Cooperation of Dialogic Standard Runtime Library (SRL) and CORBA

Technical Notes

Dominik Radziszowski
Bartosz Klimek
e-mail:{radzisz,bartekk}@student.uci.agh.edu.pl

Institute of Computer Science
University of Mining and Metallurgy
Cracow, Poland

May 1999

The aim of this document is to briefly describe what we did to develop a program using Dialogic boards and CORBA on Windows NT at the same time. We hope it will be useful for anybody who will have a similar task. However, most details given here can be found in appropriate manuals and documentation.

1 System Architecture

The figure below presents an overview of the architecture of our system. This is just an outline. If more detailed information is needed, please consult other documents of ours.

The Telephone Exchange Interface (TEI) is a client process running on a Windows NT server (netj.ics.agh.edu.pl). It communicates with Dialogic boards through the Dialogic Standard Runtime Library (which in fact involves the Dialogic driver as well). TEI reacts to events coming from the boards by calling synchronously the Call Redirection Manager's operation. The Call Redirection Manager (CRM) is a server running on a SunOS machine (iris.ics.agh.edu.pl).

Those components are part of a larger object-oriented distributed system based on CORBA.

2 Interprocess Communication — CORBA

CRM, as well as other servers running on iris, uses Orbix 2.3 — an ORB from IONA Technologies. TEI uses omniORB 2.7.0 — a free ORB from Olivetti. Essentially, they cooperate

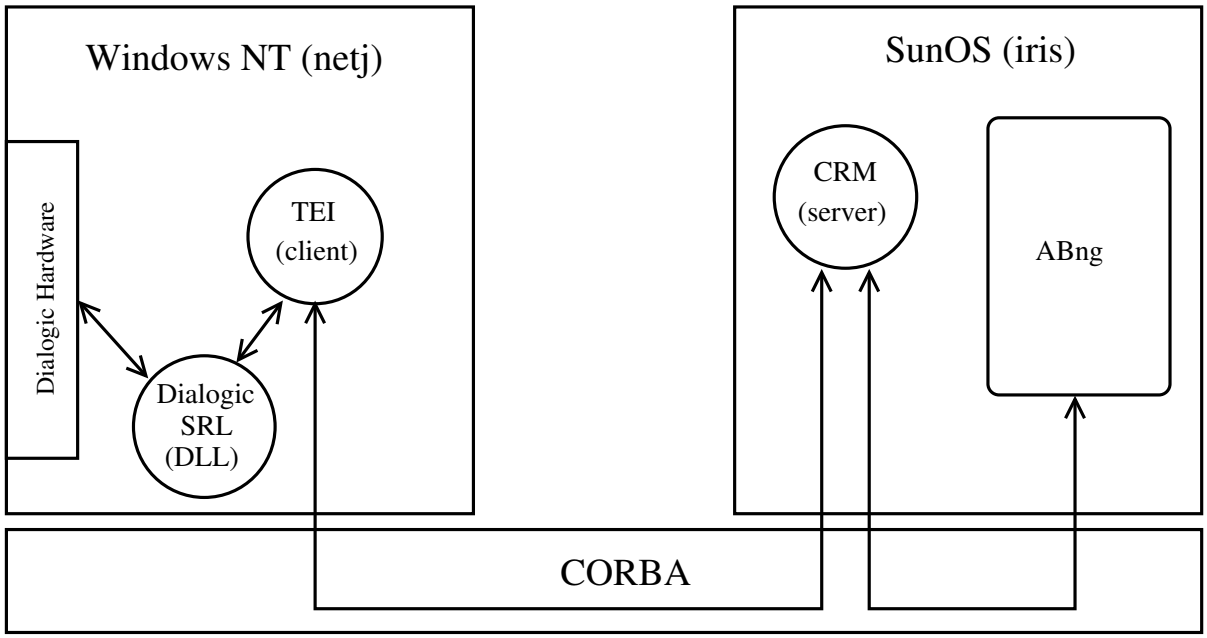

Figure 1: The Architecture of Telephone Exchange Interface

without any problems. The only obstacle we had to face was a bug in Orbix (mended in more recent releases): an attempt of an omniORB process to confirm that a given remote object (sitting inside an Orbix process) implements a given interface always failed, even if the object did implement the interface. This made the *_narrow* method of the C++ interface class always return *_nil*. The solution (by Daniel Mencnarowski, thanks) we accepted is effective but not very ellegant. We modified the code of omniORB so that it just ignores what the remote ORB answers. In method *omniObject::_real_is_a*, file `src/lib/omniORB2/object.cc` found in the omniORB sources directory, we added a line, marked with a comment below:

```
...
switch (_c.ReceiveReply())
{
   case GIOP::NO_EXCEPTION:
   {
     _result <<= _c;
     _c.RequestCompleted();
     _result = 1;   // *** this line has been added ***
     return _result;
     break;
   }
...
}
...
```

We are aware that this trick may be somewhat dangerous in case of faulty programs using the modified omniORB, and may lead to strange behaviour and hard to find errors.

To recompile omniORB we needed WIN32 GNU tools, which can be downloaded from:

ftp://ftp.orl.co.uk/pub/omniORB/gnu-win32-lite.zip

It is noteworthy that an improper version of the tools is likely to cause a lot of trouble, which happened in our case...

When omniORB is built, please get sure that the libraries and dlls in `lib\x86_win32` and bin\x86_win32 have been replaced with their newer versions.

Following the documentation we recommend adding

d:\home\radzisz\omniORB_2.7.0\bin\x86_win32

(or some other appropriate path) to the PATH environment variable so that dlls can be loaded from there.

Another important question we had to answer was where TEI would get the reference of CRM from. We used the Orbix Name Service running on iris. The string representation of its reference (IOR) was just copied through the clipboard from iris to the register of netj. On iris, the IOR of Name Service can be obtained this way:

```
$ cat $NS_IOR_FILE
```

Let us now cite the omniORB documentation (`README.win32`):

"Start the tool REGEDT32.EXE (on NT) or REGEDIT.EXE (on Windows '95). Select the key HKEY_LOCAL_MACHINE\SOFTWARE\ORL\omniORB\2.0 (or create it if it doesn't exist). Add a string value (known as a REG_SZ data type when using REGEDT32) called NAMESERVICE, and paste the IOR in to the value field associated with it."

This operation must be carried out carefully. Our advice is to avoid using Windows telnet for this purpose because it has problems with copying multiple lines of text. It is better to copy $NS_IOR_FILE from iris to netj, open it with notepad, select all the text, copy it to the clipboard and then paste it in the register.

CRM is registered in NS with naming context "CRS"/"CRM", and its reference can be resolved this way (see also `tei.cpp`):

```
CosNaming::Name name;
name.length(2);

name[0].id   = (const char*) "CRS";
name[0].kind = (const char*) "my_context";
name[1].id   = (const char*) "CRM";
name[1].kind = (const char*) "Object";

CORBA::Object_var obj;
try {
  // rootContext is a reference to the root context of NS,
  // resolved earlier
  obj = rootContext->resolve(name);
}
catch(CosNaming::NamingContext::NotFound& ) {
  cerr << "Context not found." << endl;
  exit(1);
}
```

Compiling and linking programs using omniORB with MS Visual C++ compiler requires the following actions:

- go to Project|Project Settings, tab: C++, Category: "Preprocessor", Preprocessor definitions, and add:

  `__WIN32__, __x86__, __NT__, __OSVERSION__=4`

- go to Additional include directories, and add:

  `d:\home\radzisz\programy\omniORB_2.7.0\include`

  or anything appropriate if omniORB is installed somewhere else,

- go to Category: "Code Generation", Use runtime library, and select: "Multithreaded DLL",

- go to tab: Link, Category: "Input", Object/library modules, and add:

  `omniORB270_rt.lib, omnithread2_rt.lib, wsock32.lib`

- go to Additional library path, and add:

  `d:\home\radzisz\programy\omniORB_2.7.0\lib\x86_win32`

  or anything appropriate if omniORB is installed somewhere else.

3 Programming Dialogic Boards

Dialogic boards (D41/ESC and MSI/SC-R in our system) can be controlled through Standard Runtime Library. This is a C library available on Windows NT and various Unices. Even a short discussion of its features is beyond the scope of this document so we will focus on a few issues that seem to be troublesome for a beginner to this subject.

- To compile a program using the boards, these settings in Visual C++ development environment must be done:
  - go to Project|Project Settings, tab: C++, Category: "Preprocessor", Additional include directories, and add:

    `d:\dialogic\inc`

    or anything appropriate if Dialogic software is installed somewhere else,
  - go to Category: "Code Generation", Use runtime library, and select: "Multithreaded DLL",
  - go to tab: Link, Category: "Input", Object/library modules, and add:

    `libdxxmt.lib, libsrlmt.lib, libdtimt.lib, libfaxmt.lib, sctools.lib`

    (of course this is just an example, in other projects the set of necessary libraries may be different),
  - go to Additional library path, and add:

    `d:\dialogic\lib`

    or anything appropriate if Dialogic software is installed somewhere else.

- We noticed that an annoying linker warning about conflicting libraries disappeared when we added "libc.lib" to the "Ignore default libraries" field in the Link tab.

- So far we haven't seen any harmful interference between omniORB and SRL.
- Programmers should be careful when passing *TPT* structures to SRL functions. Setting the *tp_type* field of the structure in a wrong way may cause a system crash followed by a reboot. This is probably a Dialogic driver bug, or an NT bug, or both.
- When the SC bus default routing is changed, please keep in mind that it is not restored when the program quits. This can bring about unexpected results. It is a good rule not to assume anything about routing when the program starts up, and to restore the routing before the program quits (it is not always possible). We haven't yet discovered any Dialogic tool that could restore the default routing.
- We do not know how to detect a hangup when we accept an incoming call from an outbound line. Standard methods, i.e. loop current off detection and PBX tone signalling, do not work. We tried to use PBExpert to find the tones but it worked for dial tone only.

4 Sources of Additional Information

- *http://www.orl.co.uk*
- *http://www.dialogic.com*
- Dialogic Documentation
- omniORB Documentation
- Orbix Documentation
- Visual C++ Documentation