CORBA in a Real-Time Game Environment

Jeroen Broekhuizen (0219428)
Richard Ssekibuule (0440752)
Development of Large Software Systems
14 December 2004

Abstract.
In 2002 Bioware released their newest title and a revolutionary game in the RPG section for that time: NeverWinter Nights. Its success was great and now a days there exists lots of Persistent Worlds (PW). With the NWNX tool even more features, like persistency, hash-tables, etc, can be added to the NWN server. In this paper we try to figure out if it is possible to use CORBA as an underlying protocol to set up a generic communication channel between multiple NWN servers. We do this by implementing a chat system enabling players on different servers to chat with each other and build further on that principle.

1. Introduction

NeverWinter Nights (NWN) is a 3D role-playing game based on the famous Dungeons & Dragons rule set, a fantasy pen & paper role-playing game of Wizards of The Coast. With this game came also the toolset in which the entire game was build. Soon people created their own modules and Persistent Worlds.

Today it is necessary for a persistent world (pw) to fulfill some requirements to get a large player base. To do so Hugo Wallenstein (and later also Jeroen Broekhuizen) implemented the NWN eXtender (NWNX) program. With this program people can dynamically add extra features to the NWN server, think about persistency, hash tables, etc. These features are implemented in separate library files (dll's on windows and as shared object in linux) which can be downloaded from the internet. A set of classes is made available for download so that everybody with experience in C/C++ can create their own extensions.

1.1 Why using CORBA?

CORBA has continued to evolve over the past decade and has been accepted by many industries as their middleware solution (a more detailed discussion about CORBA is given in the next

chapter). It enables the exchange of distributed information, independent of hardware platforms, programming languages and operating systems.

The solid base of CORBA can be of particular use in the NWN environment. During recent developments Jeroen has achieved a solution for the file transmission error in the NWN server (he made a new extension for NWNX which takes care of it). With a CORBA implementation we might even go one step further and send over other objects than characters like the monsters, which is currently not possible.

Another reason why using CORBA is that through years of development CORBA has become very stable and reliable. For the gaming environment these are two very important properties. Nothing is worse than a player who looses his stuff during a transfer to another server due to a failure (whether it's during connection or the actual transmission doesn't really mater).

With CORBA it's also easy to create platform and language independent code which can be reused easily satisfying both Windows and Linux owners. It enables community members which don't have experience in C/C++ but do have experience in another language, which supports CORBA, to contribute to the community.

1.2 Expected results

Using CORBA results in more overhead as the exchange of the data must be reliable. This overhead can be avoided for example by using the socket libraries. But one has to keep in mind that besides speed robustness and reliability are very important too.

At the end of this research we hope to get clear whether or not CORBA is suitable as a communication channel between two or more NWN servers. As a computing paradigm we will implement a simple chatting system which is completely based on the original CORBA specification.

In the rest of this document we will explain what CORBA is. After that a practical case study of a chatting system between multiple NWN servers is discussed and suggestions are made to further improve this system making it a generic platform independent communication channel.

2. What is CORBA?

An in-depth discussion of CORBA falls outside this research and is discusses in numerous good books. What we will do is give the reader a short introduction in the parts of CORBA we used with the implementation of the chatting system, which is discussed later in this document. CORBA is a standard developed by the Object Management Group (OMG) [2].

2.1 Distributed Objects concept

As envisioned by CORBA, distributed objects (DO) are the melding of concepts from two paradigms – client-server (or more precisely, distributed computing) and Object Orientation (OO), with some explicit differences:

- A client knows an object by its interface; Objects are not always local with respect to their clients.
- Dynamic composition may compose objects into new applications.
- Objects hide many of the underlying differences in architecture through encapsulation.

2.2 An ORB as middleware solution

As objects become more pervasive, the scale of their constructs expands from individual processes or run-times to operating systems to distributed operating systems.

One scaling mechanism is an object request broker (ORB). From the application viewpoint it acts as a platform for distributed objects. An ORB may run on top of operating systems that are not object-oriented. In such a case, the ORB encapsulates some aspects of the underlying OS and networking layers just as an object encapsulates its attributes. In this capacity it is middleware. It hides as much of the underlying platform as possible under its object abstraction (see figure below).

Middleware hides the underlying details from the application. It does this by providing a common set of services across all the platforms on which it lives. This means that an independent software vendor (ISV) is able to write to one set of application program interfaces (APIs), making the code that

much portable more portable. The APIs are the interfaces to middleware.

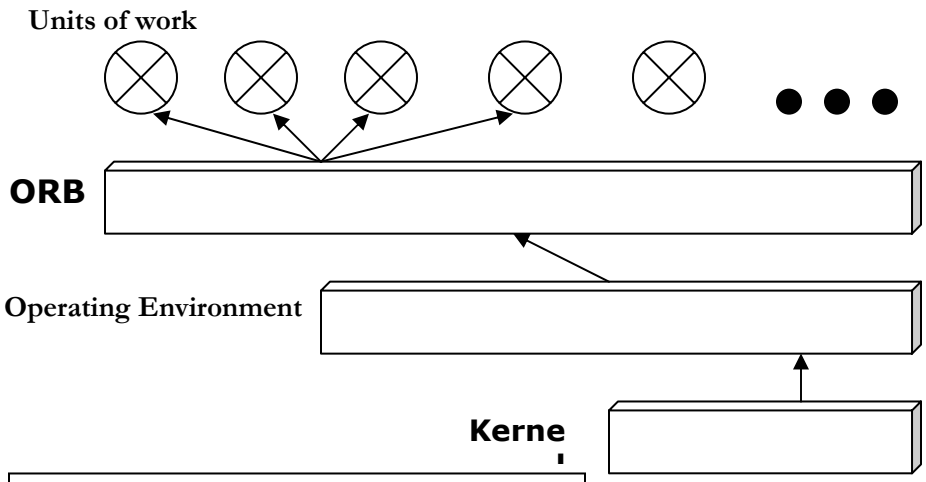

FIG 1: ORB as a layer above OS and network

2.3 Interface definition language

Before a client can use an interface, it should be aware of its structure. In CORBA this is done in the special Interface definition language (IDL). With this language a programmer can describe the interface with its properties and methods.

2.4 Naming service

It offers a way for a client to turn a human-readable name into an object reference, on which the client can subsequently invoke operations in the normal way. This comes particularly in play in networked environments.

3. A case study: chat system

Before we can say anything about CORBA we have to implement a system. With this system we can test the performance and see if it fast and reliable enough for a real-time game environment.

During this research we implemented a communication channel over which players on different NWN servers can chat with each other. In the next sections we will discuss the evolution of this system, and with each step explain why it's good or not.

3.1 OO implementation with CORBA

As explained in the previous chapter in order to use CORBA a programmer has to decide on which ORB he wants to use as underlying CORBA platform. During the implementation we used omniORB [3], which supports numerous platforms including Windows and Linux, other ORB's including those implemented in the Java programming language and it is thread safe. Various people suggest that omniORB is one of the fastest ORB's currently available on the internet, and it is free to use.

There are three programs developed during the research. The first program is the chat server with the actual CORBA implementation. At start up it binds to the name service so that clients are able to call its functionality. The second program is the extension of NWNX. It enables the game to connect to the chat server and call its available features. The last program is a simple graphical desktop client for testing purposes. The code for these programs can be downloaded from cvs [4].

As little time was available, we implemented a very basic chat system. The complete system is based on the IDL file listed below. It defines a structure in which a recipient and the actual message can be stored.

Listing 1

```
struct Message {
   string recipient;
   string text;
};
typedef sequence<Message> Mseq;

interface Chat {
   void addMessage (in Message msg);
   string getMessage (in string name);
};
```

The chat interface contains two methods for adding and retrieving messages. The messages will be stored in an instance of an *Mseq* sequence object. A sequence behaves like a dynamic array and is part of the IDL definition language. You can compare it with a stripped down version of the STL vector (but it hasn't got much helper functions, like removing of an item).

The current implementation is very limited in capabilities due to the constraints of NWNX and the NWN server. As there is

no other option a polling mechanism had to be implemented. Every second the game asks the chat server if there is a new message for any of its current players. If there is a message it will be returned as a string to the game and is displayed on the players screen. The 'asking' and 'return' are implemented as method invocations of the functions defined in the IDL listing above.

With the graphical desktop client a user can also connect to the chat server and start talking to players who are playing the game. The communication with the chat server is implemented with exactly the same code as the NWNX extension described above.

The system as described above performs well in a practical situation. Playing the game with over 15 people and 2 servers resulted in a good performance. In case of a simulated chat server failure the games and servers kept working without problems. Only the messages were of course not transferred anymore.

3.2 A generic approach

We now have a chatting channel between multiple servers. It would be nice if we could extend this channel and accept other kinds of communication on it making it a generic channel.

There are several ways to do this, but after some research we figured out that using the combination of XML and CORBA would result in a perfect way to make a generic and extensible communication channel. XML will be used as the representation of the data. To use XML in CORBA a programmer must represent the XML structure in an IDL definition file. This can also be done in several ways as explained in *Doug Schmidt* and *Steve Vinoski* article [5]. In that same article they state that this method still somewhat cumbersome and error-prone is.

The OMG implemented adopted a new specification in the IDL language which simplifies the process of representing the XML tree: XMLDOM: DOM/Value mapping [6]. Unfortunately this new specification is still not much adopted by ORB producers.

4. Optimizations

Distributed Real-Time Environment (DRE) systems possess stringent quality of service (QoS) constraints, such as bandwidth, latency, jitter, and dependability requirements.
The Real-time CORBA specification [7] was standardized by the OMG to support the QoS needs of certain classes of (Distributed Real-Time Environment) DRE systems, *i.e.*, those that rely on fixed-priority scheduling. Specifically, Real-time CORBA provides standard features that allow DRE applications to configure and control the following system resources:
– Processor resources via thread pools, priority mechanisms, intra-process mutexes, and a global scheduling service for real-time applications with fixed priorities,
– Communication resources via protocol properties and explicit bindings to server objects using priority bands and private connections, and
– Memory resources via buffering requests in queues and bounding the size of thread pools.

4.1 General optimizations

A thorough explanation of this topic falls outside the scope of this paper, so only one important matter will be listed here.

In the previous chapter you saw that the messages are stored in a linear list. Fetching a message can thus in a worse case take O(n) time where n is the number of messages currently in the list. As the messages are looked up every second this list won't grow very big.

The CORBA implementation takes care of the reliability of the data, but in the event of a system failure all messages will be lost. So usage of an external database system will make the server more reliable. Besides reliability using a database will also speed up the system as it has a faster lookup method implemented.

4.2 Using real-time CORBA

The goals of Real-time CORBA specification is to support developers in building predictable distributed system, a prerequisite for ensuring real-time performance, by providing mechanisms to control the use of processor, memory, and

network resources. The application manages the resources by using the Real-time CORBA interfaces and the ORB's mechanisms to coordinate the activities that comprise the application.

Systems required to react to an event or complete task in specific time are known as Real-Time systems. Real-Time can further be divided into two categories:

- Hard Real-Time
  The classic definition is that hard real-time applications fail if the system timing requirement is not met. Good examples are digital fly-by-wire system of aircraft and missile self cruise system.
- Soft Real-Time
  The Soft real-time applications can tolerate some degree of latency in what they are required by the system. The time constrains violation is not fatal, but the system performance may degrade. Examples are vendor machine, printer controller and of course games.

The communication time delay of remote method invocation has to be considered in real-time constrains. The underlying choice of transport mechanism (protocols and networking hardware) has a significant impact on overall system performance.

Major implementations of high performance systems are implemented using TAO middleware; a high-performance, real-time ORB endsystem targeted for applications with deterministic and statistical QoS requirements, as well as "best-effort" requirements. The TAO ORB endsystem contains the network interface, OS, communication protocol, and CORBA-compliant middleware components and features.

As we are working with a real-time game environment we fall in the soft real-time category. Latency is an important aspect for a game. For example when a player fires a missile a player wants to see it coming. If there is much lag (high latency) then the game will react on this by calculating a possible location of the missile based on it trajectory and the amount of lag. When latency becomes a bottleneck in the communication channel TAO could possibly be a solution (there was no time for an actual implementation).

Conclusions

In real-time game environments speed and reliability are both important aspects. As described in a previous chapter the chat system we implemented performed above expectations. In this case it was not a matter of micro seconds, so CORBA should be fast enough for these kinds of operations.

We think that it is possible to make a generic communication channel with the techniques described above: using XML as representation of the data which should be transferred with CORBA. This system should perform fast enough keeping in mind that we are looking at milliseconds (current computers and internet connections are fast enough). Using the optimizations described in part 4, like using TAO, good even speed things up more.

Of course there is also a matter of security. Unfortunately we haven't had time to do research after how secure CORBA really is. Can it for example use encryption mechanisms like SSL during network communications?

References

[1] Blackdagger Corp, http://www.avlis.org.

[2] OMG, Object Management Group, http://www.omg.org/.

[3] omniORB, http://omniorb.sourceforge.net

[4] NWNX cvs, Jeroen Broekhuizen, http://nwnx.homedns.org/cgi-bin/cvsweb.cgi

[5] D.C. Schmidt and S. Vinoski. "CORBA and XML, Part 2: XML as CORBA Data", C/C++ Users Journal C++ Experts Forum, July 2001, http://www.cuj.com/documents/s=7993/cujcexp1907vinoski/.

[6] Bea Systems, e.a., "XMLDOM: DOM/Value Mapping Revised Submission", OMG document, August 2000, http://www.info.fundp.ac.be/~ven/CIS/OMG/XMLDOM. DOM - Value Mapping.pdf

[7] Object Management Group, *Real-time CORBA Specification*, OMG Document formal/02-08-02 ed., Aug. 2002.

[8] Techniques for Enhancing Real-time CORBA Quality of Service, IEEE Proceedings, May 2003, http://www.cs.wustl.edu/~schmidt/PDF/IEEE-proc.pdf