Medical Informatics Europe '97
C. Pappas et al. (Eds.)
IOS Press, 1997

A New Object Request Broker Architecture in Implementing a Medical Picture Retrieval System

Nicolas Silberzahn[1], Franck Lefevre[2], Pierre Le Beux[1]

1. *Laboratoire d'Informatique Médicale, Faculté de Médecine, 35043 Rennes, France.*
2. *K1 Informatique, 15 rue du Point du Jour, 61200 Argentan, France.*

Abstract. This paper describes the approach we used to implement the world wide web user interface of MediPict, a medical picture retrieval system. The best software architecture seemed to be the three-tier client/server architecture using Objects Request Brokers (ORB) which appears to have many benefits. Although CORBA appears to be the standard ORB, we found it more adequate to develop a different approach. This new ORB resolves many of CORBA's weaknesses (non coherent types, need of a static IDL, maintenance difficulties) by using the same language on the top and middleware tiers.

1. Introduction

Information systems have been using the two-tier client/server model for a long time. The best software architecture now seems to be the three-tier client/server architecture using Objects Request Brokers (ORB). In 1992, the Object Management Group (OMG), a consortium formed in 1989 to define the standards required for distributed object systems in heterogeneous environments, defined the services to be provided by an ORB and approved a standard architecture called the Common Object Request Broker Architecture (CORBA™) [1].

In developing a medical picture retrieval system, MediPict, we were lead to develop a new and more convenient ORB. This paper describes the MediPict ORB, how it is implemented and its differences with CORBA.

This new technology may have a major impact on medical and hospital information systems.

2. Material and methods

2.1 The three-tier architecture

In the two-tier client/server model, the client implements the user presentation and the application logic, the servers stores the data.

In the three-tier architecture, the application logic is separated from the user presentation in a new middle tier called the middleware. This architecture promises to provide applications that are easier to build, manage and evolve than with the original two-tier: clients are simpler and smaller, application logic can be reused in different applications, different clients can easily be made for the same application for specialized users...

The Internet World Wide Web [2], using the HTTP protocol [3], becomes the *de facto* standard for user presentation and information distribution. Shifting from a static

publishing medium to an application medium, it appears to be the way the user presentation tier (the top tier) should be implemented.

The last tier is still the database which stores the data used by the application logic.

The middle tier is implemented as remote code accessible through the network that can be shared across many applications. ORB seems to become the way the middle tier will be implemented. An ORB allows applications to communicate with another by supporting message calls between objects through a network, no matter where they are located (on the same machine or across the network): using an ORB, a client can transparently invoke a method on a server object. Objects appear to be local to their clients which do not know what machine an object resides on.

2.2 Actual middle tier implementations

To date, there are many ORB implementations [4], most of them (if not all) claim to be CORBA-compliant. In fact, CORBA appears to be the standard ORB. It works as follows:

On the client side, remote objects are seen as proxies (local representation of a remote object) and used as they were local objects. When called, the proxy delegates the call to the remote object, and gets the result.

To be able to delegate the calls and build the proxies, CORBA needs a description of the remote interface. Such description should be provided by the programmer through a description file using the Interface Description Language (IDL). This file needs to be compiled to generate client and server code.

[5] showed how to call CORBA objects from the http server using a preprocessor to imbed calling results in html templates pages. The preprocessor code is written using the TCL language and use a particular package TclDii to map TCL calls and data models to the CORBA.

[6] showed how World Wide Web integration can be made on the client side using Java™ on a Java compatible browser: Java code running on the client can call remote CORBA objects.

To build a widely accepted standard, the Object Management Group, whose membership includes over 500 hardware and software vendors, had to provide one that promotes independence in hardware architectures, languages and operating systems. By complying to CORBA, software objects can be written in any computer language, can run on any machine, and in any operating system. However, the vendor's benefit is not necessarily the same as the programmer's; thus the price to pay for the any-any-any seems to be too high.

The main limitation of the CORBA architecture comes from the need for an IDL. All possible calls have to be known at build time since the same IDL is compiled at both client and server sides. Thus:

- CORBA requires a neutral type representation. A client script written in TCL calling a CORBA service written in C^{++} first needs a TCL to IDL type mapping on the client side and then an IDL to C^{++} type mapping on the server side. A Java code running on a browser also needs a JAVA to IDL type mapping. Of course mapping is possible by lowering both system types.
- The middleware cannot be dynamically programmed nor modified from an administrator client.

- The middleware is harder to program and maintain since any change should be done on both side: the server and any client should be modified. The middleware also takes a long time to program because the modification needs an edit-compile-update-test cycle.

3. Results

3.1 MediPict ORB's Description

Designing the MediPict software architecture, we found it more convenient and efficient to develop a new object request broker.

MediPict's ORB uses a new language that we called iScript. The iScript language is a full featured interpreted language, close to a lisp interpreter [7], that we wrote in C++. This language runs on both top and middle tier sides and contains the built-in capability to call itself through the network. We added distributed capabilities to allow a running script to call remote functions running on a different machine.

The following example shows an example of iScript code running on the user presentation tier. It defines two proxies for two remote functions `TestAccount` and `DbSearch` whose code is on two different distant computers.

```
(define PictureSearch (account key)
  ;Find the Object Brokers on the net
  (setq AccountBroker (FindObjectBroker "194.164.35.179" "Account"))
  (setq PictDb (FindObjectBroker "194.164.32.137" "MedicalImagingDb"))
  ;Import API & define proxies
  (DefineRemote AccountBroker 'TestAccount)
  (DefineRemote PictDb 'DbSearch)
  ; Use them as local calls
  (when (TestAccount account)
        (ShowResult (DbSearch key))
        )
  )
```

Of course, in a real application, it is better to define proxies once at the beginning of the program. When called, this code stops and waits for the proxies `TestAccount` and `DbSearch` to return. The proxies delegate the calls to the remote procedures, wait and return back the result of the remote calls. It would also be possible to do asynchronous calls, but we did not need them for the MediPict application.

Thus, the middleware implementing the application logic objects is completely written using iScript, as is the user presentation tier. Communication between the two is handled transparently by the language itself.

3.2 The low level communication layer

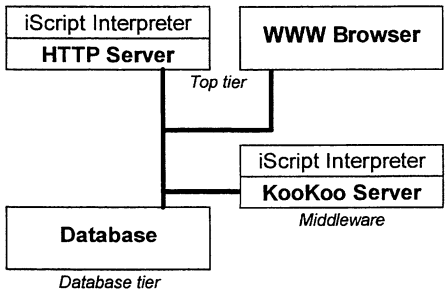
The low level TCP/IP communication layer is built as a non specific tool (known as the KooKoo server) running as an independent server that allows any remote dynamic library to be called through the network. The iScript language is implemented as a Dynamic Linked Library (dll) and resides on the server's computer. The server receives incoming requests and calls the iScript dll in concurrent threads.

The iScript language also runs on the top tier. A call to a remote function is made through a proxy. This proxy calls the KooKoo server that calls the iScript function. The call includes the name of the iScript function to be called, the dll name and the arguments description. There is always one binary bloc as argument since all iScript function arguments are gathered in an unique binary bloc. In fact, every iScript object has an equivalent binary bloc.

As an example, suppose a call to a remote iScript function with a picture, a string and two numbers as arguments. Every binary bloc equivalent of the arguments are concatened using a special grammar. This bloc is given to the KooKoo server which calls the iScript Function. This function does the opposite: a parser reads back the picture, the string and the two numbers from the bloc. The function executes and gathers its result again in one bloc which is sent back to the client. On the client side, the bloc is parsed to give the result to the initial calling iScript function.

Even the ORB object, the iScript Interpreter Object, any function can be managed by iScript, thus converted to a bloc, thus be used as arguments of a call. This way is it possible to send a complete iScript interpreter (including symbol table, functions...) to another iScript interpreter on a remote computer.

3.3 The Middleware call



The MediPict architecture

Since both belong to the same tier in the three-tier model, we consider the top tier client side to be both the Web Browser and the HTTP server. Remote objects are called from the HTTP server, using the CGI. A C^{++} written CGI scans HTML templates and extracts embedded iScript scripts, executes them and replaces them with their execution result in a new HTML page that is sent back to the client browser as with WebStar [Almasi 96].

```
<HTML>
<HEAD><TITLE>HTML document for the World Wide Web</TITLE></HEAD>
<BODY>
   <H1> This is today's date [date "DDMMYYYY"]</H1>
   2 + (3 x 5) make [+ 2 (* 3 5)]
</BODY>
</HTML>
```

When sent to the browser, the HTML page is:

```
<HTML>
<HEAD><TITLE>HTML document for the World Wide Web</TITLE></HEAD>
<BODY>
   <H1> This is today's date 12/12/1996</H1>
   2 + (3 x 5) make 17
</BODY>
</HTML>
```

4. Discussion

In the MediPict's Object Broker:

- The same data model is used on both middleware side and client sides. Since the same language is used on both sides, there is no need to map two different type systems. Language and data model are coherent and homogeneous.

- The middleware can be dynamically modified (or even built from scratch) from any client that is allowed to, since iScript contains functions not only to call but also to modify the Object Broker. To take an elementary example, here is how a "Double" remote procedure can be created:

```
(progn
   (setq myOrb (NewObjectBroker "194.164.35.179" "DoubleInterpreter"))
   ;Define the Double Function on the remote computer
   (Define myOrb "(de Double (x) (* x 2))")
   ;Define Double's proxy
   (DefineRemote myOrb 'Double)
   ;Call the Double function which will run on the remote computer
   (print (Double 32))
   )
```

- There is no need to maintain an IDL on both side: the IDL information comes directly on the network with the call itself.

5. Conclusion

The three-tier model using an Object Request Broker (ORB) as middleware, seems currently to be the best architecture in managing information systems. In spite of certain limitations, Corba or Corba compliant ORB are the *de facto* standard ORB. In this paper, we have described a new ORB that we use in developing MediPict, a medical picture retrieval system. Unlike Corba, MediPict ORB, uses the same language (that we called iScript) and the same data model on both the middleware and client sides. Therefore, language and data model are coherent and homogeneous. Whereas in Corba all possible calls have to be known at build time, iScript contains functions not only to call but also to modify the ORB. Thus the middleware can be dynamically programmed or modified from an administrator client, according to his specific needs or data base evolution.

With the MediPict ORB, the three-tier architecture becomes easier to evolve, more coherent as well as being dynamically modifiable. It thus appears to be a well adapted tool for developing our medical picture retrieval system.

6. References

[1] The Common Object Request Broker: Architecture and specification (CORBA) Revision 1.2. OMG TC Doc. 93.12.43, Object Management Group, Framingham Mass., Dec 1993.

[2] Berners-Lee T., Cailliau R., Luotonen A., Frystyk Nielsen H. and Secret A. The World Wide Web. Communications of the ACM 37 (8) (Aug 94)

[3] Berners-Lee T., Fielding R. T. and Frystyk Nielsen H. Hypertext transfer protocol HTTP 1.0 Internet Draft, IETF HTTP Working Group, Sept 1994

[4] Orfali R., Harkey D. and Edwards J. The Essential Distributed Objects Survival Guide, Wiley 1995

[5] Integrating the WWW and CORBA-Based Environments. First Class OMG Magazine Volume VI, Issue I

[6] CORBA the Greek gets wired on JAVA. First Class OMG Magazine Volume VI, Issue I

[7] McCarty J. Recursive functions of symbolic expressions and their computation by machine. Communications of the ACM 3 (4)