A Framework for ADS Application Software Development Based on CORBA

Stephen S. Yau and Shaowei Mao
Computer Science and Engineering Department
Arizona State University
Tempe, AZ 85281-5406 USA
email: {yau, mao}@asu.edu

Abstract

Autonomous Decentralized System (ADS) which has the characteristics of on-line maintainability, on-line expandability and fault-tolerance has been successfully used in many distributed computing domains, such as factory automation, traffic control, office automation, nuclear power plants. In order to realize many benefits of object-oriented software development, a framework for ADS application software development based on Common Object Request Broker Architecture (CORBA), which is a set of standards for object systems in heterogeneous distributed environments, is presented. In this framework, CORBA is extended and built over ADS system software. A CASE environment for ADS application software development based on CORBA is also presented.

Keyword: Autonomous Decentralized System, application software development, CASE environment, CORBA, framework.

1. Introduction

Autonomous Decentralized System (ADS) [1] [2], which has the characteristics of on-line expandability, on-line maintainability and fault-tolerance, has been successfully used in many distributed computing domains, such as factory automation, traffic control, office automation and nuclear power plants. In order to have effective ADS application software development, Distributed Object Computing (DOC) [3] [4] seems to be a very promising approach because it provides a much better way to capture the inherently decentralized nature of distributed computing and many benefits of object-oriented technology (encapsulation, reuse, portability, and expandability) for distributed application software as for stand-alone application software. Object-oriented middleware, such as Object Management Group's (OMG's) Common Object Request Broker Architecture (CORBA) [5] [6] and Microsoft's Distributed Component Object Model (DCOM) [7], is an enabling technology for DOC. CORBA is an emerging industry standard for distributed object systems. DCOM is an application-level protocol for object-oriented remote procedure calls for distributed, component-based systems. In order to use DOC in ADS application software development, ADS system software should be enhanced to support distributed objects. One of the approaches to supporting distributed objects in ADS system software is to add CORBA over ADS system software. Since CORBA is a common distributed infrastructure supported by many vendors, this approach also guarantees the interoperability with other CORBA-compliant distributed computing systems.

In this paper, we will present a framework for ADS application software development based on CORBA. In our framework, we will build CORBA over ADS system software, extend CORBA to retain ADS characteristics, and then provide a CASE environment for ADS application software development based on CORBA. We will use an Automatic Teller Machine (ATM) as an example to illustrate how our framework will work.

2. Background

Proceedings of the 3rd Int'l Symposium on Autonomous Decentralized Systems (ISADS '97)
0-8186-7783-X/97 $10.00 © 1997 IEEE

In this section, we will provide an overview of CORBA and ADS system software architecture for the sake of completeness.

2.1. Overview of CORBA

CORBA [5][6] is a set of standards which enables objects to transparently make and receive requests and responses in a distributed environment. It is based on OMG object model, where a client can send a message to an object cross address space. The client accesses the services of the object by a well-defined encapsulating interface which isolates the client from the implementation of the services and the object interprets the message to decide what service to perform. The object model describes object semantics and object implementation. Object semantics is related to the client, and includes such concepts as object and object reference, requests and operations, types and signatures. Object implementation includes such concepts as methods, execution engines, and activation.

The interface of an object in CORBA is defined by OMG Interface Definition Language (IDL). IDL is a declarative language which describes the services of the objects and needs to be mapped into particular programming languages. IDL mappings to C, C++, and Smalltalk have been specified by OMG. An IDL compiler is needed to bind IDL to a particular programming language.

The message passing between a client and an object implementation is performed by the Object Request Broker (ORB). The ORB, together with object adapters, provides all the mechanisms required to find the object implementation for the request, to prepare the object implementation to receive the request, to transfer the request, to activate and deactivate the object implementation, and to create and manage object references. The ORB functionality is defined by the ORB interface using pseudo-IDL and its binding to C++ is specified by OMG.

An object adapter specifies how an object implementation access services provided by the ORB. There are several object adapters with interfaces that are appropriate for specific kinds of objects. Basic Object Adapter is an object adapter specified by OMG, and can be used for most ORB objects with conventional implementation.

OMG also specifies a set of services, called *Common Object Services*, to provide the basic functions for using and implementing objects and a set of services, called *Common Object Facilities*, to provide general purpose capabilities useful in many applications.

2.2. ADS System Software Architecture

Conceptually, ADS has the feature that every software subsystem has autonomy to manage itself and coordinate with other software subsystems [1] [2]. Coordination is achieved by communicating with other software subsystems through Data Field (DF), in which the data circulates and software subsystems select the data according to the content code. The software subsystem in ADS is called *Atom*. Every Atom is connected to Data Field. Data also can circulate among the software modules in Atom. Data Field in Atom is called *Atom Data Field.*

Atom consists of not only the application software, but also its own management system software called *Autonomous Control Processor* (ACP). Each ACP is self-contained, operates according to its local information and communicates asynchronously with other ACPs by message broadcast in Data Field. Data Field Management Module in ACP is responsible for receiving the data from Data Field and sending the data into Data Field. The application software module is driven by the data from Data Field according to its content code. It is activated by Execution Management Module in ACP, receives the data from Data Field, processes the data and sends the resultant data to Data Field. Therefore, each ACP can operate even when other ACPs fail, and fault tolerance at system level is achieved. Fault tolerance at the application software level is supported by replicating the application software modules in different Atoms with a threshold-voting mechanism. The replica application software modules process the same data from Data Field and send the resultant data to Data Field. Data Consistency Management Module in each ACP selects the correct resultant data from the replica application software modules by the threshold-voting mechanism. On-line expandability is supported by the construction management module as an application software module. The construction management module can independently install the application software module to an Atom without interrupting other

Proceedings of the 3rd Int'l Symposium on Autonomous Decentralized Systems (ISADS '97)
0-8186-7783-X/97 $10.00 © 1997 IEEE

Atoms. On-line maintainability is supported by the Built-In Test module (BIT) in each ACP and the EXternal Tester module (EXT) as application software modules. The BIT and EXT can independently test the application software modules and decide to start the operation of the application software modules according to the test result. The ACP software architecture is shown in Fig. 1.

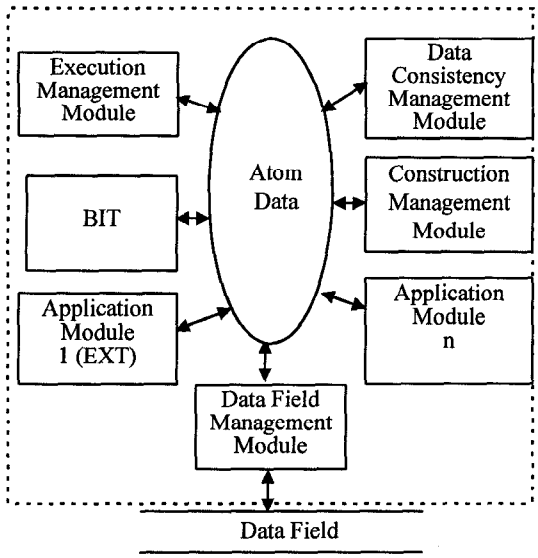

Fig. 1. The ACP software architecture

3. Building CORBA over ADS System Software

Since ADS system software provides a communication layer and the mechanisms to locate, activate and deactivate an application software module [1] [2], we decide to implement CORBA over ADS system software. Specifically, we will implement the ORB functionality by a pair of libraries, one for client application, one for server application, and ORB daemon. ORB daemon is implemented by a few system software modules from *ACP*, and it is responsible for locating, activating and deactivating objects, initiating and receiving remote object request. The client library can initiate the remote object request by forwarding it to ORB daemon and the server library can initiate and receive remote object request through ORB daemon. We will also implement Interface Repository (IR) as a system-resident ADS application. The software architecture of our CORBA implementation is shown in Fig. 2.

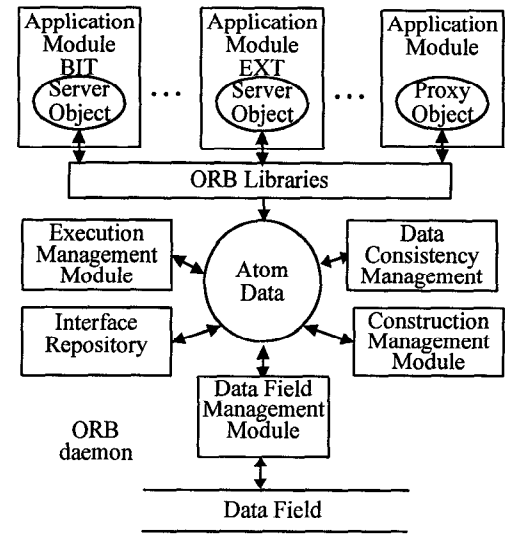

Fig.2. Software architecture of our CORBA

Our implementation also includes an IDL compiler. The IDL compiler provides a language binding from OMG IDL to C++. It generates C++ stub code for client application, implementation skeleton for server application and type information for IR. In our implementation, a client corresponds to an ADS application software module. The default behavior of C++ stub code generated by the IDL compiler is to marshal the remote object request, forward it to the ORB daemon, receive the result from the ORB daemon, and unmarshal it. All these functions are encapsulated in a proxy object generated by the IDL compiler for every IDL interface. The proxy object provides the same methods as the remote server object so that the client invokes these methods of the proxy object just the same as the remote server object. The proxy object also provides other functions in our implementation, such as data consistency check discussed in the next section. The default behavior of the implementation skeleton is to register implementation definition to the ORB daemon, to create or destroy the object reference, and to prepare to receive the requests from the ORB daemon and send the result to the ORB daemon. In our implementation, a server corresponds to an ADS application software module. Each application software module can contain multiple active objects of a given implementation. The Interface Repository provides type information for other application modules to check type dynamically.

Execution Management Module in *ACP* is modified to support part of BOA's functionality,

Proceedings of the 3rd Int'l Symposium on Autonomous Decentralized Systems (ISADS '97)
0-8186-7783-X/97 $10.00 © 1997 IEEE

which is to maintain implementation repository, bind the client to the server object in the software application module and activate the application software module according to the information in the implementation repository in the response to the remote object requests.

4. Extending CORBA to support ADS characteristics

In order to maintain ADS characteristics of on-line maintainability, on-line expandability and fault-tolerance, CORBA needs to be extended to support object group, state transfer, data consistency check and object migration.

4.1. Object Group

Fault-tolerance can be achieved by replicating the server object in different *Atoms* to form an object group to respond to the requests from a client. Basically, there are two kinds of replica. One is Active Replica, in which every server object in the object group responds to a request. If at least one object in the object group works, the client gets the services. However, in Active Replica, each object in the object group needs to be consistent with each other. If one object in the object group fails, it needs to be restored to the same state of the other objects. Another is Passive Replica in which only one object as the primary object responds to the requests from a client, other objects act as the backup objects. Once the object fails to provide the services, one of the backup objects takes over and continues to provide the services. In Passive Replica, it takes time for the backup object to get to the same state as the primary object. Active Replica Object Group is correspondent to replica application software modules in ADS. In our implementation, we select Active Replica Object Group, and proxy object in a client is bound to an active replica server object group instead of a single object implementation shown in Fig. 3.

Group object communication protocol in our CORBA implementation is a broadcast protocol since ADS is built over a reliable LAN which provides broadcast. In our group object communication, each message is broadcasted with a header, in which there is a message identifier containing the identity of the broadcasting ORB daemon, a message sequence number, and content code. The operation of

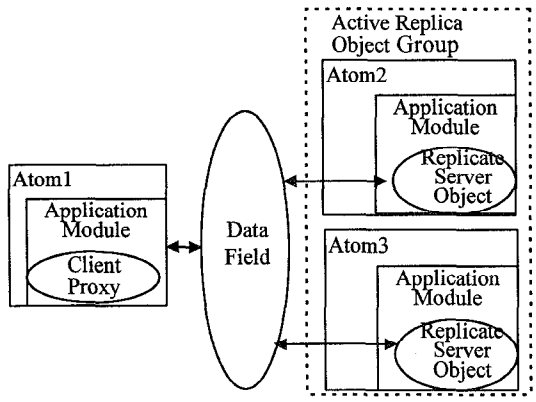

Fig. 3. Active Replica Object Group

group object communication protocol is illustrated with the following scenarios:

- When a client calls an object's binding operation, the ORB daemon A broadcasts a message with reserved content code for the server object binding. The data field of the message includes interface name, object name.
- The ORB daemon B which receives the A's message checks if there is the required server object in its own host which can provide the services and broadcasts an acknowledgment message indicating whether it can provide the services. The ORB daemon C on receiving the B's acknowledgment message will check if it has received A's message. If so, it acknowledges A's message by broadcasting a message indicating whether it can provide the services. If not, it broadcasts a negative acknowledgment message indicating that it has not received A's message. Any ORB daemon which receives C's negative acknowledgment message will rebroadcast A's message if it has received A's message.
- The ORB daemon A collects all acknowledgment messages within some time interval, builds an object group for this client, assigns an In content code and Out content code for the object group and broadcasts a message to inform all the ORB daemons of the creation of an object group. The message includes the assigned content codes and the group member list. Each member in the object group will use the same content codes in the following communication.
- All ORB daemons which can provide services activate ADS application software module in which the server object in the object group resides to prepare for receiving the requests from the client.

Proceedings of the 3rd Int'l Symposium on Autonomous Decentralized Systems (ISADS '97)
0-8186-7783-X/97 $10.00 © 1997 IEEE

- The client initiates a request by asking the ORB daemon A to broadcast the request.
- An ORB daemon D in the object group receives the A's request, and checks the sequence number in the message to see if it is more than one greater than the largest sequence number it received from source A. If not, it processes the request, records the sequence number in the message, calls the method of the server object, and broadcasts the acknowledgment message in which the result is packed. Otherwise, it has missed some messages from source A, and broadcasts an acknowledgment message indicating that it cannot provide the service and asks for state transfer. State transfer is equivalent to method invocation on one of the other objects in the same object group to transfer to that object's state. After receiving D's acknowledgment message, the ORB daemon E will respond to A's request in the same way as D by broadcasting acknowledgment message for A's request if it has received A's request. If it has not received A's request, it will broadcast a negative acknowledgment message for A's request. Any ORB daemon which received E's negative acknowledgment message (including A) will rebroadcast A's request if it has received A's request.
- The ORB daemon collects all acknowledgment messages within some time interval, uses threshold-voting mechanism to select the most-likely correct result, and forwards it to the client.
- When the client exits, it asks the ORB daemon to broadcast a message to inform each member in the object group to release the resource for the object group.

The group object communication protocol described above can tolerate transient transmission failure and provides the total order of the messages and atomic delivery of the messages to server-client group communication over LAN in our CORBA implementation. Hence, it guarantees that the state of the object group is consistent if the server object does not fail.

4.2. State Transfer

If a new object joins the object group or a object recovers from failure, the object needs to get to the current state of the object group. The oldest object in the group will send its state to the new object. During this period, the ORB daemons will delay all the requests to the object group until the state transfer is completed.

4.3. Object Migration

Sometimes, a member of the server object group needs to migrate to some other machine in order to achieve load-balancing and fault-tolerance. The migration must not affect the services of the server object group. A possible solution will work as follows.

The server object leaves the server object group and the executable image of the server object is transferred to another Atom by Construction Management Module in ACP. The ORB daemon in that Atom will activate the server object application module. The server object will rejoin the server object group again and restore to the current state of the server object group by state transfer.

5. ADS Application Software Development Framework based on CORBA

Our framework for ADS application software development based on CORBA is shown in Fig. 4. It is modified from our object-oriented software development framework for ADS [8] [9].

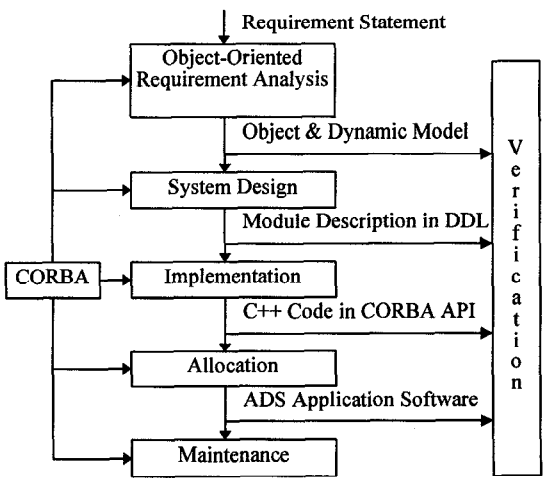

Fig. 4. Our object-oriented framework for ADS application software development

Our framework has the following phases: object-oriented requirements analysis, system design, implementation, allocation, verification

Proceedings of the 3rd Int'l Symposium on Autonomous Decentralized Systems (ISADS '97)
0-8186-7783-X/97 $10.00 © 1997 IEEE

and maintenance. We start ADS software development with a set of requirement statements, which is transformed into the object model and dynamic model using object-oriented requirements analysis (OORA) technique. The object model shows classes and their hierarchical structures derived from the knowledge about application domain and the requirement statements. The dynamic model shows a finite-state machine model for each class in the object model. The object model and dynamic model are represented by the module description in Design Description Language (DDL) [8] in the system design phase. Then, each module design in DDL is implemented in C++ and the implemented modules are allocated to processors. At the end of each phase, the adequacy of the results of each phase is verified. We have a CASE environment to support all these phases. Our CASE environment consists of CASE tools for object-oriented analysis and system design [9], object clustering [10] and module allocation [11].

In our new framework, we add the support for CORBA-based application software development in all the phases and our CASE environment. In the OORA phase, we will emphasize the role of objects, persistency of objects, object life cycle, association relationship among objects, and data flow between the producer object and the consumer object in order to provide enough information for object clustering [10] and organize class inheritance hierarchy to make full use of CORBA common object services [6]. In the design phase, our object-clustering CASE tool [10] will cluster the objects to the modules, identify the objects which export their interfaces to other modules, generate IDL interfaces according to the object information in the object model, and use our IDL compiler to generate IDL stubs and implementation skeleton for the implementation phase. In the implementation phase, each class is implemented in C++ using IDL stubs and implementation skeleton generated in the previous phase, and then each module is implemented in C++ using CORBA APIs. In the allocation phase, our CASE tool will emulate the real distributed systems by the workstation cluster, allocate the modules to the workstations according to our allocation algorithm [11], replicate the same module to different Atoms to achieve fault-tolerance, verify the consistency between the modules and the Atoms, run the application to tune our allocation. During the maintenance phase, the object implementation in each module can be enhanced without affecting the clients which use the services of the object because CORBA completely separates the interface from the object implementation.

6. An example

In this section, we will use an Automatic Teller Machine (ATM) example [9] to illustrate how our framework supports ADS application software development based on CORBA. The software requirements for the ATM system are specified as follows:

Develop the software to support a computerized banking system with automatic teller machines (ATMs) to be shared by a consortium of banks. Each bank has its own computer to maintain its accounts and make updates to accounts. ATMs communicate with a central computer of the consortium. An ATM accepts a cash card, interacts with the user, communicate with the central computer to process transactions, and dispenses cash. For simplicity purpose, the system is assumed to have two ATMs, two Banks and one consortium.

For the object-oriented requirements analysis, we obtain the object and dynamic models of the ATM system according to the procedures in our framework [9]. The object model is shown in Fig. 5.

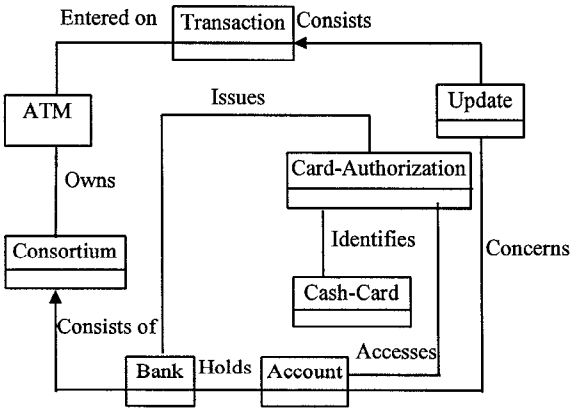

Fig. 5. The object diagram for the ATM system

For the system design, we first identify objects and inter-object communications. In this example, the objects are ATM1, Cash_Card1, ATM2, Cash_Card2, Consortium, Bank1, Transaction1, Bank2, Transaction2, Account1,

Proceedings of the 3rd Int'l Symposium on Autonomous Decentralized Systems (ISADS '97)
0-8186-7783-X/97 $10.00 © 1997 IEEE

Update1, Card_Authorization1, Account2, Update2, and Card_Authorization2 as shown in Fig. 6. The inter-object communications are showed by the lines with arrows in Fig. 6. In the next step, we cluster the objects into several modules shown in Fig. 6, where the objects in the gray boxes are identified as the objects which export their interfaces to other modules and IDL interfaces are generated by IDL generator according to the information in the object model. Then, IDL stubs and implementation skeletons are generated using our IDL compiler.

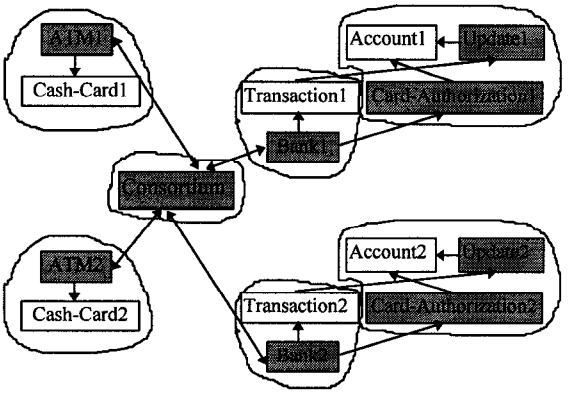

Fig. 6. The object clusters of the ATM example

For the implementation phase, the IDL stubs are used by client objects to invoke the methods of the remote objects, and implementation skeletons are used to implement the remote objects. For the allocation phase, we allocate the modules to different *Atoms* as shown in Fig. 7. The server object Consortium is replicated to two different *Atoms* to form an active replica object group to provide the services to other modules to achieve fault tolerance. Bank1 module and Update1 module are allocated to the same *Atom* since communication between these two modules are heavy. Similarly, Bank2 module and Update2 module are allocated to the same *Atom*. The two ATM modules are allocated to two different *Atoms*.

7. Discussion and Future Work

In this paper, we have presented a framework for ADS application software development based on CORBA. We are implementing the CORBA over ADS system software, extending the CORBA to achieve ADS characteristics, and modifying our CASE environment for ADS application software

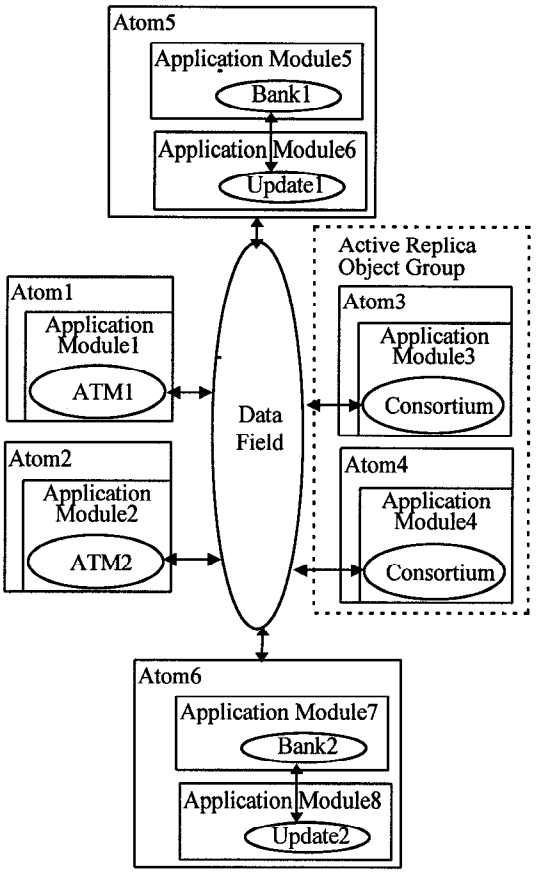

Fig. 7. Module Allocation

development based on CORBA. We also plan to incorporate a performance monitor into the ORB daemon in our implementation to obtain communication traffic information among different modules to dynamically support load-balancing among the hosts and reconfiguration of the application software modules by object migration and achieve interoperability with other CORBA implementations in the near future.

Since ADS has been successfully used in a number of different domains, we plan to analyze existing ADS applications in the different domains using our framework, migrate existing ADS applications to our CORBA-compliant development and running environment, and build domain-specific frameworks for ADS applications.

In order to migrating existing ADS application software modules to our CORBA-compliant development and running environment, we must encapsulate ADS application software modules by IDL interface. Basically, there are two ways for encapsulation. One is coarse encapsulation, which specifies the

Proceedings of the 3rd Int'l Symposium on Autonomous Decentralized Systems (ISADS '97)
0-8186-7783-X/97 $10.00 © 1997 IEEE

functionality of every ADS application software module by IDL interface. The object implementation just responds a request by executing the executable image of the application software module with different parameters. Another is fine encapsulation, which factors out the common functions among different ADS application software modules, encapsulates it as the common object services available to every object, and rewrites ADS application software modules as the application objects using common object services.

Since many ADS applications are time critical, we will extend our CORBA-based framework for real-time systems. In order to meet rigid temporal requirements in real time systems, we must have high-performance CORBA implementation which makes full use of underlying communication link, like Asynchronous Transfer Mode (ATM), and provide Quality of Services(QoS) for distributed objects accessible by ADS application developers.

Acknowledgment

This work is supported under the collaborative research agreement between Arizona State University and Hitachi, Ltd.

References

1. K. Kawano, M. Orimo and K. Mori, "Autonomous Decentralized Systems: Concept, Data Field Architecture and Future Trend", *Proc . First Int'l Symp. on Autonomous Decentralized Systems*, 1993, pp. 28-34.
2. K. Mori, et. al., "Autonomous Decentralized Software Structure and its Application", *Proc. FJCC'86*, 1986, pp. 1056-1063.
3. Douglas C. Schmidt and Steve Vinoski, "Introduction to Distributed Object Computing", *C++ Report*, January 1995.
4. Douglas C. Schmidt and Steve Vinoski, "Modeling Distributed Object Computing", *C++ Report*, February 1995.
5. OMG, *The Common Object Request Broker: Architecture and Specification*, 2.0 edition, July 1995.
6. OMG, *CORBAService: Common Object Service Specification*, 95-3-31 edition, March 1994.
7. Microsoft, *Distributed Component Object Model Protocol – DCOM/1.0*, http://ds1.internic.net/internet-drafts/draft-brown-dcom-v1-spec-00.txt.
8. S. S. Yau and G.-H. Oh, "An Object-Oriented Approach to Software Development for Autonomous Decentralized Systems", *Proc . First Int'l Symp. on Autonomous Decentralized Systems*, 1993, pp. 37-43.
9. S. S. Yau, et. al., "An Object-Oriented Approach to Software Development for Autonomous Decentralized Systems", *Proc . Second Int'l Symp. on Autonomous Decentralized Systems*, 1995, pp. 405-411.
10. S. S. Yau and H. Ying, "A Clustering Algorithm for Object-Oriented Development of Distributed Computing System Software", *Proc. 5th IEEE Workshop on Future Trends of Distributed Computing Systems*, 1995, pp. 274-281.
11. S. S. Yau and V. R. Satish, "A task Allocation Algorithm for Distributed Computing Systems", *Proc. 17th Int'l Computer Software & Applications Conf. (COMPSAC 93)*, 1993, pp. 336-342.
12. Bhavani Thuraisingham, Peter Krup and Victor Wolfe, "On Real-Time Extensions to Object Request Brokers: A Panel Position Paper", *Proc. Second Int'l Workshop on Object-oriented Real-time Dependable Systems*, 1996.
13. Gotter Sean, *Inside Taligent Technology*, Addison-Wesley, 1995.
14. INOA Technologies Ltd, "The Orbix Architecture", January 1995, http: //www-usa.iona.com/www/Obix/arch/Summary.html.
15. John A. Zinky, David E. Bakken and Richard Schantz, "Overview of Quality of Service for Distributed Objects", http: //www.bbn.com/offerings/dcutu/duduse/ Dualuse-final.html.
16. Jon Siegel, *CORBA Fundamentals and Programming*, John Wiley & Sons, 1996
17. Kenneth P. Birman and Robbert Van Renesse, "Reliable Distributed Computing with the Isis Toolkit", IEEE Computer Society Press, 1994.
18. Paul D. Ezhilchelvan, Raimundo A. Macedo and Santosh K. Shrivastava, "Newtop: A Fault-Tolerant Group Communication Protocol", http: //arjuna.ncl.ac.uk/arjuna/papers.html
19. P. M. Melliar-Smith, et. al, "Broadcast Protocols for Distributed Systems", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 1, pp. 17-25, January, 1990.

Proceedings of the 3rd Int'l Symposium on Autonomous Decentralized Systems (ISADS '97)
0-8186-7783-X/97 $10.00 © 1997 IEEE