A NEW DISTRIBUTED COMPOSITE OBJECT MODEL FOR COLLABORATIVE COMPUTING

Güray YILMAZ [1] and Nadia ERDOĞAN [2]
[1] *Dept. of Computer Engineering, Air Force Academy, 34807 Yeşilyurt, İstanbul, Turkey*
[2] *Dept. of Computer Engineering, İstanbul Tech. University, 80626 Maslak, İstanbul, Turkey*
g.yilmaz@hho.edu.tr, erdogan@cs.itu.edu.tr

Abstract: Distributed systems provide sharing of resources and information over a computer network. A key design issue that makes these systems attractive is that all aspects related to distribution are transparent to users. But, current wide area distributed systems fail to hide implementation aspects related to distribution of the information. Solutions developed for local systems do not scale to wide area case. In this paper we have developed a new object model, called Distributed Composite Object (DCO) model. We have also designed and implemented a software layer, DCOBE that can be placed on top of Java programming language to provide a uniform interface for collaborative application developers to use. This layer provides basic mechanisms for facilities needed by many applications, such as communication, data replication and partitioning, consistency and dynamic deployment of application.

Keywords: Wide area distributed systems, collaborative computing, distributed composite objects, middleware solutions.

1. Introduction

Distributed systems provide sharing of resources and information over a computer network. A key design issue that makes these systems attractive is that all aspects related to the distribution are transparent to users. Unfortunately, general-purpose wide area distributed systems that allow users to share and manage arbitrary resources in a transparent way hardly exist. Constructing wide area applications, such as sharing data across the Internet, often requires a substantial development effort. This is mainly caused by the lack of proper communication facilities as offered by the underlying operating systems and middleware solutions (Steen, Homburg, and Tanenbaum 1999).

All well-known distributed systems today adopt the remote-object model. In this model, an object is located at a single location only, whereas the client is offered access transparency through a proxy interface. At best, the object is allowed to move to other locations without having to explicitly inform the client. There are a number of serious drawbacks to the remote-object model, most notably its lack of scalability. The remote-object model itself provides no mechanisms that support a developer in designing and implementing different invocation schemes, which is necessary if we are to apply scaling techniques such as caching, replication and distribution (Neuman 1994).

As an alternative to the remote-object model, we have designed and implemented a new object model, called *Distributed Composite Object* (DCO) model. In our opinion, internet-wide distributed-collaborative applications generally manipulate large or coarse-grained objects. But clients of this type of applications are usually interested in only a small part of the object. So, if we partition a large-grained object into the finer-grained *subobjects* (SO), an

application's performance would be enhanced by the reduction in the amount of data accessed and the amount of data transferred unnecessarily between distributed sites.

The fundamental idea behind the design of the DCO is that it is *physically distributed* over multiple sites. Most current middleware systems, such as DCOM (Eddon and Eddon 1998), DCE (Rosenberry, Kennedy and Fisher 1992) and CORBA (OMG 1999), view a distributed object as an object running on a single machine, possibly with copies on other machines. This object is presented to remote clients as a local object by means of proxies. Our view of what a distributed object is gives us flexibility with respect to replication, caching and distribution of the object's state.

The rest of this paper is organized as follows. In section 2, DCOBE architecture and its basics are introduced. DCO object model and its components are explained in section 3. Management of consistency is introduced in section 4. In section 5, object generation is explained. and finally our conclusions are stated in section 6.

2. DCOBE Architecture

In this paper, we describe the overall design and rationale for the ***D****istributed* ***C****omposite* ***O****bject* ***B****ased* ***E****nvironment* (DCOBE) toolkit and its internals, as well as our early experience in implementing a distributed composite object model in Java. The DCOBE toolkit is a collection of Java packages intended to support the construction of customized distributed composite objects. Specifically, we have designed a software layer that can be placed on top of Java virtual machine (Gosling, Joy and Steele 1996) to provide a uniform interface for Internet-wide collaborative application developers to use. This layer provides basic mechanisms for facilities needed by many applications, such as object composition, replication, consistency, and dynamic deployment of application.

Using DCOBE, the programmer can develop his application as if it were to be executed in a centralized environment. Furthermore, the application is configured for a distributed setting without any modification to the application source code. This is achieved by the addition of some pieces of code to the objects automatically, which ensures dynamic binding, synchronization functions and consistency protocols.

The basic functionalities of the DCOBE system are:

- DCOBE is a toolkit that is specifically designed for developing internet-wide distributed object-based applications.
- DCOBE adopts the DCO model. The fundamental idea behind the design of the DCO is that it is *physically distributed* through subobjects (SO). In other words, the object's state consists of the states of the several SOs resides on different nodes at the same time.
- Dynamic deployment. Applications are dynamically deployed to the requesting nodes from the node that hosts the application: thus we do not require applications to be installed on a machine prior to execution.
- Dynamic adaptation. A large-scale wide area application should be dynamically adaptable by adding or removing components. DCOBE allows add, remove, or change a SO of the DCO without affecting clients that are presently bound to that object.

- Replication. The key requirement of distributed collaborative applications is replication. DCOBE allows shared SOs of the DCO to be replicated on cooperating nodes, thus enabling local invocation on distributed objects and thus reducing latency.
- Transparency. In DCOBE, a distributed collaborative application can be developed as if it were to be run centralized. Distribution, synchronization and consistency issues are programmed separately from the application code. This also enables the execution of existing centralized applications in a distributed environment.

2.1. DCOBE Elements

We have designed DCOBE system in order to support the distributed composite object model explained shortly above. A schematic view of the major elements of DCOBE architecture is presented in Figure 1. DCOBE system consists of two basic elements: a DCOBE_server and several DCOBE_clients. DCOBE_server and DCOBE_clients consist of Java packages running on a JVM that facilitates programming on distributed applications and provide distributed composite object based environment to the users.

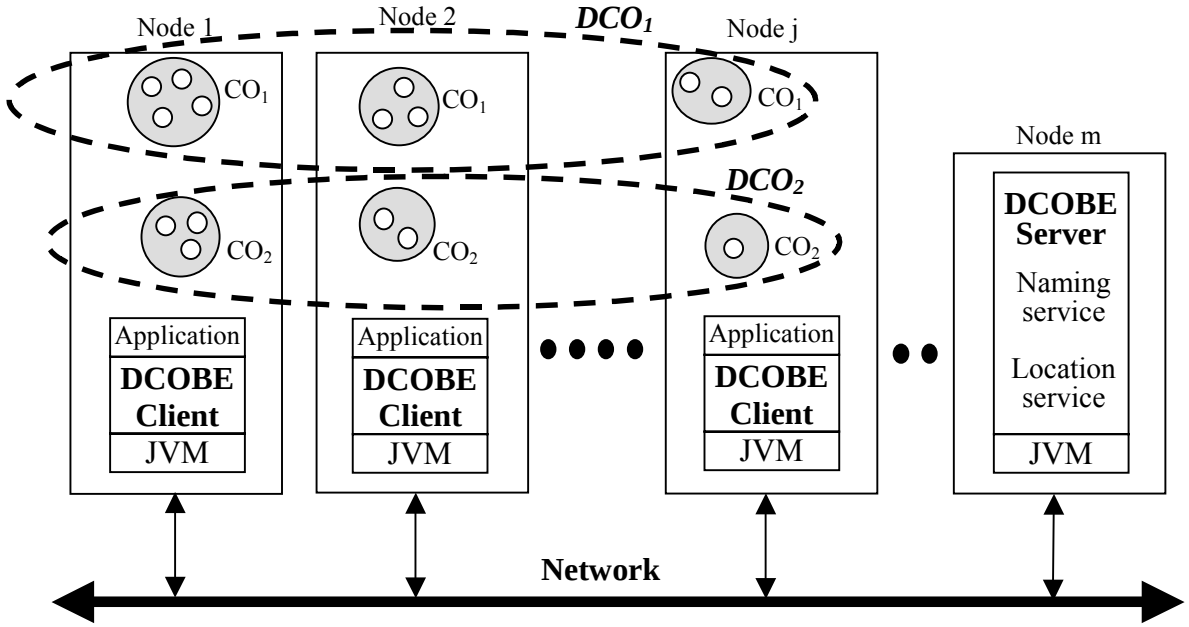

Figure 1. DCOBE architecture.

In order to initialize the DCOBE system, the DCOBE_server is started on a definite node with its location information (host name and port number). The other DCOBE_clients, which participate the DCOBE system later, communicate with the DCOBE_server using this location information.

An application program may start a DCOBE_client on any node. The DCOBE_client contacts the DCOBE_server to register itself. In the second step of the registration process, DCOBE_server sends location information of other participating DCOBE_clients to the new DCOBE_client and also sends location information of the new DCOBE_client to the former DCOBE_clients. So, each DCOBE_client can communicate with other directly. During execution, new DCOBE_clients may be included and former clients may be removed from the system dynamically.

3. Distributed Composite Objects

The DCO model is based on the idea that the implementor of an object can distribute the state of the object among multiple SOs. This implies that the state of the *root object* becomes partitioned. These SOs can be placed on different sites, may be replicated, or may even be composite objects themselves.

In DCO model, several SOs are grouped together and form a *Composite Object* (CO). A CO is created on a single node with its SOs. Then, CO can be replicated over several sites on demand together with its some of the SOs. COs which spread out on several nodes constitute a DCO. In Figure 2, there is a DCO spreads over three sites.

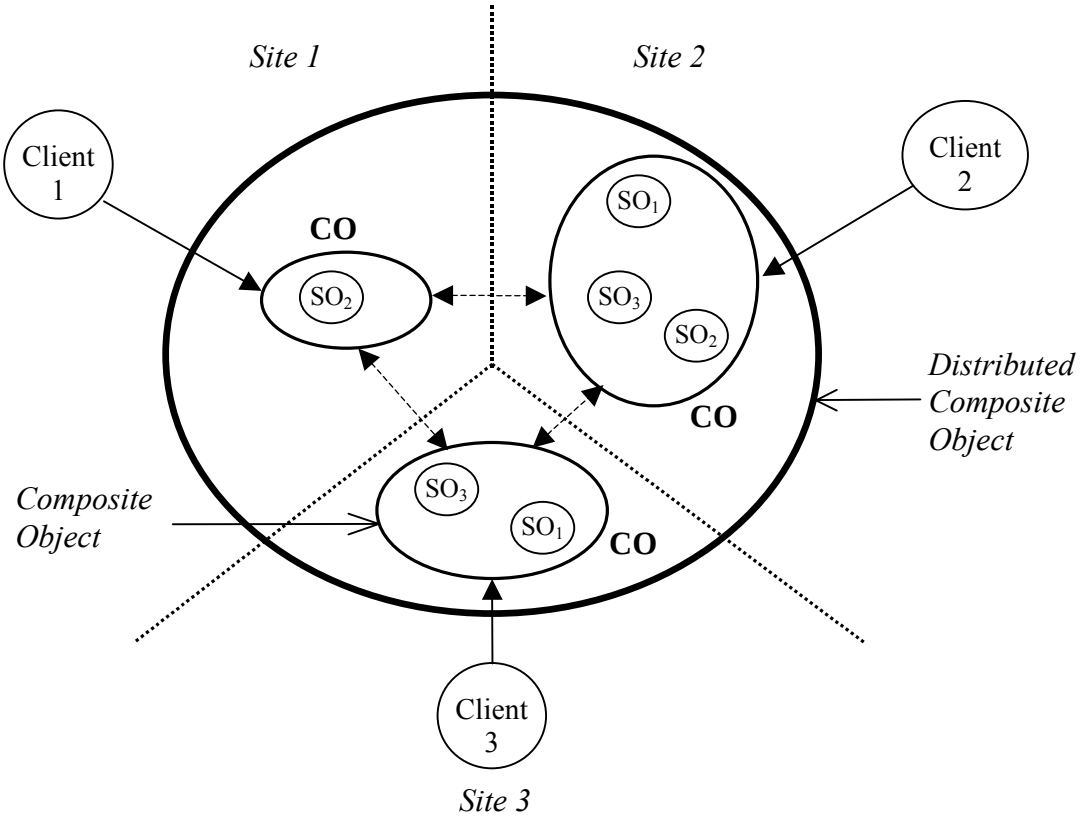

Figure 2. A distributed composite object, which is distributed over three address spaces with its three SOs.

If we examine Figure 1, a CO is constructed on *Site 2* with its three SOs (SO_1, SO_2 and SO_3). Then, it is replicated on *Site 1* with only SO_1, and *Site 3* with SO_1 and SO_3. Only the designer of the CO knows the structure of the composition, while its users don't have any information about the layout of the object except for CO's offered interface. They are also not interested in which SOs have been loaded on their site, and how they cope with dynamic binding, synchronization and consistency issues.

Like the *Fragmented Objects* model (Makpangou, Gourhant, LeNarzul and Shaphiro 1994), a DCO has two aspects: From the clients point of view, it is a single shared object. It is shared by several client objects, which are localized in different address spaces, possibly on different sites. It is accessed via a programmer-defined interface. Its components, and in particular their distribution are not visible. From the designers point of view, a DCO encapsulates a set of

cooperating SOs. Each SO of the DCO is an elementary object. In other words, each SO has a centralized representation.

3.1. Structure of the Subobject

As mentioned above, a CO consists of several SOs. A SO resides in a single address space and communicates with other SOs in order to provide the consistent image of the DCO. As is shown in Figure 3, each SO is formed of two objects: a *fragment object* and *a control object.* A SO implements the interface, which its fragment object offers.

Fragment object, in Figure 3, is a local fragment that implements part of the actual semantics of the DCO. A developer is responsible for constructing a class object for each different kind of fragment object that is part of the DCO. The code of the fragment object is constructed personally by developers, while the code for the control object and the subobject are generated automatically by an object generator using interface specifications of the fragment object.

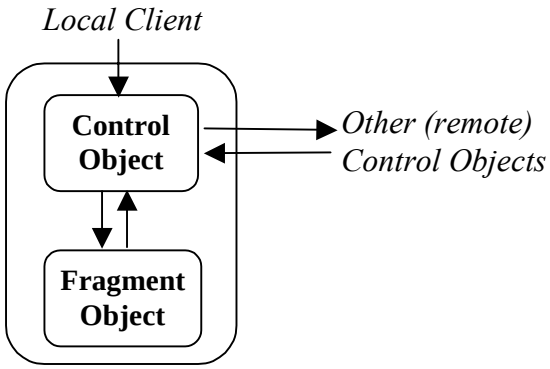

Figure 3. Structure of the subobject.

Control object generation is based on the interface of the corresponding fragment object. It controls two types of invocation requests: those coming from the local client and those coming in through the network from other control objects. The interface the control object offers to the local client is the same as the fragment object's interface. In addition, it offers a secondary interface which consists of certain consistency and concurrency control methods, to other control objects that reside on different address spaces.

Control object is responsible for handling incoming invocation requests. The global state of the distributed object is made up the state of its various fragment objects and these fragment objects are replicated for reasons of availability, fault tolerance or performance. Therefore, the control object is also responsible for providing concurrency and consistency issues.

4. Management of Consistency

On a distributed system, the runtime support system is expected to take care of all consistency and coherency issues if more than one process accesses an object. In DCOBE, COs are replicated on multiple nodes with its several SOs. Therefore, before access to the SO, a client process has to receive the required lock on that SO. When the method execution is completed,

if the SO's state has been changed, other replicas of that SO must also be updated according to a consistency management protocol (Mosberger 1993). In this study, we present two consistency management protocols for replicated objects: write invalidate and write update.

With ***write-invalidate replication***, a write operation invalidates all current copies of a SO. One node is designated as the owner of an object, and this node always has the most up-to-date value. The owner of an object may change during execution. Other nodes get a copy of the SO when they initiate a read operation. As long as this copy remains valid, all read operations can access this copy. Whenever a write operation has to be executed, all current copies are invalidated, so that nodes have to fetch the new value on the next operation. The runtime system has to guarantee that the node that executes the write operation becomes the owner of the object so that no two write operations can execute concurrently. More advanced implementations allow multiple writers and order the updates when a node fetches a new copy but we have not included this scheme in our model.

With ***write-update replication***, write operations are multicast to all nodes that have a replica, and all replicas are updated in place by executing the operation locally. It has been proven that using a total ordering on the multicast messages guarantees sequential consistency. Furthermore, it is possible to integrate this update mechanism with the access mechanism for non-replicated objects. Since the function is shipped to all nodes that have a replica of the object, the runtime system must have access to the operation code and the arguments.

5. Object Generation

The programmer develops applications using the Java Programming Language without any language extension, nor system support classes, in a manner similar to developing a centralized application program. First, the code for each different fragment class which constitutes a DCO is written. (To provide distinction between this type of classes and normal Java classes, we use prefix *Sub*). After this step, interface declarations are written for these classes. An access mode keyword, which indicates type of the method, reader-R or writer-W, is assigned to each method in the interface declaration.

Since, an application is developed in a centralized manner, it does not deal with synchronization and consistency issues. A second step in the configuration is to associate synchronization and consistency mechanisms to each class. In other words, class file for the control object is generated in this step.

After the creation of the control object's class file, an application can be configured for distribution by creating class file of the connective object, which is generated from the interface definition of the fragment class.

6. Conclusion

In this paper, we have described the design and implementation issues of a new object model called ***distributed composite objects***. The DCO model is based on the idea that the implementor of an object can distribute the state of the object among multiple SOs which can be placed on different sites, may be replicated, or may even be composite objects themselves.

We also have given details on DCOBE, a toolkit that facilitates the development of internet-wide distributed applications, which use the DCO model. In DCOBE, applications are dynamically deployed to the requesting nodes from the node that hosts the application; thus we do not require applications to be installed on a machine prior to execution. In addition, a large-scale wide area application should be dynamically adaptable by adding or removing components. DCOBE allows addition, removal, or change of a SO of the DCO without affecting clients that are presently bound to that object.

DCOBE allows shared SOs of the DCO to be replicated on cooperating nodes, thus enabling local invocation on distributed objects and thus reducing latency.

In DCOBE, a distributed collaborative application can be developed as if it were to be run centralized. Distribution, synchronization and consistency are programmed separately from the application code. This also enables the execution of existing centralized applications in a distributed environment.

References

Neuman, B.C., (1994), "Scale in Distributed Systems", In *Readings in Distributed Computing Systems,* IEEE Computer Society Press.

Mosberger, D., (1993), "Memory Consistency Models", *Operating Systems Revie*w, pages 18–26, Jan.

Rosenberry, D., Kennedy D., and Fisher, G., (1992), *Understanding DCE*, O'Reilly and Associates, Sebastopol, California.

Eddon, G., and Eddon, H., (1998), *Inside Distributed COM*, Microsoft Press, Redmond, WA.

Gosling, J., Joy B., and Steele G., (1996), *The Java language Specification,* Addison-Wesley Developers Press, Sunsoft Java Series.

Makpangou M., Gourhant, Y., LeNarzul J.P., and Shaphiro, M., (1994), "Fragmented Objects for Distributed Abstractions", In T.L. Casavant and M. Singhal, (eds.), *Readings in Distributed Computing Systems,* pp. 170-186, IEEE Computer Society Press.

Steen, M.V., Homburg, P., and Tanenbaum, A., (1999), "Globe: A Wide-Area Distributed System", *IEEE Concurrency,* 7(1), pp. 70-78, Jan.-Mar.

Object Management Group (OMG), (1999), *The Common Object Request Broker Architecture and Specification. Revision 2.3.1.* OMG Document formal/99-10-07, Object Management Group, Framingham, MA.