and security is described in the preceding section.

16.3.1 Scalability
The Object Activation Daemon (OAD) and the Implementation Repository (IR), both part of the CORBA runtime, allow efficient use of resources by only instantiating objects when required. The centralized Naming service provides location independence for applications and their users, while the Trader service allows more sophisticated component searches by providing a “Yellow Pages”-like model for registration. Static load-balancing among multiple replicas of an application is available via the Naming service.

16.3.2 Reliability
Reliability is provided through the Transactions service, which supports an Open Group Distributed Transaction Processing (DTP) – compliant model for distributed transactions. Some CORBA vendors have implemented failover support for both application servers and server platforms.

16.3.3 Security
There are two levels of security within CORBA. Level 1 allows an application that is unaware of security to participate in a secure domain. It provides user authentication, authorization via ACLs, data encryption, and integrity and optional non-repudiation.

Level 2 requires applications to be security-aware, thus enforcing stronger versions of the security policies described above for Level 1. There were not any Level 2 implementations available at the time of Sayeed's writing in June 1998. There is also a "Level 0", that has been defined by some vendors. It is a Secure Socket Layer implementation of IIOP. It allows user authentication and data encryption.

16.3.4 Manageability

Some vendors have sophisticated tools for centrally configuring and administering CORBA applications. One vendor, Iona allows CORBA applications to be centrally managed from any SNMP-compliant system management console.

Sayeed depicts, that "while CORBA's main strength is allowing developers to use best-of-breed services from different vendors, this capability is also its biggest weakness. It is very difficult, if not impossible, for example, to get one vendor's security implementation to work with another transaction service. Since no vendor has a complete solution for addressing the above requirements, interaction issues are introduced when CORBA is used to build business-critical applications."

16.4 Advantages and disadvantages with DCOM and CORBA

The pros and cons in table 4, are not based on the level of technology, rather the functionality today and in the future, in the eyes of people interacting with the systems. They are the ones that finally are going to choose which architecture to use.

DCOM	CORBA
ADVANTAGES	
Microsoft has total control of the environments that it supports which means that standardization is not an issue. (Win NT, Win 95).	Many important organizations are involved with CORBA, thus a large industry that probably want to implement what they agree on.
Easy to make COM applications become distributed (DCOM), and there are many COM-applications.	Supports many different environments.
Extensive support for multiple languages.	Extensive support for multiple languages.
Language independent on a per-component level (as long as the language support pointers).	Primarily implemented on top of very reliable environments such as Unix and MVS makes it a good choice for mission critical applications.
Easy to manage and use.	Technically a better solution than DCOM.
Cheap and run on cheap computers. Large set of commercially available components.	Strong load balancing (specification).
DISADVANTAGES	
There are DCOM ports to some 10-15 different environments, but none of the ports allows MTS to be run on the ported environment.	The consortium has big problems agreeing upon standards. Thus, almost everything is up to the individual vendor to specify, i.e. interoperability problems between different CORBA implementations.
"Only" Microsoft that is backing up DCOM.	Although language independent on a per application level, it is not the same as a per component level.
The reliability of Windows NT (which is what DCOM runs on) is not considered high enough for mission critical applications.	Primarily implemented on top of expensive Unix or MVS machines. (Though, there are implementations for Linux available).
Load balancing is missing, which could be a bottleneck for large networks.	Expensive.

Table 4. ***Advantages and disadvantages of DCOM and CORBA.***

Valuable sources used for information on the DCOM/CORBA topic were [48], [49], and [50].

[1] David Sprott et al., Component-Based Development, 1998 Report Series Volume 1, Hesslewood Hall, Great Britain, April 1998. Butler Consulting Group Limited.

[2] Kraig Brockshmidt, What OLE Is Really About, July 1996, http://www.microsoft.com/oledev/dcom/aboutole.htm. Accessed: Nov 3, 1998.

[3] Bernd Krämer, Michael Papazoglou, Heinz-W. Schmidt, Information Systems Interoperability, Taunton, England, Research Studies Press Ltd., 1998.

[4] Richard Veryard. Component-Based Development, Veryard Projects, http://www.users.globalnet.co.uk/~rxv/CBDmain/index.htm**.** August 13, 1998. Accessed: Nov 19, 1998.

[5] Andrew S. Tanenbaum, Distributed Operating Systems, New Jersey, USA. Prentice Hall, Inc. 1995.

[6] Roger Sessions, CORBA vs. EJB vs. MDCA; THE BATTLE FOR THE MIDDLE TIER, Object Watch issue 12, May 15, 1998, http://www.objectwatch.com/issue12.htm. Accessed: Oct 19, 1998.

[7] – Keith Brown, Security Briefs, Microsoft Systems Journal, November 1998, http://www.microsoft.com/msj/1198/security/security1198.htm. Accessed: Mar 23, 1999.

[8] David Chappell, The joy of reference counting, Chappell and Associates. http://www.chappellassoc.com/art3.htm. August 1997. Accessed: Oct 26, 1998.

[9] Don Box, Essential COM, second printing, Massachusetts, USA. Addison Wesley Longman, Inc. February 1998.

[10] Roger Sessions, COM and DCOM: Microsoft's vision for distributed objects, John Wiley & Sons, Inc. 1998.

[11] Dr Richard Grimes, Professional DCOM Programming, Canada, Wrox Press Ltd. 1997.

[12] Don Box et al., Seven Tips for Building Better COM-Based Applications. Microsoft Systems Journal, October 1998.

[13] Jerry Anderson, ActiveX Programming with Visual C++ 5.0, Macmillan Computer Publishing. April 1997, http://www.mcp.com/848361600/0-7897/0-7897-1030-7/index.htm. Accessed: Nov 16, 1998.

[14] Deborah Melewski and Jack Vaughan, COM momentum builds, Applications Development Trends. http://www.adtmag.com/pub/jun98/sr603.htm. June 1998. Accessed: Oct 16, 1998.

[15] – Microsoft Corporation, A Guide to Reviewing and Evaluating Microsoft Transaction Server, http://premium.microsoft.com/msdn/library/backgrnd/html/msdn_mtsrevgd.htm, 1998. Accessed Jan 28, 1999.

[16] – Vlad Kroutik, Building distributed applications with DCOM and MTS, Application Development Trends, Nov 98 issue, http://www.adtmag.com/pub/Nov98/fe1101b.htm. Accessed Mar 23, 1999.

[17] Roger Jennings et al., Database Workshop Microsoft Transaction Server 2.0, USA, Sams Publishing. 1997.

[18] Max Dolgicer, MTS: Fast train coming?, Application Development Trends. www.adtmag.com/pub/aug98/fe801.htm. Aug 1998. Accessed: Oct 16 1998.

[19] Don Box, Q&A ACTIVEX/COM, Microsoft Systems Journal, March 1998. http://www.microsof.com/msj/0398/activex0398.htm. March 1998. Accessed: Oct 22 1998.

[20] David Chappell, How Microsoft Transaction Server Changes the COM Programming Model, Microsoft Systems Journal, January issue. http://www.microsof.com/msj/0198/mtscom/mtscom.htm. Jan 1998. Accessed: Oct 26 1998.

[21] Don Box, Q&A ACTIVEX/COM, Microsoft Systems Journal, November 1998. http://www.microsof.com/msj/1198/com/com1198.htm. Jan 1998. Accessed: Oct 22 1998.

[22] David Chappell, Object, Components, and Transactions. Object Watch, Nov 10, 1998. http://www.chappellassoc.com/art4.htm. April 1998. Accessed: Oct 26, 1998.

[23] Maros Cunderlik. Take Advantage of MTS in Your Distributed System with Custom Resource Dispensers. Microsoft Systems Journal, September 1998. http://www.microsoft.com/msj/0998/resource/resource.htm. September 1998. Accessed: Oct 1, 1998.

[24] Mark Spenik and Orryn Sledge, Microsoft SQL Server 6.5 DBA Survival Guide, Second Edition. Macmillan Computer. http://www.itknowledge.com/reference/dir.databases.relationaldatabases.html. Accessed: Nov 19, 1998.

[25] David Chappell, Microsoft Message Queue Is a Fast, Efficient Choice for Your Distributed Application, Micrsoft Systems Journal, July 1998. http://www.microsoft.com/msj/0798/messagequeue.htm. Accessed: Oct 22 1998.

[26] Mark Bukovec and Dick Dievendorff, Use MSMQ and MTS to Simplify the Building of Transactional Applications, Microsoft Systems Journal, July 1998. http://www.microsoft.com/msj/0798/mtsmsmq/mtsmsmq.htm. Accessed: Oct 22, 1998.

[27] Microsoft Message Queuing Services. 1997 Microsoft Corporation. Downloadable from: http://www.microsoft.com/ntserver/appservice/techdetails/overview/msmqrevguide.asp. Accessed: 8 Dec 1998.

[28] – Bristol Technology, Wind/U, http://www.bristol.com/. Accessed Mar 26, 1999.

[29] – Software AG, EntireX, http://www.softwareag.com/corporat/solutions/entirex/entirex.htm. Accessed Mar 26, 1999.

[30] – Michael Nelson, Using Distributed COM with Firewalls, June 1998, http://www.iapetus.com/dcom/dcomfw.htm. Accessed Mar 30, 1999.

[31] – Steve Garone, Managing Component-Based Development: SELECT Software Tools – An IDC White Paper, http://www.selectst.com/downloads/IDC/IDC.asp. Accessed Dec 14, 1998.

[32] – Microsoft, A Guide to Reviewing and Evaluating Microsoft Transaction Server, http://premium.microsoft.com/msdn/library/backgrnd/html/msdn_mtsrevgd.htm. Accessed Jan 28, 1999.

[33] Tom Armstrong, COM + MTS = COM+, Visual C++ developers journal February 1999 issue, http://www.vcdj.com/magazin/febmag99/commts1.asp. Accessed: Mar 24, 1999.

[34] Mary Jo Foley, Another Microsoft DNA Component Falls Into Place, ZDNet. http://www.zdnet.com/sr/stories/news/0,4538,2148766,00.html. October 14, 1998. Accessed: 19 Nov 1998.

[35] Michael Moeller, Microsoft filling out Forms+, PC Week Online and Mary Jo Foley, Sm@rt Reseller, PC Week. http://www.zdnet.com/sr/stories/news/0,4538,360245,00.html. October 9, 1998. Accessed: 19 Nov 1998.

[36] Bob Trott and Katherine Bull, Microsoft looks to single UI to access disparate data, InfoWorld Electric. http://www.infoworld.com/cgi-bin/displayStory.pl?980911.whmsui.htm. Sep 11, 1998. Accessed: 19 Nov 1998.

[37] Mary Jo Foley, Microsoft pushes the DNA envelope, Sm@art Reseller, PC Week. http://www.zdnet.com/pcweek/news/0803/04adna.html. August 4, 1998. Accessed: 19 Nov 1998.

[38] Martin Veitch, Storage plans in missing piece of DNA jigsaw, IT Week. http://www.zdnet.co.uk/news/1998/41/ns-5776.html. 17 Oct 1998. Accessed: 19 Nov 1998.

[39] Stuart J. Johnston, Microsoft Aims To Simplify Distributed App Development. http://www.informationweek.com/newsflash/nf699/0910_st3.htm. September 10, 1998. Accessed: 19 Nov 1998.

[40] Microsoft Corporation, DHTML, HTML & CSS, 1999. http://msdn.microsoft.com/workshop/c-frame.htm#/workshop/author/default.asp. Accessed: 7 Apr 1998.

[41] David Chappell, Interceptors Add Object Functionality, BYTE Magazine, December 1997, http://www.byte.com/art/9712/sec6/art6.htm. December 1997. Accessed: Nov 18, 1998.

[42] David Chappell. Articles ”COM+” and “COM+ Redux”. ActiveX/COM Columns from Component Strategies Magazine, http://www.chappellassoc.com/article.htm. Apr 1998, July 1998, respectively. Accessed: 26 Oct 98.

[43] Mary Kirtland, Object-Oriented Software Development Made Simple with COM+ Runtime Services, Microsoft Systems Journal. November 97 issue. p. 49-59.

[44] Mary Kirtland, The COM+ Programming Model Makes it Easy to Write Components in Any Language, Microsoft Systems Journal, Dec 97 issue, http://www.microsoft.com/msj/1297/complus2/complus2top.htm. Accessed: 26 Oct 98.

[45] Sally Cusack, Here today, COM tomorrow, Application Development Trends. Jun 98 issue. http://www.adtmag.com/pub/jun98/devtrends.htm, June 1998. Accessed: 16 Oct 98.

[46] Michael Moeller, COM+ services add up, PC Week Online, 13 Apr 1998. http://www.zdnet.com/pcweek/news/0413/13com.html. Accessed: 1 Oct 98.

[47] Imran Sayeed, ”A Corba, DCOM primer”. ADT Magazine June 1998, http://www.adt.com/pub/jun98/fe601.htm. June 1998. Accessed: Nov 16, 1998.

[48] - http://www.sun.com/swdevelopment/whitepapers/CORBA-vs-DCOM.pdf;$sessionid$2WKYN3AAAAH4HAMUVFZE4GUBS1JHEUDO

[49] Yun Wang and Tom Young, Distributed object infrastructures, InfoWorld, November, 1997. http://www.infoworld.com/cgi-bin/displayTC.pl?/971020analysis.htm. Oct 20, 1997. Accessed: Oct 16, 1998.

[50] Various authors, Java Brews Up a Storm in the Enterprise, Network Computing Online, 1998, http://www.nwc.com/919/919f1.html. Accessed: Oct 14 1998.