ABSTRACT

This paper proposes a simulation framework that effectively uses the Common Object Request Broker Architecture (CORBA) technology to provide a flexible, extensible, platform-independent and language-independent simulation environment in distributed environment. In distributed computing, objects may be distributed on different computers through out a network, living within their own address space outside of an application, and yet appear as though they were local to an application. In order to reduce the network program complexity, and to increase the interoperability, communication between objects in a heterogeneous environment, a middleware platform is needed. The CORBA specification is such a middleware platform by the Object Management Group. Object Request Broker (ORB) enables to make transparently make and receive requests and responses in a distributed environment. MICO2.3.7 is a fully complaint CORBA 2.3 implementation is used to develop the application.

In the simulation model, a sea scenario is considered. The Vessel Dynamics Simulator (VDS), is an independent system, which simulates the scenario. The scenario consists of a number of vessels - that is different interacting entities like ships and submarines, and their movements in a particular area of operation. These entities must coordinate in real time to make decisions to defend from an attack or to launch an offensive. The user specifies the scenario to be simulated by giving various parameters like position, velocity, bearing and diving rate at each point in the way path. Navigation information like position, heading, speed etc. must be known consistently and to a high degree of accuracy throughout the ship.

The applicability of the CORBA features like BOA, POA, and Event Service to data transfer in the distributed environment is used for the implementation of the application. The Basic Object Adapter (BOA), is used to activate the object implementations when their service is requested by a client. Portable Object Adapter (POA) supports for the transparent activation of objects, single servant to support many object identities and delegate request for a non-existent servant through a default servant or through a servant manager. Event channels are like servers. These are objects which performs the work of delivering events from producers to all interest customers. Events are delivered through two mechanisms called push and the other as pull mechanism.
2. OBJECTIVE

The objective of the paper is to develop an application for simulation in a distributed environment using CORBA and to use the different features available in the specification. Object Request Broker (ORB) enables to make transparently and receive requests and responses in a distributed environment. MICO 2.3.7 is a fully complaint CORBA 2.3 implementation is used to develop the application.

The application is aimed to develop a simulator namely, Vessel Dynamic Simulator (VDS), which is part of a distributed simulation environment. The environment includes different interfacing entities including submarines and ships. Vessel dynamics simulates the movement of vessels like ships and submarines within a specified sea area. These entities must coordinate in real time to make decisions to defend from attack or to launch an offensive. Hence, the parameters of vessels such as position, direction of movement etc. must be known to a high degree of accuracy. The parameters of vessels such as position, direction of movement etc. are to be sent to the other component of the system namely, Display System, where the output of the system is presented. Of the \( n \) vessels one vessel is designated as own vessel and relative to the own vessel, the distance (range) and direction (bearing) of all other \( n-1 \) vessels are computed.

Different features of CORBA for data communication can be used. The Basic Object Adapter (BOA), is used to activate the object implementations when their service is requested by a client. Another way of activating the objects is through Portable Object Adapter (POA). Event channels are like servers. These are objects which performs the work of delivering events from producers to all interest customers. Events are delivered through two mechanisms called push and the other as pull mechanism.

3. DESIGN AND IMPLEMENTATION

The system design phase focuses on the detailed implementation of the system under requirement specification.

Since the Vessel Dynamics System (VDS) as well as the Display System are independent systems over the network which operate in parallel, during the simulation run the vessel parameters are generated continuously at a time interval, \( dt \), and the Display System needs to receive this data.

MICO 2.3.7, a fully complaint CORBA 2.3 implementation, is used to develop the application. Since the implementation of objects can be done in different programming languages, the specification of objects interface and the implementation are to be separated. The implementation is done using C++ and the interface is defined using an Interface Definition Language (IDL).

A MICO application can be created by generating files for the client, server and idl. The client file is of the form “client.cc”, server is of the form server.cc and idl as “idlfile.idl”. The IDL looks like a C++ reduced classes. The idl is created with ‘.idl’ extension. Then it is compiled using the idl compiler and as a result two files are generated, one with extension .h and the other with extension .cc. The .h files contain the class declarations for the base class and .cc contains implementation of those classes and some supporting code, in the selected programming language. During the process of compilation, it also does the marshalling and de-marshalling.

In order to access the ORB applications, ORB initialization function has to be called first. Then only the ORB can access the command line arguments.

The Basic Object Adapter (BOA), is used to activate the object implementations when their service is requested by a client. And also the server communicates with the ORB through BOA. BOA uses implementation repository to take a decision on how to activate the object. Similar to ORB initialization BOA is also to initialized.

Information about objects is stored in Implementation Repository which contains methods for creating, deleting and finding entries. For each server, exactly one entry is for the name, activation mode, shell command and list of repository ids. From this information, BOA can take a decision on which mode has to be selected.

Another way of activating the objects is through Portable Object Adapter (POA). Mainly it supports for the transparent activation of objects, single servant to support many object identities and delegate request for a non-existent servant through a default servant or through a servant manager.

Event channels are like servers. These are objects which performs the work of delivering events from producers to all interest customers. Events are delivered through two mechanisms called push and the other as pull mechanism.

The data communication has done in two methods. In order to generate the data at different time intervals, the data collected from each vessel which are assigned as vessel ids are collected and the required calculations can be made.
3.1 Method 1
To achieve this, the Display System can be implemented as a CORBA object, with two methods, one for receiving the data from VDS, and another for processing and presenting the data in the required format. The IDL for this implementation looks as follows,

typedef struct VesselParam
{
    float x_time;
    float y_time;
    short vessel_id;
};

interface GetData()
{
    oneway void GetVesselCoordinates (in VesselParam VP);
    short ProcessDisplay(out VesselParam VPI);
};

GetVesselCoordinates() and ProcessDisplay() are user defined functions to collect the vessel parameters and to display the received data in the display system.

Using the Basic Object Adapter, the above interface is implemented in the various modes like shared, unshared, persistent and pre-method. It is found that the two methods when invoked on the object run either in two different processes (or) run synchronously in a single process. This can result in the failure of requests to send the data, if the server is unavailable i.e., while the other method is still under execution. BOA supports several activation modes. An object implementation has to use special BOA functions according to the selected activation modes.

Shared activation is usually the default activation for CORBA implementation. In this mode of activation, all the objects have the same server name are contained and managed within one server. This means that when the server process is launched, all the objects that are hosted by the server are instantiated. But a mechanism would have to be found to allow clients to be services simultaneously. This would allow a server to respond to more than one client at the same time. That is the server has to be multithreaded.

In the unshared activation mode, each object has to be registered individually. This means that each object is going to run its own server process. This activation might be useful where individual objects need to be isolated in terms of their state.

Per-method activation is activation mode enables a process to be registered by a method. When invocation comes into the method a new process can be instantiated and executed.

In the case of Persistent server activation servers are to be launched manually by an operator. Further, the server names do not have to be registered with the implementation repository. Persistent servers have to be manually managed by an administrator. One of the ways this model can be used is for servers that manage other server processes.

Another method of data communication is through the use of Portable Object Adapter (POA). In this case, we can set different POA policies and a better result can be obtained. The thread-per-method policy supported by POA can be used to overcome the above mentioned drawback of BOA. MICO 2.3.7 does not implement this policy under POA.

3.2 Method 2
A standard CORBA request results in the synchronous execution of an operation by an object. If the operation defines parameters and return values, data is communicated between client and the server. In this application, a more decoupled communication model between object and the client is required. To achieve this intermediate object is defined between the VDS and the Display system, which facilitate the communication between the two systems. The IDL for this intermediate object looks as follows,

typedef struct VesselParam
{
    float x_time;
    float y_time;
    short vessel_id;
};

interface GetData()
{
    oneway void SendVesselCoordinates (in VesselParam VP);
    short ReceiveVesselCoordinates (out VesselParam VPI);
};

GetVesselOrdinates() and ProcessDisplay() are user defined functions to collect the vessel parameters and to display the received data in the display system.

Using the Basic Object Adapter, the above interface is implemented in the various modes. The SendVesselCoordinates() is called by the VDS and it sends a set of data periodically as ‘in’ parameters. These data are continuously received by the display system as ‘out’ parameters by invoking the method ReceiveVesselCoordinates() and the processed data can be displayed graphically.

The objective is to receive the data continuously from the VDS, by the Display system. If we are using the BOA as the adapter, in order to receive the data correctly a synchronization mechanism has to be made. In the case of POA, we can set POA policies.

The event channel provides the most flexible model for event delivery. The event channel is a proxy for event producers. In that respect, the introduction of channel is
the producer of events. Since once a consumer is registered with the supplier, the client need not bother about the originator and, it can get the data from the event channel. That is, here the VDS and Display system is registered with event channel, and whenever VDS sends data, the Display system can receive the changes.

The event channel offers the following benefits over a more direct model of Event communication. The event channel offers a much more scalable system than those offered by direct events communication. Scalability in this context means that a number of suppliers and interested consumers can be arbitrary. There is a greater decoupling between the client and server. This means that the event channel needs to support only certain type of events. A client subscribing to these need not bother about the originator of these events. On the other hand the supplier need not be concerned about what is ultimately consuming the events.

Events can be made persistent and transactional. This allow client to play back on event. Another major benefit is the possibility of logging events when they happen. In this fashion, if a failure occurs, the Events Channel can re-initialize it to remove from the failure.

The above interfaces defined in IDL, were implemented using the Object oriented language C++ on Linux operating system. These are integrated with the existing VDS system, which acts as a client in both the above methods. The display system is implemented in C++ using Qt Designer for graphical presentation.

4. CONCLUSION

Distributed computing extends an object oriented programming system by allowing object to be distributed across heterogeneous network so that each of these distributed object components interoperate as a unified whole. These objects may be distributed on different computers through out a network, living within their own address space outside of an application, and yet appear as though they were local to an application. In order to reduce the network program complexity, and to increase the interoperability, the communication between these distributed systems is facilitated by CORBA.

The MICO 2.3.7, which is an implementation for the CORBA 2.3 specification is used to study the features of CORBA. CORBA is an emerging standard for distributed object computing middleware. ORB is the core of CORBA architecture. An ORB eliminates many tedious error-prone, and non-portable aspects of developing and maintaining distributed applications by automating common network programming task such as object location, object activation, parameter marshalling, fault recovery, and security. In CORBA specification, objects are the basic entities considered. CORBA defines an interface to an object by defining it by a declarative language, the Interface Definition Language (IDL).

The applicability of the CORBA features like BOA, POA, and Event Service to data transfer in the distributed environment is covered.

The Basic Object Adapter (BOA), is used to activate the object implementations when their service is requested by a client. And also the server communicates with the ORB through BOA. BOA uses implementation repository to take a decision on how to activate the object. Similar to ORB initialization BOA is also to be initialized.

Information about objects is stored in Implementation Repository which contains methods for creating, deleting and finding entries. For each server, exactly one entry is for the name, activation mode, shell command and list of repository ids. From this information, BOA can take a decision on which mode has to be selected.

Another way used to activate the objects is done through Portable Object Adapter (POA). Mainly it supports for the transparent activation of objects, single servant to support many object identities and delegate request for a non-existent servant through a default servant or through a servant manager.

Event channels are like servers. These are objects which performs the work of delivering events from producers to all interest customers. Events are delivered through two mechanisms called push and the other as pull mechanism.

REFERENCES
