Using CORBA and Java for PBX Management

P. Hasselmeyer
Information Technology Transfer Office
Darmstadt University of Technology
Wilhelminenstr. 7
D-64283 Darmstadt
Germany
peer@ito.tu-darmstadt.de

M. Andrew
Bosch Telecom
Kleyerstr. 94
D-60326 Frankfurt
Germany
mark.andrew@fr.bosch.de

Abstract

Telephone Switches are characteristically long-lived, evolving systems. We describe how a legacy two-tier system for telephone switch management was re-engineered as a three-tier web-based application using CORBA and Java. In this solution the design-time advantages of portability and abstraction brought by an IDL information model are combined with the universality and ease of use of a Java GUI. Performance was shown to be largely unaffected by the addition of the extra tier, while CORBA was seen to bring unexpected new advantages in the area of scripting and RAD.

Keywords

Web-based operation and management, PBX, CORBA, Java, information model, CORBA Scripting.

1. Introduction

Private Branch Exchanges (PBXs) offer a large number of features and require management functions which can evolve over time. The administration tools are usually complicated to use and require specialists to perform common (and not so common) tasks such as creating new user accounts, or restricting calls. With each new line of PBXs new management tools are introduced. Unfortunately, the lifetime of a PBX is rather long and technology evolved at a high pace during the last decades. Old systems may only be equipped with a V.24 interface and have a command line oriented interface. Later systems come with Windows-based software communicating via proprietary protocols over ISDN lines. As each user interface looks different, people performing management tasks have to be trained to use them all. Furthermore, the management console (usually a PC running Windows software) must be equipped with the right hardware interfaces and the corresponding software.

To reduce the amount of training required as well as the amount of software needed, a single, consistent graphical user interface (GUI) is desired. Of course,

this goal cannot always be reached as some telephone exchanges offer features which other switches do not support. However, all PBXs have a common set of basic functionality. This includes for example user accounts. Some switches offer additional attributes for the basic functionality, others offer completely new functions. Depending on the type of difference the GUI has to be changed in an appropriate way.

One way to tackle this harmonization would be to write new management tools that all have the same user interface but communicate with the switch in the appropriate way. A lot of code could probably be reused but this would still require a large number of different tools – one program per PBX type. To add to this, there are often different software releases for one type of PBX. Each version may need a different administration tool because it offers different functionality or it employs a different communication protocol. Even if this problem could be handled, what about other administration tools which are unconnected with GUI concerns, such as for example batch processing? These tools would still be different for various PBXs. An integration of all models should therefore be addressed at a lower level. One method is presented in this paper. As shown in Figure 1, it employs a three-tiered architecture; Java on the client side, CORBA for the communication between the client and the server, and servers for mapping CORBA-calls to the appropriate functions of the different lines of telephone exchanges.

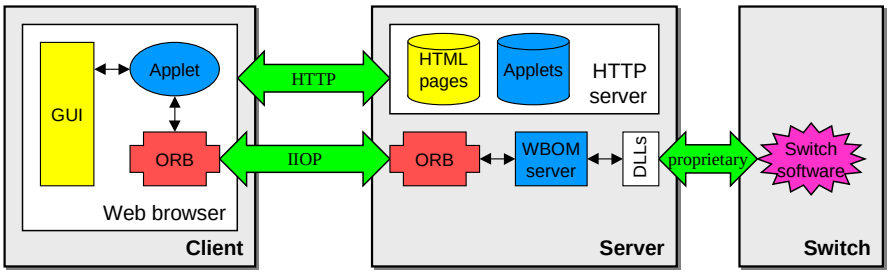

Figure 1: PBX management architecture

This paper is the result of practical experience gained in a two man-year research project called WBOM (Web-Based Operation and Management). In this project Bosch Telecom and the Darmstadt University of Technology developed a prototype of a Management system for one of Bosch's models of telephone switch. This system consists of two parts – the switch itself on the one hand and on the other hand a server PC which handles all the management operations. The PC is equipped with a set of runtime libraries, which handle communication with the switch via a proprietary protocol. The project involved the realization of a prototype, which handles only a small part of the management. This part is the so-called "Login User". This is basically a telephone account containing as most important attributes a name and a number. As added functionality a login user can log on at every phone connected to the PBX. This means that all incoming phone

calls are now routed to the telephone he logged on at. This functionality is needed when offices are assigned dynamically.

Two major problems of administration are solved by this architecture in an elegant way: the demand for a thin client and the versioning problem. A web browser is not really a small piece of software, but it does not require a complicated installation procedure. In fact, most new systems today are already equipped with such a browser. As another plus, browsers exist on most platforms and the majority of operating systems. Browsers are even becoming available for PDAs and hand-held computers. Unfortunately, the missing Java Virtual Machines prevent such small systems from being used as clients at present. No doubt this problem will vanish within the next year.

The second major problem solved by this architecture is the versioning problem. Many different software versions may exist within a PBX family. This is due both to the phased introduction of new functionality and localization for different countries. Each version might require a slightly different management tool. It is not always evident which is the correct tool for a given switch. The proposed architecture addresses this problem in a natural way. As the applet is downloaded "just-in-time" from the server and the server belongs to the switch, it is always the correct version for the administered type of exchange. This is similar to web-based management agents [1].

When designing a new management architecture, a number of issues have to be addressed. Among the most important are:

- the information model,
- the language and interface toolkit used on the client side,
- the server architecture.

The next sections cover all the aspects mentioned before. An information model is introduced, the implementations of the client and the server are described, and an evaluation of the prototype is presented.

2. The Information Model

For the existing proprietary interface between the management PC and the switch no explicit information model exists. Data is accessed via a large number of different functions. Due to their origins in the requirement phase of a Jacobson-inspired software development process these functions are known as "use cases" [2]. Each use-case represents a small management scenario, e.g. the change of an attribute of a login user.

The prototype was designed to operate with a standardized protocol. It was agreed to use the Common Object Request Broker Architecture (CORBA) [3]. This has two advantages. One is the use of the Interface Description Language (IDL). With this language it is possible to describe an information model in an abstract, machine-readable, object-oriented, implementation-independent way. The second advantage is the availability of a suite of standard communication protocols, most notably the Internet Inter-ORB Protocol (IIOP). This allows different applications

to communicate in a standardized way with other resources regardless of the Object Request Broker (ORB) manufacturer which the individual resources use as a communication framework.

The choice of using CORBA as the communication infrastructure appears to be backed by the network management community as there are a number of efforts using this technology together with SNMP [4] and OSI management [5]. Besides other advantages [6, 7, 8], the ability to cross the programming language boundary (here C++ and Java), the ease of use and the availability of development tools were the main factors that let us choose CORBA.

After having decided upon a technology to use for communication, the existing data had to be described. First, all available data had to be found and structured in a consistent and flexible way. It was decided to adhere to one basic principle: to use object-orientation as much as possible. The resulting information model is shown in Figure 2.

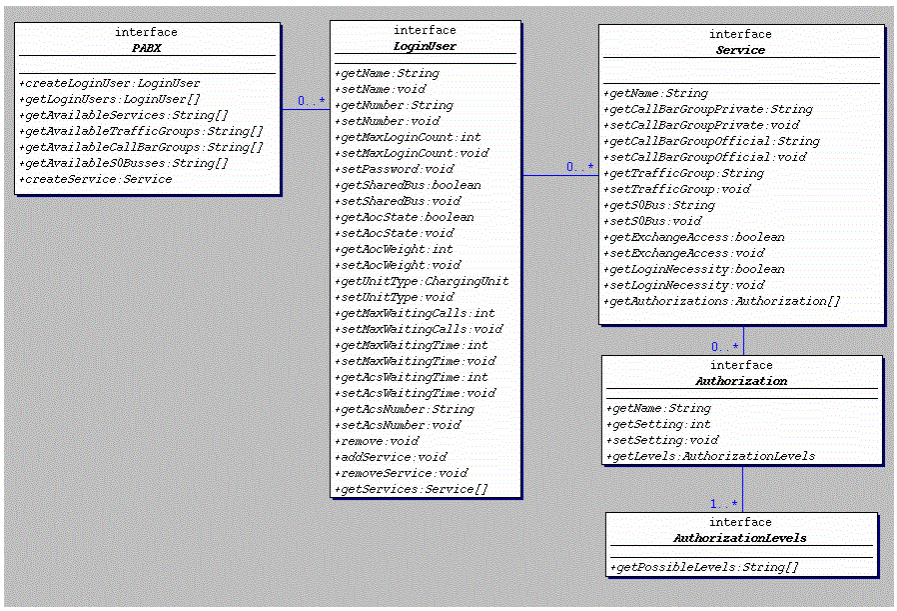

Figure 2: Information model

It was a natural choice to model login users as objects. Each login user contains a set of attributes, most notably name and number. Attributes are not modeled as CORBA attributes in our information model because according to the current IDL syntax (CORBA 2.3 [3]), the access methods generated from IDL attributes do not raise user-defined exceptions. However, this functionality is needed here as the communication link to the switch might have broken down or the changing of a login user's name failed because the name is already in use. The solution was therefore to model each attribute with two functions (which have the

same names as the corresponding automatically generated functions for that attribute would have had) which are allowed to generate the required exceptions.

A further feature of IDL which we decided not to use was the *enum*. We concluded that the possible alternative values for a type, usually represented by enums, were better modeled as a list of strings returned at run time. This would decouple the two sides, and usually allow the same client to be used although new possible values were implemented on the switch (see Figure 3 below for an example).

Login users are allowed to use a set of services, like voice or fax. As services can be dynamically assigned to login users and as they contain a number of attributes, they are modeled as objects. The login user's attribute containing the enabled services is thus a sequence of service objects.

```
typedef sequence<string> seqString;

interface PBX {
      ...
      seqString getAvailableServices();
      ...
}
```

instead of

```
interface PBX {
      ...
      enum AvailableServices {
            Fax, Voice, Video, ... }
      ...
}
```

Figure 3: IDL: enum versus dynamic string table

Each service contains a given number of authorizations, which are modeled as objects as well. Authorizations allow or restrict access to features like three party calls, call diversion, and calling party number display. As each authorization can have a different number of levels and a different meaning for each level, an additional object – called AuthorizationLevel – is attached to each Authorization object. AuthorizationLevel objects are simple containers for the human readable identification strings of all available levels.

To get access to all login users as well as to lists of other objects like the names of available S_0 busses, a root object was introduced, which is called PBX. This object is the initial reference for every client to access all further data.

3. The Server

As shown in Figure 1, the server is based on two pieces of software. To communicate with the telephone switch, the server uses functions which are readily available as a set of so-called "Dynamic Link Libraries" (DLLs). These libraries are bound to the program code at run-time. The libraries hide the complexities of

establishing a connection to the switch, encoding the data, and decoding the response. Although written in C++, the libraries expose a functional interface to their users. The interface resembles the data format on the wire to the switch. For each function invocation, a packet has to be created and filled with the instruction identification and a varying number of arguments. Another function then sends the packet to the switch and receives its response. The response contains a result code and the requested data or, in the case of an error, some additional information about the source of the error.

The other communication interface of the server uses CORBA. Here, too, the communication functions are supplied by the infrastructure of the ORB used. The object model described in the previous section was translated to C++ by using an appropriate compiler. Each interface defined in IDL is mapped to a C++-object containing all described functions and attributes. Of course, the functions do not contain any code as IDL only describes the appearance of the objects but not the associated semantics. This is what has to be done when writing the server: all functions described in IDL have to be filled with code, which realizes the intended functionality. In the case of the WBOM-server, this basically means translating the object-oriented function calls to appropriate function invocations on the PBX-side.

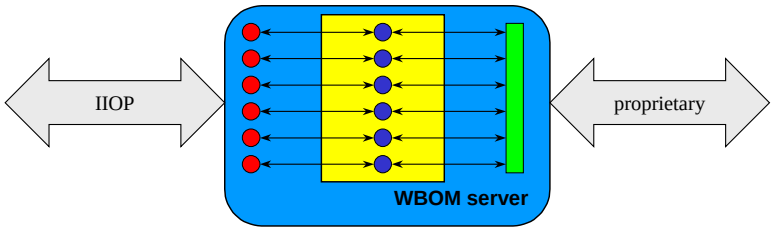

Figure 4: Internal realization of the server

As Figure 4 shows, the translation of data from one side to the other is not done directly, but there is another layer of abstraction in between. The objects in the middle are used to encapsulate the peculiarities of the functional interface on the switch-side. One of these encapsulator objects is attached to each CORBA-object. Because these encapsulator objects handle communication, all that is left for the CORBA-objects is to implement behavior that is on a more abstract level. An example should demonstrate this. On the switch side, a new login user can only be created if it has a name and a telephone number. It was decided not to reflect such restrictions at the object model level as this would involve a tighter coupling with the business logic of a specific switch implementation. Therefore, on the CORBA-side, a login user can be created without these attributes having been set. CORBA-objects have this switch specific knowledge and do not forward the request to create a new login user to the switch until they have been assigned a name and a number. The actual request to add a new login user on the switch is handled by the encapsulator objects as they know which message has to be sent.

The WBOM-server was implemented in C++. This is due to the existing communication libraries being written in C++ and being present in native code. Wrapping the libraries with a layer of Java code would be possible but deemed to be too time-consuming for a prototype. Here, CORBA revealed one of its strengths – the language independence.

4. The Client

One of the goals of the new PBX administration framework was to use up-to-date standard technology. It was therefore decided to let the user interface run in off-the-shelf web browsers. The technologies employed are HTML and Java. HTML is only used to invoke the applet and supply some parameters to bootstrap it. The design of the graphical user interface (GUI) closely resembles the appearance of the pre-existing administration tool. The user interface is based on standard Abstract Window Toolkit (AWT) classes. An alternative would have been to use the higher level Swing widget set from Sun's newly released Java Foundation classes. However, these were only available in a beta version at the time of development and were subject to continual change.

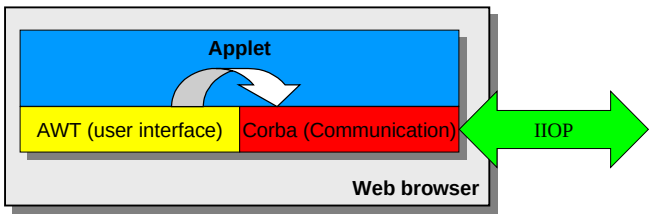

Figure 5: High-level applet design

The basic layout of the applet is shown in Figure 5. It is based on AWT for the user interface and on CORBA for the communication with the server. Besides displaying the user interface, the main functionality of the applet is the transfer of data from the GUI to the CORBA-objects and vice-versa. Two examples of the applet displaying the details of a login user are shown in Figure 6.

The classes of the applet are structured according to the design of the user interface. There are four different screens showing different attributes of a login user or a service. Each screen is represented and handled by one class. The left snapshot in Figure 6 shows the screen for displaying a login user's main attributes. To switch between each of the four screens, the user can click on the radio buttons at the top of the screen. These four "tabs" as well as the three buttons "OK", "Cancel" ("Abbrechen"), and "Apply" ("Übernehmen") are handled by a container object, which creates the window, manages the four screens, and distributes events to them. There can be at most one editing window per login user but any number of login users can be viewed and edited at the same time. All editing windows are managed by yet another object, which exists only once per applet. It performs the

startup of the applet, establishes a connection to the server, and displays the list of available login users.

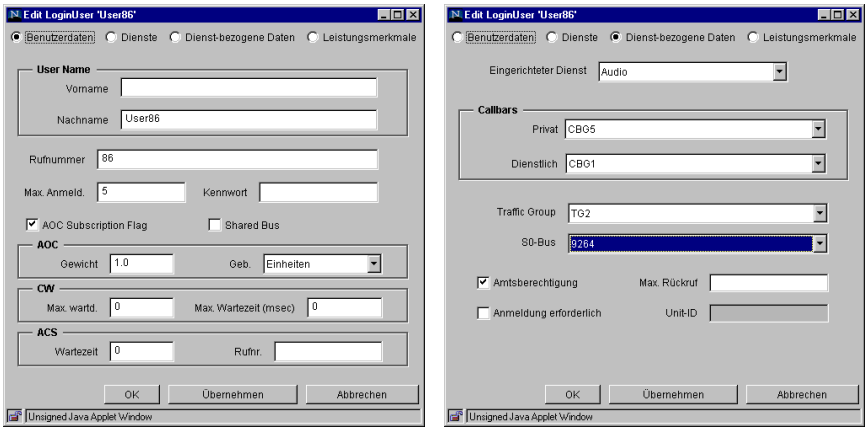

Figure 6: Two snapshots of the client's user interface

To be able to communicate with the server, the client needs to have a reference to the root object. This reference must include the IP address of the server, the port the Object Request Broker (ORB) is listening on, and an identification of the desired object. All this information is included in a so-called Interoperable Object Reference (IOR) which can be represented as a string in a standardized way. The IOR of the PBX root object is transferred inside the HTML page starting the applet. It is supplied as one of the applet's parameters. As soon as the server has finished its initialization, it creates a new HTML page with a predefined name, which contains the invocation of the applet including the IOR of the created root object.

5. Performance

Although administration is an interactive task and performance is not of paramount importance, it is interesting to see how a distributed solution performs in comparison to a more "traditional" approach. Objective time measurements have not been performed, as this is an interactive tool where only the user's subjective sense of a good response time is important. Furthermore, time measurements depend on a large number of circumstances such as network congestion, which are sometimes hard to reproduce.

The performance of the presented solution can be assessed in terms of a number of different reaction times and delays. As the software uses a distributed approach, a lot of communication between the different parts is going on. Therefore, the reaction times are influenced by the speed of the communication. Most of the communication cycles involve the transfer of only a small amount of data, so that the actual data transfer time is rather short compared to the time

needed to prepare the data and push it through a number of software layers. All experiments have been conducted with a 10Mbit/s Ethernet connection between the client and the server and a 64kbit/s ISDN connection between the server and the switch. The different times that can be assessed are:

- startup time of the server,
- startup time of the applet,
- response time to user actions,
- communication time between the client and the server, and
- communication time between the server and the PBX.

Startup of the server includes creation of the Object Request Broker (ORB) and logging into the PBX. In addition, a table of all available ISDN S_0 busses has to be built due to the way the switch assigns login users to these busses. Especially the latter action can take a few seconds, depending on the number of installed busses.

The startup of the applet includes transferring the HTML page and the applet to the client. Furthermore, the browser might have to start the Java Virtual Machine. All this takes around 10 to 20 seconds. Currently, all applet classes are transferred individually. On one hand, this reduces start-up time as classes are loaded when they are needed and initially, only a few are required. On the other hand, each class needs a separate communication cycle. Putting all classes together in one Java Archive (JAR) could speed up the loading time. At the same time, using a signed JAR file, integrity of the classes could be guaranteed and the origin of the code verified. A further issue is the client-side ORB. This could be either the built-in VisiBroker client where the Netscape browser is used, the standard Java client-side ORB where a JDK1.2 plugin has been installed or a third-party product such as ORBacus. In the first two cases, download time can be correspondingly shortened.

After starting the applet, the list of login users is requested from the server. Depending on the number of login users, this can take a few seconds, because each login user needs a new proxy object on the client side and needs to be entered into the list of available login users. For each login user two CORBA-requests have to be executed – one to read the name and one to get the number of the user. This sets a limit on the scalability of the solution in an environment where there are many thousands of users.

The response time to user requests is usually short. A few circumstances can slow down the reaction of the applet, though. When the attributes of a user are displayed for the first time, a few more classes have to be read from the web server. The complete set of a user's data is transferred from the switch to the server as soon as any of its data (other than name and number) is accessed for the first time. All further read accesses by the client operate on the cached copy of the user. This server-side caching reduces the communication time for reading a login user's data dramatically. As a disadvantage, the cached data could be invalid if another administration tool changes some data of a cached login user. It is currently not anticipated to allow multiple administrative tools to operate on the same data at the

same time. If this were ever desired in the future, the PBX would need to send notifications to all involved tools when any data is changed. The server would then forward the notification to the applet.

Changes to a login user's data are not transferred from the client to the server until the button "OK" or "Apply" is clicked. As a result, changing the value of an attribute does not cause any delay as no communication is involved. As soon as the changes are to be committed, all of a login user's data is sent to the server. The client does currently not keep track of the changed attributes. Although this results in a large number of CORBA-requests, the time needed for this part is rather short. As the server receives the request to change an attribute's value, it compares the new value to the currently cached value. If these are the same, the change request is not forwarded to the switch. If the values differ, the new value is sent to the PBX, which requires a much larger amount of time than the CORBA communication request. As each changed attribute's value is transferred to the switch individually, the time to commit all changes is linearly dependent on the number of changes. This amount of time could be reduced if all changes were accumulated on the server and sent to the switch as one batch. Unfortunately, the server does not know which change is the last one in a sequence. To solve this, some kind of transaction context could be introduced. The client has to tell the server about the start and the end of the changes it is sending. The server is then able to accumulate all the necessary changes until it receives the end-signal. Of course, errors detected by the switch can only be sent to the client when it tells the server to commit all previous changes.

From the previous sections one can deduce a general observation: the processing and communication time of the switch is the bottleneck. CORBA communication is performed quickly, but as soon as some data has to be transferred to or from the switch, the performance drops dramatically.

As part of the evaluation of the new tool, its performance was compared to that of an existing administration program. As both tools acquire their data at completely different occasions, a comparison can be in favor of each depending on the evaluated scenario. The fairest comparison is probably the time for transferring all changes to the PBX. As the preexisting tool sends all of a login user's data in one go no matter which attributes changed, it is slower when there are only minimal or no changes. But the transfer time stays constant, even when a large number of changes have to be committed. The implementation presented in this paper becomes slower with the number of changes. So, for three or more changes the existing, "traditional" tool is faster. Of course, the proposed encapsulation in a transaction environment could level this difference.

6. CORBA Support for RAD and Scripting

While C++ and Java both have their roles to play, there are other languages which are easier to use and better suited to the prototyping phases of a project. One of these is Python [9], an interpreted Rapid Application Development (RAD) language whose simple but powerful language model generally leads to a higher

level of productivity where performance and embedded functionality are not of primary importance. With the help of the Python ORB Fnorb [10], Python code can be plugged into existing distributed applications for simulation purposes.

In our case we used it to simulate the WBOM-server and the underlying PBX. A dummy implementation running on a workstation could be written in an hour, this provided an environment to test the client without having to worry about the availability of a real PBX.

A striking feature emerged from the combination of CORBA with a loosely typed language. The use of IDL to specify the interfaces added a previously unavailable level of type-safety to the components so specified. Similar to Smalltalk, Python methods accept and return objects whose type is only known at run time. Types are not specified statically and mismatches are first detected when a method for a particular object cannot be found at run time. However, in this case, the binding to the IDL interface allowed us to specify return and parameter types, conformance to which was then systematically checked at run time every time the component boundary was crossed. In this way a balance was struck between ease of use and security, and we had a high degree of confidence that our simulation was appropriate.

A system controlled by an interactive client – such as our Java GUI – will always have two limitations. Firstly it is difficult to automate testing of the system as a whole via the client. Secondly batch processing (e.g. dealing with large amounts of PBX data in some programmatic way) is not possible. Embedded scripting languages, in particular TCL [11], are frequently used to provide these features for monolithic applications. Reacting to this trend, the OMG issued a Request for Proposal (RFP) for a standardised CORBA scripting language. Nearly all major scripting languages were proposed: after all, all they needed was an IDL language mapping and a supporting run time library to make them CORBA-enabled. Amongst the submissions however was a newcomer called CorbaScript [12]. This language showed that CORBA opens up new possibilities for distributed applications in this area.

As shown in Figure 7 CorbaScript exploits the reflection capabilities provided by the CORBA Interface Repository and constructs remote calls using the Dynamic Invocation Interface. For the user this means that, once the IDL file for the interfaces in question have been loaded into the repository, and an initial object reference (in our case the PBX) has been imported into the scripting environment (maybe by means of a naming service, or via a known file or URL) he can make remote calls directly using the names provided by the IDL interface without having to worry about compiling the IDL, starting the ORB and all the other technical details.

In this interactive environment the user can explore and test his application. The scripting language provides a simple but adequate set of control structures and data types so that scripts can be written for regression tests. Libraries of scripts can then be created which could allow an end user to perform operations which were not foreseen by the GUI.

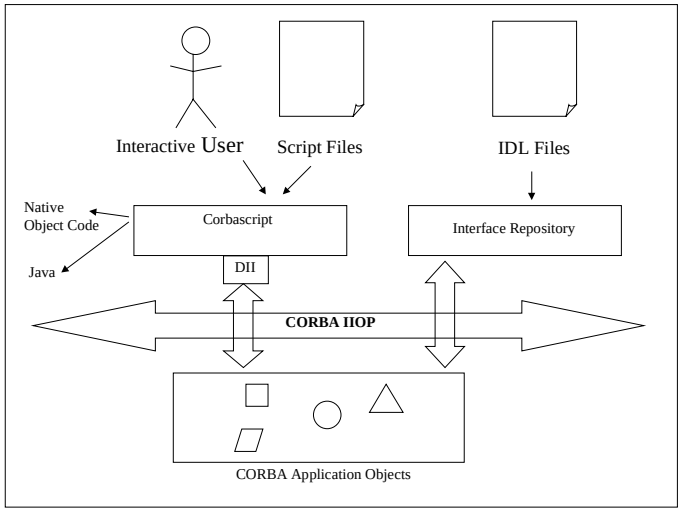

Figure 7: CorbaScript Architecture

7. Conclusion

While we had no reason to doubt the feasibility of our approach to integrating a legacy management capability into a state-of-the-art distributed architecture, the process of implementation revealed initially unsuspected issues. Although Java and CORBA make an excellent match for thin clients, CORBA opens the way for much more. IDL not only provides the glue between distributed components but also offers a level of abstraction for modeling data. However, the information model has to take real-world issues such as performance and scalability into account without sacrificing generality.

Our experiences showed that Java and CORBA enable the creation of a system which offers approximately the same functionality as a less flexible and less portable preexisting application. Furthermore, it automatically provides new capabilities such as scripting, RAD, and design level support. A logical next step would be the extension of the information model to cover all management functions of the PBX where issues such as the correlation of different objects could be addressed.

References

[1] Mullaney, P. Overview of a Web-based Agent. *The Simple Times 4*,3 (July 1996). http://www.simple-times.org/pub/simple-times/issues/4-3.html.

[2] Jacobson, I. *Object Oriented Software Engineering – A Use Case Driven Approach.* Addison-Wesley, Wokingham, England, 1992.

[3] Object Management Group. *The Common Object Request Broker: Architecture and Specification*, Revision 2.3. OMG document 98-12-01. ftp://ftp.omg.org/pub/docs/formal/98-12-01.pdf.gz.
[4] Aschemann, G., Mohr, T., Ruppert, M. Integration of SNMP into a CORBA- and Web-based Management Environment. *Kommunikation in Verteilten Systemen (KiVS)*, Darmstadt, Germany, March 1999.
[5] Canela, Z., Bardout, Y., Voyer, F. Integrating Web-Based User Interfaces in TMN Systems. *Network Operations and Management Symposium (NOMS)*, New Orleans, LA, February 1998.
[6] Haggerty, P., Seetharaman, K. The Benefits of CORBA-Based Network Management. *Communications of the ACM 41*,10 (October 1998), 73-79.
[7] Mazumdar, S. Inter-Domain Management between CORBA and SNMP. *Distributed Systems: Operations & Management (DSOM)*, L'Aquila, Italy, October 1996.
[8] Garbanati, L. Enabling Electronic Communications for Local Competition – Why CORBA? *Network Operations and Management Symposium (NOMS)*, New Orleans, LA, February 1998.
[9] http://www.python.org
[10] http://www.dstc.edu.au/AU/staff/martin-chilvers/Fnorb/
[11] http://www.scriptics.com
[12] http://corbaweb.lifl.fr/CorbaScript/