An agent tool for integrating component-based applications

Ebtehal Alsaggaf [1] and Prof.Dr. Fathy albouraey [2]
Faculty of Computing and Information Technology, King Abdulaziz University KAU, Jeddah, Saudi Arabia
[1]Ebte-alawi@hotmail.com , [2]fathy55@yahoo.com

ABSTRACT: *The concept of building the best component-based application becomes very attractive. But, the reusable components must comply with specific underlying middleware architecture in order to interact with each other efficiently. This dependency on the underlying architecture creates compatibility problems. Anyway, software agents provide a powerful new method for implementing the information systems. Multi-Agent System (MAS) can communicate and cooperate with each other to solve complex problems and implement complex systems. So, in this research, we will define how component CORBA can interact with DCOM by using MAS. This will able to make us freedom to choose the best attributes from both systems and combine them to build the best possible application that specifically suits my environment.*

Keywords: *CORBA, DCOM , COM, mobile agent, MAS.*

1. INTRODUCTION

The software applications become more complex and computer hardware becomes more powerful and more affordable distributed computing in future. Component-based software is emerged as an important developmental strategy focusing on achieving systems development. It can be defined as a unit of software that implements some known functions and hides the implementation of these functions behind the interfaces that it exposes to its environment [1]. Component technology offers many advantages for scientific computing since it allows reusability, interoperability, maintability, adaptability, distribution that can used easily, efficiently in application development. It speeds the development of applications, operating systems or other components. It enables the developers to write distributed applications in the same way of writing non distributed applications. As a result, scientists can focus their attention to overall application design and integration [2].

Many reusable components are available on the Internet. But, they must comply with specific underlying middleware architecture in order to interact with each other efficiently. This dependency on the underlying architecture creates compatibility problems between components based on different architectures. There are three most widely-used component standards, which are Component Object Model (COM/DCOM) [1], Common Object Request Broker Architecture (CORBA) [3, 4] and Java/Remote Method Invocation (Java/RMI) [3].

The (DCOM) and (CORBA) are two models that enable software components with different descent to work together. The aims of component-based software development are to achieve multiple quality objectives, including interoperability, reusability, implementation transparency and extensibility [4].

Anyway, Agents are applied as interaction entities to mediate differences between components [8]. The term agent means a variety of things to a variety of people, commonly it is defined as independent software program, which runs on behalf of a network user. It can run when the user is disconnected from the network [4]. However, mobile agent-based computing, being high-level and flexible, can be a useful tool in rapid prototyping due to its high level of abstraction and ease of use. MAS researchers develop communications languages, interaction protocols, and agent architectures that facilitate the development of multi-agent systems. In this study, multi-agent system is proposed to solve the compatibility problem by designed integration tool between CORBA and DCOM technologies.

1.1 CORBA and DCOM comparison

Both DCOM and CORBA frameworks provide client-server type of communications. To request a service, a client invokes a method implemented by a remote object, which acts as the server in the client-server model. The service provided by the server is encapsulated as an object and the interface of an object is described in an Interface Definition Language (IDL). The interfaces defined in an IDL file serve as a contract between a server and its clients. Clients interact with a server by invoking methods described in the IDL. The actual object implementation is hidden from the client. CORBA also supports multiple inheritances at the IDL level, but DCOM does not. Instead, the notion of an object having multiple interfaces is used to achieve a similar purpose in DCOM. CORBA IDL can also specify exceptions [9].

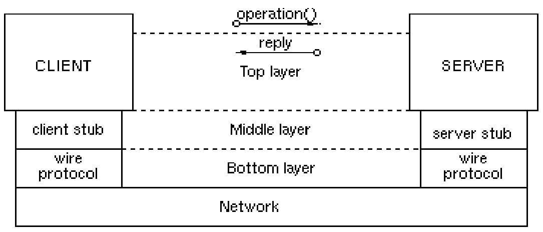


Figure 1: RPC structure

	DCOM	CORBA
Top layer: Basic programming architecture		
Common base class	`IUnknown`	`CORBA::Object`
Object class identifier	`CLSID`	interface name
Interface identifier	`IID`	interface name
Client-side object activation	`CoCreateInstance()`	a method call/`bind()`
Object handle	interface pointer	object reference
Middle layer: Remoting architecture		
Name to implementation mapping	Registry	Implementation Repository
Type information for methods	Type library	Interface Repository
Locate implementation	SCM	ORB
Activate implementation	SCM	OA
Client-side stub	proxy	stub/proxy
Server-side stub	stub	skeleton
Bottom layer: Wire protocol architecture		
Server endpoint resolver	OXID resolver	ORB
Server endpoint	object exporter	OA
Object reference	OBJREF	IOR (or object reference)
Object reference generation	object exporter	OA
Marshaling data format	NDR	CDR
Interface instance identifier	IPID	object_key

Table 1. Table of the Summary of corresponding terms and entities

In both DCOM and CORBA, the interactions between a client process and an object server are implemented as object-oriented RPC-style communications. Figure 1 shows a typical RPC structure. In DCOM, the client stub is referred to as the proxy and the server stub is referred to as the stub. In contrast, the client stub in CORBA is called the stub and the server stub is called the skeleton. Sometimes, the term "proxy" is also used to refer to a running instance of the stub in CORBA.

Their main differences are summarized in Table 1. First, DCOM supports objects with multiple interfaces and provides a standard QueryInterface() method to navigate among the interfaces. This also introduces the notion of an object proxy/stub dynamically loading multiple interface proxies/stubs in the remoting layer. Such concepts do not exist in CORBA. Second, every CORBA interface inherits from CORBA::Object, the constructor of which implicitly performs such common tasks as object registration, object reference generation, skeleton instantiation, etc. In DCOM, such tasks are either explicitly performed by the server programs or handled dynamically by DCOM run-time system. Third, DCOM's wire protocol is strongly tied to RPC, but CORBA's is not. Finally, we would like to point out that DCOM specification contains many details that are considered as implementation issues and not specified by CORBA. As a result, they used the Orbix implementation in many places in order to complete the side-by-side descriptions [5].

1.2 Software agents

Software agents, one of the most exciting new developments in computer software technology, can be used for quickly and easily building integrated enterprise systems. The idea of having a software agent that can perform complex tasks on our behalf is intuitively appealing [9]. The natural next step is to use MAS that communicate and cooperate with each other to solve complex problems and implement complex systems. Software agents provide a powerful new method for implementing these information systems.

Mobile agent-based computing is an attractive, though not widely accepted model for structuring distributed solutions. The most distinctive feature of this model is the mobile agent: a migrating entity, with the capability to transfer its current state and code to a different network location. Compared to remote communication, migration could reduce network traffic. Furthermore, mobile agents can function independently of their dispatching host and contact it later only to return a small set of results. Relevant application domains for mobile agents are distributed information retrieval, monitoring and filtering [7].

There are several reasons for the quite limited acceptance of the mobile agent technology. First, it's quite difficult to identify a distributed problem whose solution can be based on mobile agents only, instead of an equivalent or even better "classical" message-passing or Web Services solution. Another major concern is security: how to protect agents and servers from one another. Nevertheless, mobile agent-based computing, being high-level and flexible, can be a useful tool in rapid prototyping. Due to its high level of abstraction and ease of use, it can also be applied as a teaching tool in introducing students to distributed computing [8].

However, applications require multiple Agents that can work together. A MAS is a loosely coupled network of software agents that interact to solve problems [6, 7]. The difficulty arises from the need to understand how to combine elements of various content languages and interaction protocols in order to construct meaningful and appropriate messages [10] but, it has the following advantages [7]:

1. A MAS distributes computational resources and capabilities across a network of interconnected Agents. A MAS is decentralized and thus does not suffer from the "single point of failure" problem in centralized systems.
2. A MAS allows for the interconnection and interoperation of multiple existing systems.
3. A MAS efficiently retrieves, filters, and globally coordinates information from sources that are spatially distributed.
4. In MAS, computation is asynchronous.
5. A MAS enhances overall system performance, efficiency, reliability, extensibility, robustness, maintainability, responsiveness, flexibility, and reuse.

2 RELATED WORK

2.1 COM-CORBA Interoperability

This book is the first complete guide to do the architects of COM-CORBA bridge to make distributed objects work in a heterogeneous environment, developers must bridge the gap between Microsoft COM/DCOM and the industry CORBA standard. It starts with easy-to-understand descriptions of both COM and CORBA, exploding the myth of complexity that surrounds these technologies. Next, it delivers a step-by-step guide to building your own working, scalable and transparent COM/CORBA systems, integrating Windows and UNIX. It has CD-ROM which includes MS-Access source code for all examples, plus trial versions of IONAs Orbix COMet, the first commercial bridge for linking COM and CORBA modules, and OrbixWEB 3.0 tools for building Internet-based CORBA Server applications [1].

2.2 Multi-Technology Distributed Objects and their Integration

In this article discussing the basic incompatibility points, and overviewing the basic strategies for bridging the gap between CORBA, DCOM, and RMI. Most of the work in the area, they surveyed concerns bridging CORBA and DCOM. This is expected considering the widespread deployment of Microsoft's operating systems and the acceptance of CORBA as the most mature middleware architecture. Moreover, the early presence of a variety of COM components and ORB products from commercial companies led developers to use those products. As a result the bridging between CORBA and DCOM was an urgent need.

They can distinguish two basic approaches for bridging, the static bridging, and the dynamic bridging. Under static bridging, the creation of an intermediate code to make the calls between the different systems is required. The disadvantage of the static bridge is that any changes on the interfaces require a change in the bridge. In dynamic bridging there is no code depended on the types of calls,the implementation belongs to commercial companies which have released many bridge tools, compliant with OMG's specification. Some of these products are PeerLogic's COM2CORBA, IONA's OrbixCOMet Desktop, and Visual Edge's ObjectBridge. All the above products realize one of the interface mappings that OMG specifies [2].

Many attempts have been undertaken to bridge the gap between the underlying object architectures,until now the use of a single middleware product is the most reliable solution.

2.3 A Component-Based Architecture for Multi-Agent Systems

This study introduced a formal multilayered component-based architecture towards developing dependable MAS. They had not investigated any specific approach for verifying the MAS design yet.

In a large system, some problem solving required agents that have the BDI set, the MAS is not only heterogeneous but also has a heavy overhead on the system execution. This complexity must be resolved at the architecture level so that in an implementation the complexity does not arise. In [6], they introduced a formal multilayered component-based architecture towards developing dependable MAS. They had not investigated any specific approach for verifying the MAS design yet. However, it seems feasible that they could provide a uniform platform for both programming and verifying MASs, if they provided reasoning rules for Lucx.

After we studied the above researches, it can be feasible for us to design MAS as integration tool for component-based application by investigating specific widely component standards, which are DCOM and CORBA.

3 THE CURRENT WORK

There are several phases to develop the best component-based application, the first phase is analysis phase which is concerned on user requirements and then present system model which is corresponds to use cases in object oriented design. The second phase is design which specifies the different roles to be developed in the software system, and their interactions. It is composed of the role model and the interaction model. The third phase is implementation. The last phase is testing phase which defines the types of used tests. These phases called Software Development Cycle.

3.1 System Analysis

Implementing distributed computing presents many challenges with respect to middleware in general and CORBA and DCOM specifically. Depending on both the business and technical problems that need to be solved, the greatest probability is make Integration between CORBA & DCOM to build the best component-based application from hybrids of the best technology and tools available at the time and the concept of building application which contains components from both CORBA & DCOM, is giving you the freedom to choose the best attributes from both technologies [1,2].

How do we make this Integration?

For many organizations, a business and technology need exists for “Integrating” between CORBA and DCOM. This generally means providing a bridge between CORBA and COM, and two mappings.

In order to transparently couple components from DCOM and CORBA, some of bridging software is needed to handle the translation of types and object references between the two systems.

What actually required building abi-directional Bridge between CORBA & DCOM. ?

We should to be able to do the following [1]:

• Transparently contact object in one system to other.

• Use data types from one system as though they were native type in the other system.

• Maintain identity & integrity of the types as they pass through the bridge in order to reconstitute them later.

We can distinguish two basic approaches for bridging, the static bridging, and the dynamic bridging .

Static bridging: This provides statically generated marshalling code to make the actual call between the object systems. Separate code is needed for each interface that is to be exposed to other object system. Static bridging also implies that is an interface-specific package (DLLs, configuration files, ect.) which needs to be deployed with client application.

Dynamic bridging: This provides a single point of access between the two systems which all calls go through. No marshalling code is required to expose each new interface to the other object system.

In either case, a proxy is needed on the client side to intercept and pass on the call to remote machine, with a stub on the server side to receive it. (Of course, if the call is being made in-process, it will occur directly between the calling object and the target object, with no proxy or stub required) Hence, all that is required to provide a bridge between CORBA and DCOM is to provide some thing (bridge call) which belongs to the current object system and sent the bridge call to MAS.

Under static bridging in the two technologies, the creation of an intermediate code to make the calls between the different systems is required. That intermediate code would be called the bridge object which could be sent and receive the call function to and from MAS. In dynamic bridging, the operation is based on the existence of a dynamic mechanism which can manage any call in spite of the interfaces [1, 2].

To make the Integration, we will first consider the bridge between the client and server and the bridge between two components for both technologies which we found them the same bridge thus needing to build two mapping tables: function table [Tablt3] and data type table [Tablt2].

DCOM	CORBA
short	short
unsigned short	short
long	int
Unsigned long	int
double	double
float	float
char	char
Boolean	Boolean
byte	byte

TABLE 2: DATA TYPE TABLE

DCOM	CORBA
IUnknown	`CORBA::Object`
QueryInterface	-
Addref	-
Release	-
CoCreateInnstance	a method call/`bind()`
UUID	-
CLSID	interface name
get()	get()
set()	set()

TABLE 3: FUNCTION TABLE

The same bridge use in both ways. In our model, the integrator (bridge) is a MAS, which can contain a set of software agents that may run on one computer, and may be distributed on different computers in the network. The system contains different agents having different functions and tasks.

Based of the features of Multi-Agent system MAS, it is natural to introduce MAS in our system where it can be applied to bridging and mapping the two most widespread technologies.

When the component need to call another component in the same techniques nothing to do else if the component need to call another component in different techniques then send a message to MAS.

3.2 System Architecture

The objectives of design this system model is to make Integration between CORBA and DCOM technologies in the way in which DCOM and CORBA are differ and resemble, organizing the comparison according to some studied criteria .This system consists of Combine Agent, Mapper Agent, Manager Agent, Agent library, DCOM component and CORBA component. The sequence of the work among Agents as the following:

1. The Manager Agent receives the massage of the call function which sent by CORBA component (for example), the main job is achieved by the Manger Agent. The functions of Manger Agent are:

- Receive this call and determine the kind of technology.
- Send this call to Mapper Agent.
- Manages all active Agents.
- Last, sent the mapping call to other technology.

2. The Mapper Agent. Separate the formula and sent all sub formula to corresponding agent, as the following:

- The Interface Agent takes and reads its Interface, comparing CORBA Interface to its corresponding DCOM from library Agent and then written the equivalent one of new mapping Interface
- The function Agent takes and reads its function, comparing CORBA function to its corresponding DCOM from library Agent and then written the equivalent one of new mapping function.
- The data type Agent takes and reads its data, comparing CORBA data type to its corresponding DCOM from library Agent and then written the equivalent one of new mapping data type.

3. Agent library has three tables: function table [Tablt 3] and data type table [Tablt 2] and Table of the corresponding terms and entities [Tablt 1]. These tables are be fixed and stored in the database. The function table has two columns, one for DCOM functions and the other column is for the corresponding function in CORBA. The same technique will be applied for data type table in which each data type of DCOM arranged to its corresponding data type in CORBA , Table of the corresponding terms and entities summarizes the corresponding terms and entities in the two architectures.

4. Agent library will come then after receiving a message from Interface Agent, function Agent or The data type Agent .For example, if Interface Agent will send message to Agent library that will execute a query to the database using JDBC (Java Data Base Connectivity). The result of query will be represented as object. This will result in an array of objects (may be one object).

5. The Combine Agent takes the result of mapping from data type Agent, function Agent and Interface Agent. Then combine them to building anew DCOM formal and sent this formal to the Manger Agent which sending them to the DCOM

6. The DCOM component receives the call formula to complete the operation, and then produce the result of function which will return back to MAS.

7. The MAS will be applied the same technique for a DCOM result to produce a new CORBA formal after mapping in it (which may be the replying of call function).

At the end, The Integration between CORBA & DCOM technologies will be completed (see Figures 2, 3).

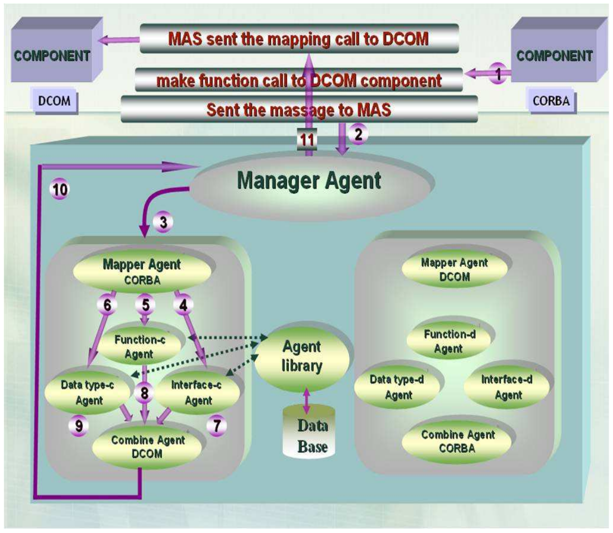


Figure 2: CORBA to DCOM steps in MAS

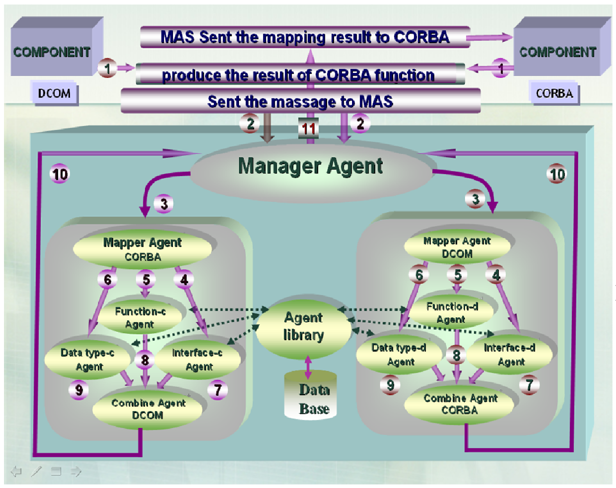


Figure 3: DCOM to CORBA steps in MAS

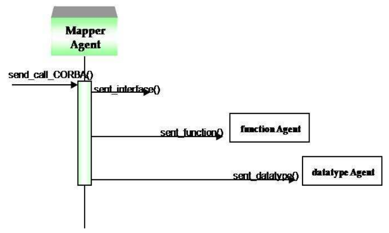


Figure 6: Interaction Diagram for the Mapper Agent

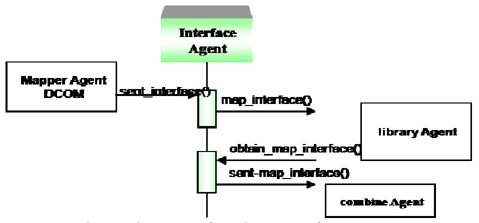


Figure 7:.Interaction Diagram for the Interface Agent

3.3 System Detailed Design

Detailed design phase consists of the following steps:

- Design Interaction diagram for the whole system and for each agent which presents the agents and messages between them.
- Design Block interfaces by written all functions and their parameters for each agent
- Design database schema.

3.3.1. Interaction Diagram

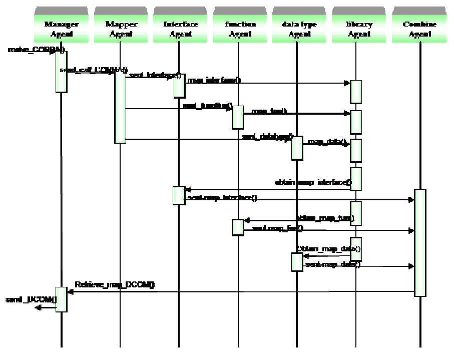


Figure 4: Interaction Diagram for the whole system

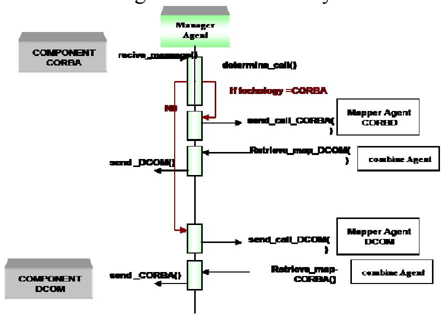


Figure 5: Interaction Diagram for the Manager Agent

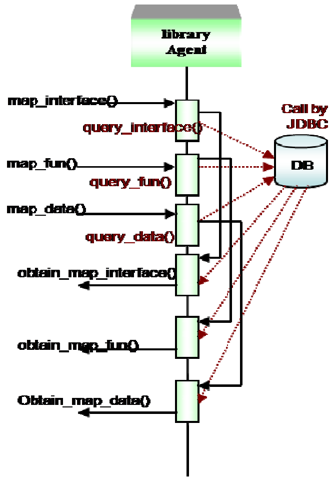


Figure 9: Interaction Diagram for the library Agent

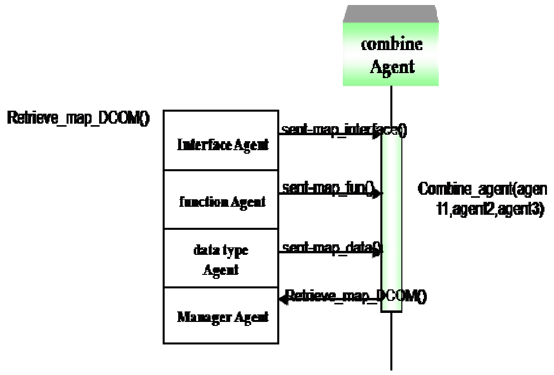


Figure 9: Diagram for the combine Agent

3.3.2. Block interfaces

In this section we built block interfaces by written all functions and their parameters for each agent as following:

Manager Agent Functions
recive_massege()
determine_call()
send_call_CORBA()
send_call_DCOM()
Retrieve_map-CORBA()
Retrieve_map_DCOM()
send _CORBA()
send _DCOM()

Mapper Agent Functions
sent_interface()
sent_function()
sent_datatype()

library Agent Functions
query_interface()
query _fun()
query _data()
obtain_map_interface()
obtain_map_fun()
obtain_map_data()

Interface Agent Functions
map_interface()
sent-map_interface()

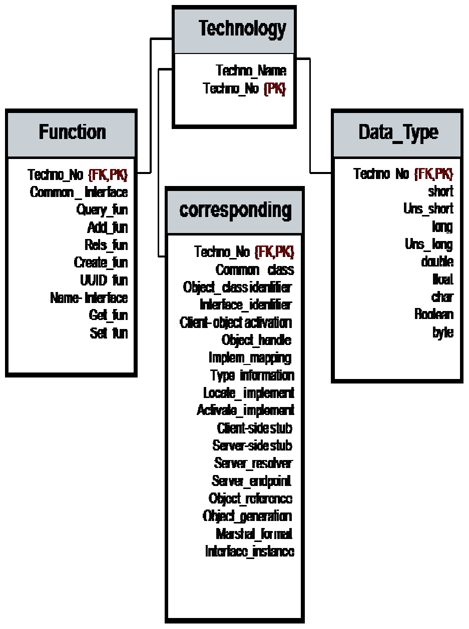
function Agent Functions
map_fun()
sent-map_fun()

data type Agent Functions
map_data()
sent-map_data()

Combine Agent Functions
Combine_agent(agent1,agent2,agent3)
Send_result()

3.3.3. Database schema

This structure represents the data base tables in the model:



4 DISCUSSION

This section will briefly discuss some of the advantages of our system model. Some of these advantages are applicable to all distributed computing technologies, including CORBA and DCOM. In additional, it will also add the advantages of MAS that we have mentioned in section 1.4, thus our system model can able to make how to combine elements of various Interfaces , data types and functions in order to construct meaningful and appropriate messages .So it will have the following advantages:

- The components will able to interact with each other for two components, hosted on different component architectures. In 3.3 section, we describe the details of design interaction diagram for the whole system and for each agent.
- Speeding up development processes: Since applications can be built from existing pre-built components, this helps to maintain the speed up of development process tremendously.
- Improving deployment flexibility: Organizations can easily customize an application for different areas by simply changing certain components in the overall application.
- Lowering maintenance costs: Certain functions of an application can be grouped into discreet components, which can be upgraded without retrofitting the whole application.
- Improving scalability: Since applications are built from many objects, the objects can be redeployed to different machines when needs arise or even

multiple copies of the same components can run simultaneously on different machines.

Moreover, it will achieve multiple quality objectives for developing it, including:

1. Interoperability and reusability by using block interfaces which have all functions and their parameters for each agent in the MAS model
2. Implementation transparency and extensibility were done by using all functions and their parameters for each agent and also by representing the database schema which include three tables that stored in it (see 3.3 sections).

Finally, we will build a software integration tool for component-based application by using MAS.

5 CONCLUSION AND FUTURE WORK

As a result the bridging between CORBA and DCOM was an urgent need. For software component to integrate with each other, it is difficult if not impossible for two objects conforming to dissimilar technologies to interact with each other. The above model is the way which uses the software agents to build a software integration tool between CORBA and DCOM technologies which will give us the freedom to choose the best attributes from both systems and combine them to build the best possible component-based application.

This model combine the advantages of MAS , CORBA and DCOM technologies ,thus it able to transparently contact object in one system to other rely on the features of MAS and it also use data types from one system as though they were native type in the other system by Agent library with DB. This will maintain identity and integrity of the types as they pass through the bridge in order to reconstitute them later. This will achieve multiple quality objectives, including interoperability, reusability, implementation transparency and extensibility

In the future work, we want to improve our model to support integration tool for all Component technologies to build a software component-based application by using MAS. MAS will be modifying by adding new two mapping for each new technology which it adding to it. Finally, the system that we have designed stills a basic model. So, we will interest to complete the system developing and implementing it.

6 References

[1] R Geraghty, S Joyce, T Moriarty and G. Noone, *"Com-Corba interoperability"*, Prentice Hall PTR Upper Saddle River, NJ, USA, 1999.

[2] Raptis, K., D. Spinellis, and S. Katsikas, *"Multi-technology distributed objects and their integration. Computer Standards & Interfaces"*, vol. 23 no. 3, pp. 157-168, 2001.

[3] *"The CORBA Programming Model"*, 2008, http://download.oracle.com/docs/cd/E15261_01/tuxedo/docs11gr1/tech_articles/CORBA.html

[4] IBM, *"Is web services the reincarnation of CORBA?"* ,2001, http://www.ibm.com/developerworks/webservices/library/ws-arc3/

[5] Pritchard J.,*" COM and CORBA side by side: architectures, strategies, and implementations"*, Addison-Wesley Professional ,1999.

[6] Wan, K.Y. and V. Alagar. *"A Component-Based Architecture for Multi-Agent Systems"*, IEEE Computer Society ,2006.

[7] Agent Technology ,Green Paper ,*"Agent Working Group "*,OMG Document ec/2000-03-01,Version 0.91, 2000.

[8] Dr. R.A. Adey, A.K. Noor, B.H.V. Topping, *"Advances in Engineering Software"*, vol. 30 no. 8, 2010.

[9] Manvi, S.S. and P. Venkataram, *"Applications of agent technology in communications: a review. Computer communications"*, vol. 27 no. 15, pp. 1493-1508 ,2004.

[10] Shepherdson, J.W., H. Lee, and P. Mihailescu, *"mPower—a component-based development framework for multi-agent systems to support business processes"*, BT Technology Journal, vol. 25 no.3, pp. 260-271, 2007.