Distributed Software Components

Rensselaer CSCI-6964

Last updated:[*] April 21, 1999

This two-credit graduate course is being offered in the Spring 1999 semester. It meets Thursdays 2-3:50pm in Carnegie 101. Check this web site frequently during the semester for the latest information. The resource list in particular will be updated extensively.

General Information

Instructor

Professor Dave Musser
Lally 313, 276-8660, musser@cs.rpi.edu
Office Hours: Tues & Thurs 10:00-11:30am or by appointment

Brief overview

This course examines concepts of software components as a basis for developing distributed systems. Essential properties of server, client, and dual server/client components are studied, with particular attention to structuring dual server/client components to support generic programming at the distributed systems level. Students formulate and carry out team projects to develop experimental components and applications using CORBA or DCOM. Subtopics include basic CORBA programming, CORBA Interface Definition Language (IDL), IDL to C++ type mappings, exception handling, CORBA and DCOM integration, Java/CORBA programming, and the CORBA Interface Repository. The exact choice of topics depends on the interests and background of the students. 2 credit hours.

Prerequisites

CSCI-2300 Data Structures and Algorithms[1] or equivalent. **Note: the prerequisite listed in the Class Hour Schedule is incorrect.**

The main background needed is programming experience and familiarity with data structures and algorithms. Programming will be done primarily in C++. The CORBA Interface Definition Language (IDL) is covered in the textbook and will be discussed in class. It relates closely to the C++ features that support object-oriented programming—classes, encapsulation, inheritance, dynamic polymorphism (using class derivation and virtual functions). If you are unfamiliar with the purpose and use of those features of C++, you will need to learn or review the basic ideas on your own (there are dozens of C++ books that emphasize object-oriented programming). But we will deal with some of

[*] The most up to date version of this information will be found at http://www.cs.rpi.edu/~musser/dsc/index.html.

[1] http://www.rpi.edu/dept/catalog/98-99/Courses/CSCI.html#2300

the nitty-gritty details in this course as we look at examples using CORBA. One nice aspect of learning IDL is that it presents a simpler view of O-O design issues than does C++ or most any programming language. In any case, although programming distributed computing systems using CORBA is obviously most closely related to the O-O programming paradigm, one major goal of this course is to examine to what extent another paradigm—*generic programming*, sometimes known also as *algorithm-oriented programming*—can be used.

So how much knowledge of generic programming principles is assumed? Let's discuss this ....

Familiarity with network hardware or software protocols would be useful to some extent (mainly to better understand what kinds of errors can be encountered when distributing a computation over a network), but it is not essential. (Maybe we'll draw upon the experience of some of the students who do have such background.)

What it's not

This course is not on, but does draw ideas and goals from

- software engineering
- object-oriented design and implementation
- generic programming
- literate programming (on which, more later)

Homework, projects, and grading

There will be homework assignments and a programming project (no exams). You will not be required to turn in homework assignments, but you are encouraged to do so, via email, at any time—feel free to submit incomplete solutions and ask for help if you are stuck. These homework assignments will not count as part of the course grade; they mainly are for you to get some practice with and test your comprehension of the ideas discussed in class. Looking at your solutions and questions also helps me assess which points need further discussion and which don't.

The course grade will be based mainly on your project, including a detailed written report and a presentation of it to the rest of the class near the end of the term. It will be a team project. Requirements on team size and further details are given in a later section.

Collaboration

You are encouraged to collaborate with other students in the class on the homework and programming project. If you do collaborate you must write up solutions on your own and acknowledge your collaborators in the write-up. If you obtain a solution with help (e.g., through library work, another student, etc.), acknowledge your source, and write up the solution on your own.

I don't expect any incidents of academic dishonesty in this course, but any that do occur will be dealt with according the policies stated in the *Rensselaer Handbook*.

Resources

Textbooks

Seán Baker, *CORBA Distributed Objects Using Orbix*, Addison-Wesley, 1997, ISBN 0201-92475-7.

There are many other books on CORBA. Most of them seem to be long on propaganda (for distributed computing in general as well as CORBA) and short on actual CORBA programming issues and examples. Nevertheless it would probably be useful to have at least one other book. Watch this spot for suggestions.

Selected lecture notes and example code

Getting started with CORBA

This is the front office example discussed in Chapter 2 of the textbook. Thanks go to Shawn Bisgrove for adapting the textbook source code to work with MICO, both the FreeBSD and the Windows versions. See his `README` file for more details. See also his DSC web page[2] for further information related to this example, including a POA version (see Section 4.4 of the MICO Manual for information about the Portable Object Adaptor).

Introduction to CORBA IDL

Overview adapted from Chapter 4 of the Orbix Programmer's Guide; also corresponds fairly closely to Chapter 3 of the textbook. There is also a postscript version.

IDL to C++ mapping of basic and complex types

A series of small IDL source files (`*.idl`) that you can compile with an IDL compiler to see how it maps some of the basic types to C++. Based on Chapters 4 and 5 of the textbook. You can browse the `*.h` and `*.cc` files the MICO IDL compiler produces; these are the files I was showing in class. *Try these on your own and continue experimenting with the mappings discussed in the remaining sections of Chapters 4 and 5.* There is a zip file you can download containing the existing `*.idl` files.

Matrix-vector product example

This file directory contains the example discussed in class on 2/18/99 (slightly extended). We first discussed matrix0.cpp, a matrix-vector product computation done in ordinary sequential computation, with no use of CORBA or any other kind of distributed components. However, some design choices were made with the goal of ultimately using some of its code and generic programming techniques in a distributed version. (As mentioned in class, these programs are only intended as simple exercises aimed at exploring ideas about combining generic programming techniques with distributed computing. For a serious effort at providing distributed components for linear algebra one should start with close examination of efforts such as the PETSc library,[3] which does distributed computation but doesn't employ generic programming, or the Matrix

[2] http://www.cs.rpi.edu/~bisgrs/cs.html

[3] http://www-unix.mcs.anl.gov/petsc

Template Library (MTL),[4] which makes extensive use of generic programming but doesn't (yet) do distributed computation.)

We then discussed a crude distributed version, with

- A server that provides a matrix object equipped with a `block_product` function, which is a function that is able to use a designated range of rows of a matrix `m` and a column vector `c` and produce the corresponding range of elements of the column vector `r` that is the matrix-vector product of `m` and `c`. This server writes the stringified object reference of the matrix to a file whose name is given by a command line argument. One should start two copies of the server, each with a different file, like this:

  ```
  server matrix1.ior
  server matrix2.ior
  ```

- A client that connects to two matrix servers holding the same matrix and uses their matrix `block_product` functions. The call of the first server's `block_product` function computes the top half of a matrix-vector product and the call on the second server does the second half. (In this program these calls are not done in parallel. Experimenting with that is one of the next steps.) Assuming the above commands for starting the two servers, one should start the client like this:

  ```
  client matrix1.ior matrix2.ior
  ```

  This assumes these files exist on the client's file system. If that file system is not shared with that of the servers, you must copy them manually to the client's file system. Another further step in developing this example is to avoid having to pass object references via files, by using the CORBA Naming Service.

See the `server1.log`, `server2.log`, and `client.log` files for the results of this scheme.

Try out this example yourself; experiment with running it as it is, then extend it! We will discuss some extensions in class, but that will be a lot more meaningful if you have already gotten thoroughly familiar with this basic version.

To compile and link the client and server, copy the zip file to a directory you own and unzip it. If you are using Mico on a Windows platform with VC++, do

```
nmake/f Makefile.win32 all
```

If you are using Mico on a Unix platform (monica or yoyo), do

```
gmake all
```

Unfortunately, linking with the Mico library on Unix is still taking a lot longer than it should.

Parallel version of matrix-vector product example

This file directory contains the MICO version of the example discussed in class on 3/4/99. (The code we looked at in class was actually the omniORB version.) It differs from the initial matrix-vector product version described in the

[4] http://lsc.nd.edu/research/mtl

preceding section mainly in that the first call in the client on `block_product` is performed using a Dynamic Invocation Interface (DII) request, with the request being sent to the server via a `send_deferred` call. Thus, according to DII semantics, it is non-blocking, and the client is able to proceed with the second `block_product` call to another server without waiting for the first one to finish. The example illustrates both using the DII—including some details that are not covered in the textbook (Chapter 15), such how to pass and receive objects of user-defined types—and using type `Any`. The main code changes between the sequential version and this parallel version are in `client.cpp`; no changes to the server code were necessary to make the server instances work in parallel. Note that some changes were necessary also in `Makefile` and `Makefile.win32`, to get the IDL compiler to generate extra code (such as type conversions between user-defined types and type `Any`) needed by the DII.

I also made a minor change in the server and client to have each get the matrix size, N, from the command line, and to check for the proper number of command-line parameters. I haven't made this change yet in the omniORB version.

I've compiled this example on Windows 95, with `nmake/f Makefile.win32 all`, and on SunOS, with `make all`. In the latter case I used Shawn Bisgrove's installation of MICO on dishwasher; see his DSC web page[5] to see how to set up your bash environment variables to point to his installation and get the proper `MakeVars` file. It doesn't seem to be possible to compile using the FreeBSD version of MICO on monica, since that version doesn't recognize the requisite IDL parameters for generating DII-related information. Anyway, with the SunOS version the linker works reasonably fast, which is not the case with the FreeBSD version.

I've run this example successfully with several different client-server configurations, including:

- everything running on dishwasher;
- everything running on rossini (my Windows 95 laptop);
- one server running on dishwasher, the other server on verdi (the SunOS system in my office), and the client on rossini;
- one server running on dishwasher, the other server on rossini, and the client on verdi.

With the simple way that object references are handled in this version, the IOR file created by a server must be copied to the client's machine, if they are not on the same file system. This isn't necessary, however, with the omniORB version, which uses the CORBA Naming Service.

CORBA Trading Service

The Trading Service is one of the CORBA Service Specifications. It can be seen as a generalization of the Naming Service; instead of merely providing a way for clients to search for available servers by name, it provides for servers to register their capabilities and clients to find them based on *properties*, specified via a simple *constraint language*. The full capabilities of the Trading Service are specified in official Trading Service Specification (local copy). There is an overview description in the textbook, but neither it nor the official specification has any significant example. There is a simple but helpful example in the following presentation:

[5] http://www.cs.rpi.edu/~bisgrs/cs.html

CORBA Trading Service, by Matt Verminksi[6]

Aside from their intended utility, the Naming Service, the Trading Service, and other CORBA Service Specifications are useful as examples of the use of IDL, since their interfaces and operations are presented in terms of IDL.

See also the CORBA Services Specifications section.

On MICO The MICO 2.2.5 and later distributions implement the Trading Service. There is a small example of use of the Trading service in the MICO distribution. The original version can be seen in Shawn's directory /cs/bisgrs/local/doc/mico/examples/trader, but some minor modifications were required to get it running here. The result is in

http://www.cs.rpi.edu/~musser/dsc/mico/examples/trader

I'll list the modifications, since similar changes may be necessary to get other MICO examples working:

- Create an `examples` directory and copy `MakeVars` from `/cs/bisgrs/local/doc/mico/examples/` into it.
- Create a `trader` subdirectory of your `examples` directory and copy the files in `/cs/bisgrs/local/doc/mico/examples/trader` to it.
- Copy `/cs/bisgrs/local/include/mico/typerepo.h` to your trader directory. Edit it, replacing all occurrences of `<../../mico/trader.h>` to `<mico/trader.h>`.
- Edit `main.cc`, changing

  ```
  #include <mico/typerepo.h>
  ```

  to

  ```
  #include "typerepo.h".
  ```

(These changes are necessary because the files as distributed assume you will compile and run the trader demo from the directory where the distribution installs it.)

With those changes the files should compile and link when you do `make`. (Note that you have to have set up bash environment variables as described by Shawn, and you must be running bash.)

One other change is required, in the run script. Change

```
`pwd`/../../coss/trader/traderd
```

to

```
/cs/bisgrs/local/bin/traderd
```

Microsoft's Vision of Distributed Objects: The COM Model

Guest lecture by Richard J. Bonneau, Compaq Computer Corporation

Here are the Powerpoint slides[7] that Dr. Bonneau prepared for this lecture and for his CS colloquium talk, include those on DNA (Distributed interNetworking Applications) that he did not have time to present in full. There are additional files from some of the demos he gave.

Thanks once again to Dr. Bonneau for being so generous with his time and giving us such an excellent introduction to COM distributed components!

[6] http://www-mtl.mit.edu/~mvermins/corba/trader/ppframe.htm
[7] http://www.cs.rpi.edu/~musser/dsc/DCOM/RPI-talk.ppt

Literate programming

The code we discussed earlier for the matrix vector product example is not documented, except for a few comments in the code. Here is another version, which is essentially the same functionally as the omniORB version that uses the naming services. The main difference is in the way the program is written and documented in the "literate programming style," as briefly discussed in the document itself. Both documentation and code are prepared in a single source file, called a Nuweb file. Running the Nuweb processor on this file produces as output both a LaTeX document file and the code file(s) for input to a compiler (and any other files prepared in the document such as makefiles or sample input files). The main idea behind literate programming is that many programs are more often read by humans than executed, so it's vitally important to make them readable and to ensure that documentation remains consistent with the code. Another key idea is that the goal of presenting programs in segments arranged in a logical order (rather than the order required by the compiler) actually helps the programmer to design good programs in the first place. So one should *not* first write the code and then start reorganizing it for purposes of documentation; instead it's much better to code and document at the same time.

- Document: matrix.ps (Postscript file)
- Source file input to Nuweb to produce the document and code files: matrix.w

There is further information on Nuweb in the Other Languages and Tools section.

CORBA

CORBA FAQ

The hyperlinked CORBA FAQ,[8] by Rob Applebaum, Jim Watson, and Marshall Cline, may be useful for understanding/reviewing basic CORBA concepts and some detailed issues.

Orbix

Orbix is a commercial implementation of the CORBA standard. It is a product of Iona Technologies[9] for which we have a software license on CSLab Sun Unix machines.[10] It lives on CSLab servers at `/usr/local/iona`. It provides bindings to C++ and Java. Detailed manuals are provided in `/usr/local/iona/Orbix_2.3c/doc` and also here:

- Programmer's Guide
- Programmer's Reference

[Note: don't attempt to print out either of these in its entirety unless you have your own printer and paper—each manual is over 400 pages long. In any case you should wait until we do some checking whether a newer version is available.] There is also an *Administrator's Guide* on the server.

[8] http://www.cerfnet.com/~mpcline/Corba-FAQ/

[9] http://www.iona.com

[10] At the moment applications can be compiled with the Orbix IDL compiler and SUN CC compiler, but we can't run the ORB because our license for it has expired. This should be corrected soon. In the meantime see also the MICO implementation described next.

IDL and code files for some of the examples discussed in the *Programmer's Guide* are included in `/usr/local/iona/Orbix_2.3c/demo`.

The Java binding is provided by OrbixWeb. For further information follow the Orbix link on Alan Bivens' web page.[11]

MICO

"The acronym MICO expands to *M*ICO *I*s *CO*RBA. The intention of this project is to provide a *freely available* and *fully compliant* implementation of the CORBA 2.2 standard. ... Our goal is to keep MICO compliant to the latest CORBA standard. The sources of MICO are placed under the GNU-copyright notice."—from the MICO web page.[12]

On SunOS Use Shawn Bisgrove's installation of MICO; see his DSC web page[13] to see how to set up your bash environment variables to point to his installation and get the proper `MakeVars` file.

On FreeBSD Nathan Schimke has built MICO on a FreeBSD/Pentium platform. It can be used on at least two CS department machines, monica.cs.rpi.edu and yoyo.cs.rpi.edu. To get started, see the README file in the directory /usr/local/doc/mico/examples, which directs you to copy that directory to your home directory and try some of the examples. You can also browse a copy of the examples directory here. You may also want to look at a compressed postscript[14] version of the MICO Manual; Chapter 3 in particular takes you through a small example.

I was able to compile, build, and run the account and account2 examples with essentially no problem (but see below). The first of these isn't very interesting because it doesn't separate the server and client, but the second one does. Although the shell script in account2 runs the client on the same machine as the server, you can start the client separately on another machine. I tried running the server on monica and the client on yoyo, and it worked. I encourage you to try this example and other examples yourself.

The only problem I encountered so far is that the linking phase is extremely slow—on the order of five to ten minutes to build each of the client and the server. Surely this must be due to some configuration problem, but we'll have to see what the time can be reduced to before committing to making extensive use of this CORBA implementation. (Nathan is also installing some other free CORBA implementations and is working on getting the license for Orbix reinstated.)

On Windows If you want to install MICO on Windows 95/98/NT and Visual C++, there are a couple of possibilities, depending on which version of the compiler you have.

1. If you have VC 5.0: Shawn Bisgrove and Scott Bonneau independently succeeded in building MICO 2.2.4 on Windows NT and VC 5.0 plus Service Pack 3. Scott has made the binaries available.[15] But Service Pack 3[16] is necessary even if you are not building the system, only using it.

[11] http://www.cs.rpi.edu/~bivenj

[12] http://diamant-atm.vsb.cs.uni-frankfurt.de/~mico/

[13] http://www.cs.rpi.edu/~bisgrs/cs.html

[14] Ghostview will uncompress the file and display it automatically. You can obtain Ghostview free; see the Utilities section.

[15] http://www.cs.rpi.edu/~bonnes/mico/win32-bin.zip

[16] http://www.microsoft.com/msdownload/vs97sp/full.asp

Shawn supplied the following information which may be useful to others wanting to build MICO on VC 5.0 themselves.

> Last night, I was able to build MICO using Windows 98 and Visual C++ 5.0 with Service Pack 3. This requires some manual setup of the Visual C++ compiler not done by the install program (in the area of dos compilation).
>
> First, the Visual C++ bin directory should be present in the path and a batch file `vcvars32.bat` must be run to enter all of the environment variables needed to run `nmake` from the dos prompt. Second, `nmake` is invoked using: `nmake /f Makefile.win32 w95-all`, not `nmake /f Makefile.win32 win95-all` as the documentation states. Lastly, the `win32-bin` directory that is created should be placed in your path for easy invocation of the idl, micod, and ird. The examples will call these files through batch files that need to be slightly modified to run on Win98 (currently written for Windows NT).

2. If you have VC 6.0: the MICO 2.2.4 zip file from the MICO web page doesn't compile because of a compiler bug, but I found a small workaround. (The problem was that there was already a workaround for a bug in VC 5.0 but that workaround bumped into a different bug in VC 6.0; see the CHANGES file for a little more detail.) The modified version is available.[17] To compile it, follow the instructions in the README-WIN32 file. The information Shawn Bisgrove gives above for 5.0 applies here too, except that I already corrected the `nmake` invocation in the README-WIN32 file. Using the resulting binaries and library, I've been able to run the account2 example with the server on monica and a client on my laptop. (But so far not vice versa.)

MICO and Java/C++ interoperability Shawn Bisgrove tested Java/C++ interoperability by adapting the textbook's Front Office example and using JDK 1.2, MICO 2.2.4, and idltojava (available free from java.sun.com when you register as a developer). See his DSC web page.[18]

Additional MICO-related web links, files, and examples See Shawn Bisgrove's DSC web page.[19]

OmniORB

OmniORB[20] is another free CORBA implementation, distributed under the GNU public license by AT&T Laboratories Cambridge (formerly by ORL, which was recently acquired by AT&T). Steve Cerniglia has been experimenting it and has adapted the textbook's front office example to work with it. I've also adapted the matrix-vector product example to work with it. This version of the matrix example also differs from the initial version discussed in class on 2/18/99 in that it uses the CORBA Naming Service (as does Steve's second version of the front office example). The Naming Service is described in Chapter 7 of the textbook.

- Steve Cerniglia's `readme` file describing the steps he took to bring up the front office example in omniORB.

[17] http://www.cs.rpi.edu/~musser/dsc/mico/mico-2_2_4.zip
[18] http://www.cs.rpi.edu/~bisgrs/cs.html
[19] http://www.cs.rpi.edu/~bisgrs/cs.html
[20] http://www.uk.research.att.com/omniORB/

- Front office, version1
- Front office, version2
- Matrix-vector product example using the Naming Service
- Parallel version of the matrix-vector product example using the Dynamic Invocation Interface (DII)

Installation of omniORB on FreeBSD platforms (monica, yoyo, etc.) is imminent. If you want to install it on your own Windows 95/98/NT platform, download it from the omniORB web site mentioned above.

CORBA Services Specifications

- A hyperlinked catalog of CORBA Services[21] (lists interfaces but doesn't explain their functionality).
- CORBA Services Features Matrix.[22]
- A local copy of the CORBA Collection Services specification.
- A local copy of the CORBA Trading Service specification.

WWW resources

- The Object Management Group,[23] central site for CORBA-related information.
- A brief overview of CORBA.[24]
- Douglas Schmidt's CORBA web site,[25] which contains many CORBA-related links.

Other languages and tools

C++

Compilers At the moment, at least, it seems to be necessary to use the Sun CC compiler rather than g++ when compiling Orbix applications, but information on other available C++ compilers is provided here for general use.

- **Sun C++ (CC on CSLab Sun Unix machines)**, information to be supplied.
- **GNU C++ (g++)**, version 2.8.1, available on CSLAB and RCS systems. If you want to install it on a Windows PC, visit the GCC[26] web page at MIT.

  Strengths: Supports most template features and has a good version of the standard library, including SGI STL, which is one of the best STL versions. Free. Source code is available for both the compiler and the library.

  Weaknesses: Doesn't support namespaces (it will accept but ignore a *using namespace* declaration). Poor optimization. Produces bloated executable files.

[21] http://www.barth.netliberte.org/cos-doc/index.html
[22] http://www.vex.net/~ben/corba/cosmatrix.html
[23] http://www.omg.org
[24] http://objectworks-usa.com/papers/corba.htm
[25] http://www.cs.wustl.edu/~schmidt/corba.html
[26] http://www.gnu.ai.mit.edu/software/gcc/gcc.html

- **egcs**, version egcs-2.91.57 19980901 (egcs-1.1 release), is available on CSLAB computers:

  ```
  /usr/local/egcs/bin/g++
  ```

  According to the egcs project home page,[27] "egcs is an experimental step in the development of GCC, the GNU C compiler" (but it also compiles C++). It is free.

- **MS VC++**, version 5.0 or 6.0. See the MICO section for discussion of using either version of this compiler with MICO. If you stick with version 5.0, you'll need to install Service Pack 3.

  When using STL components with this compiler you frequently get compile-time warning messages

  ```
  C4786: symbol greater than 255 characters
  ```

  This refers to internally-generated symbols that are used in the debugger. Such symbols longer than 255 characters are truncated, which may cause confusion when debugging, but otherwise they are not a problem. To shut off the annoying warning messages, include the following line in your source program:

  ```
  #pragma warning (disable : 4786)
  ```

- ... (if you have information on any of the above or other compilers, please send me email)

WWW resources

- The home page of Bjarne Stroustrup, the creator of C++.
- C++ Moderated Newsgroup: comp.lang.c++.moderated
- The C++ Standard: a press release[28] about the final, official standard, which is available now for purchase[29] in either electronic ($18) or hardcopy form ($175). An earlier draft is available online[30] in hyperlinked form. If you have a PC and want to have (this earlier draft of) the standard on it in hyperlinked or other formats, you can download it from The University of Warwick.[31] [Note: don't attempt to print out the standard unless you have your own printer and paper—it's over 700 pages.]
- The C++ FAQ
- The Dinkum C++ Library Reference (including a complete online STL reference manual, but also covering other important parts of the library such as the `string` class).
- Silicon Graphics STL (including a complete online STL reference manual)
- STL Newbie Guide
- Yahoo Index on C and C++

[27] http://egcs.cygnus.com
[28] http://www.ncits.org/standards/pr14882.htm
[29] http://www.ansi.org
[30] http://www.cs.rpi.edu/~musser/ap/c++std/cd2
[31] http://www.maths.warwick.ac.uk/c++/pub/

Emacs

GNU Emacs, "the extensible, customizable, self-documenting real-time display editor," is available on most campus computers. If you want it on your own Windows PC, visit the GNU Emacs[32] web page at MIT.

Nuweb

See also the Literate Programming section. For the moment, on CSLab workstations run Nuweb using the command

```
~musser/public/nuweb file
```

where `file.w` is a Nuweb source file. (There is an older version of Nuweb on the CSLab file system, but the above version is better. The bug I mentioned in class 3/18/99 has been corrected.) On RCS *SUN* systems, run it with

```
~mussed/public/nuweb file
```

For running Nuweb on a Windows PC, download a binary executable. Or download and compile the source file, which is itself a Nuweb file that produces both the C source code files of the implementation and the latex source of the Nuweb Manual, as described below. Before running Nuweb on this file you must create the following directories to hold the C code: `bin`, `doc`, `inc`, and `src`. This assumes you already have a Nuweb executable. If not, download a zip archive[33] containing both the Nuweb source and the C code, and compile the C code to get a nuweb executable.

The following document describes the Nuweb command set (which is very simple) and is also useful as a moderate-size example of the literate programming style: it documents the design and implementation of Nuweb itself.

Nuweb Manual (postscript file, about 80 pages)

You probably want to print out just the first section of the manual, which describes the commands. (View it with ghostview and select the pages you want for printing.)

A nuweb mode for Emacs is provided by nuweb.el (an elisp file that can be loaded into Emacs with `EscX load-file`).

There is a bug still remaining in the modified NUWEB: it is necessary to quote `$` and `@` characters by prefixing an `@`, writing `@$` and `@@`, *within nuweb parts* as well as outside of parts. (The older NUWEB did not require this quoting within scraps.)

LaTeX

LaTeX is available on CSLab, RCS, and probably most other campus computers. To download a LaTeX system (free) for use on a Windows PC, visit

MiKTeX[34]

I recommend the following manual:

Leslie Lamport, LaTeX: A Document Preparation System User's Guide & Reference Manual, Addison-Wesley, 1986, ISBN 0-201-15790.

[32] http://www.gnu.ai.mit.edu/software/emacs/emacs.html
[33] http://www.cs.rpi.edu/~musser/gp/nuweb.zip
[34] http://www.inx.de/~cschenk/miktex/index.html

Borders or Barnes and Noble should have it, or try online at Amazon[35] or Bookpool[36] (usually cheaper but less likely to have it in stock).

See also my short note on common LaTeX errors:

> D. R. Musser, Elements of LaTeX Style, November 24, 1998.

Other utilities

Windows binaries For installation on a Windows PC, some critical utilities can be downloaded from here in binary form:

- tar.exe, for archiving/unarchiving
- gzip.exe, for compression/decompression

Postscript viewer If you have your own PC you can (and should) install a postscript viewer on it. The Aladdin Ghostview page offers free downloads for various platforms. For a Windows PC, get `gsv27550.exe` (self-extracting archive file); I haven't tried the downloads for other platforms.

[35] http://www.amazon.com
[36] http://www.bookpool.com

Projects

The projects are a way of consolidating and putting into practice some of the principles covered in the course, including not only programming techniques and language knowledge but also understanding of important computer science concepts and how they are related.

Project requirements

- Develop one or more distributed software components and demonstrate their use in a distributed system. "Develop" includes doing concept development,[37] design, implementation, experimentation, testing, and documentation.

- The distributed software components you develop should fit into one of the following frameworks: the CORBA 2.0 (or higher) standard, the Microsoft DCOM specification, or both (exploring CORBA/DCOM interoperability).

- Since one of the main goals of this course is to explore how generic programming can be introduced into the realm of distributed software components, you should attempt to make your components generic in the sense of being specializable ("configurable" may be a more accurate term in this context) to many different cases.

- You may implement your components and distributed system using C++, Java, or a combination (taking advantage of the interoperability supported by ORBs such as OrbixWeb).

- In specifying and implementing your components you should avoid dependence on non-standard features provided by individual vendors, so that the components are portable to any implementation of the standard (not an issue with DCOM). In the distributed system you develop, some use of non-standard features may be necessary, but keep it as limited as possible.

- Observe the documentation requirements and guidelines given in a later section. Note the requirement of a literate programming approach. This certainly impacts your documentation but I strongly encourage you also to employ this approach throughout the in the development of the software, not as an afterthought.

- You must form **teams of three or four** people and submit one implementation and report for the team. You may divide up responsibilities for the project as you wish, but each member of the team should keep fully informed about the design decisions, experimental results, testing results, and documentation.

- It is better to limit the scope of your project to a small task and do an excellent job on it than to attempt too much and not be able to complete it or do it well.

[37] "Concept" is used here with the technical meaning of a "family of abstractions that satisfy a common set of requirements," as exemplified in the Silicon Graphics Standard Template Library Programmer's Guide.

Important dates

- By **Monday, March 1**, inform me by **email** of what you plan to do for your project (one message per team), briefly describing both the components and the distributed application program you plan to develop (about one paragraph each).
- By **Wednesday, March 17**, submit by email a preliminary report on your project, as a basis for discussion in individual sessions with me in the week of March 22. Guidelines for the contents of this report are given in the next section.
- During the week of **March 22**, I will hold 40 minute sessions in my office to meet with teams about the current state of their project. Two or three of these meetings will be during the regular class period; we will not meet as a class in this week. The meeting schedule is given in a later section.
- On **Thursday, April 22**, in class, submit your “final” document. (“Final” is in quotes because I may ask you to revise it and resubmit it.)
- Each team will make a 20 minute oral presentation and demonstration of their project to the class, on either **Thursday, April 15**, **Thursday, April 22**, **Thursday, April 29**. See the project:schedule in a separate section.

Preliminary report contents

What to submit in your March 17 preliminary report:

- Brief overview description of the components you are developing, and the key idea(s) of the generic algorithm(s) or data structures involved (about one half to one page total), including why you think these components might be useful.
- Brief description of your test plan—what kind of tests you will set up to test for correctness and performance (about one half to one page total).

Project presentation schedule

Teams and projects

1. Shawn Bisgrove, Jan Kratky, Cory Plock, “Graph Interface Proxy Components Using CORBA”
2. Michael Czupryn, Stephen Cerniglia, Chris Seifert, “News Distribution Components”
3. Scott Bonneau, Chris Parker, “A System For CORBA-Based Parallel Computation”
4. Alan Bivens, Michelle Conway, Shri Karandikar, Ranjeeta Sinha, “File Synchronization Components”
5. Kenn Flynn, Anand Paka, Priya Rajagopal, “Distributed Generic Sorting Algorithms”
6. Ming-Shu Chung, Gang Chen, Yingmei Sun, Hui Wang, “Performance Trading Service”
7. Michael Bailey, Todd Crooks, Timothy Murphy, “Distributed Web Search Components”

Schedule for project presentations

Each presentation should fit into a half-hour time slot, including about 10 minutes to answer questions. The projector and my Thinkpad will be available for visual aids and demonstrations.

Thursday, April 15	2:00 - 2:30pm	**Team 1**
	2:30 - 3:00pm	open discussion
Thursday, April 22	2:00 - 2:30pm	**Team 4**
	2:30 - 3:00pm	**Team 6**
	break	
	3:10 - 3:40pm	**Team 3**
Thursday, April 29	2:00 - 2:30pm	**Team 2**
	2:30 - 3:00pm	**Team 5**
	break	
	3:10 - 3:40pm	**Team 7**
	finis	

Documentation Requirements and Guidelines

User documentation vs. design documentation vs. code documentation

- **User documentation**: *What* a user needs to know to use the software components to build applications; concerned with interface and abstract semantics, *not* with implementation. User documentation is best separated into a User's Guide, which introduces the software gently through key ideas, examples, and other tutorial material, and a Reference Manual, which more precisely and completely fills in the details.
- **Design documentation**: *Why* the software is designed as it is, mainly for benefit of present and future maintainers of the software; giving the rationale for interface and abstract design decisions, for the choice of particular algorithms and data structures, versus alternatives; this documentation is not for the user.
- **Code documentation**: *How* the components are implemented—coding details and design decisions that are not obvious from the code; this is not for the user but mainly for later maintainers of the code.

User documentation should not refer to the implementation design, except perhaps to help explain how components fit together (the framework); if it refers to the algorithms, data structures, or actual code, it should be only for the benefit of users who may like to know, but the information given should not be essential to proper use of a component.

For example, consider the SGI STL sort[38] function description, which describes

- what it does
- performance bounds
- stability properties

Whether it uses quicksort, merge sort, radix sort, heapsort, etc., is not (necessarily) of interest to the user. But note that for this to be true, all three of the above parts of the description must be present.

It might be mentioned in the documentation that (a version of) quicksort or introsort is the algorithm used to implement the component, but it should not be necessary for the user to know this.

User documentation is also a contract that must be maintained when code (and perhaps the whole implementation design) later changes.

User documentation of generic software components

The challenge is how to describe the abstract semantics in its full generality: just how widely is the component adaptable? This often requires creativity, and several tries—you might not realize how general the component is until you attempt to describe it carefully. Entirely new ways to use it may be suggested.

Also you may have to define new concepts, by means of sets of requirements, in order to organize your descriptions coherently.

The remaining discussion in this section will emphasize techniques for describing a component in its full generality and avoiding descriptions that would unnecessarily exclude certain uses.

[38] http://www.sgi.com/Technology/STL/sort.html

Once again, we look at examples of user-level generic component documentation from the SGI Web pages on STL.[39]

There are a few places where the SGI documentation does not describe components in their full generality, such as the `merge` function (see below).

Specifying concepts

The following definitions briefly summarize the terminology used in documenting generic software using concepts.

- Concept description: set of type requirements
- Concept: set of types that meet the described requirements
- Model of a concept: any type that satisfies the requirements
- Refinement of a concept: another concept with the same requirements plus additional ones (similar to class inheritance). If concept B is a refinement of concept A, then B has fewer models than A.
- Concept hierarchy (collection of concepts and their refinement relation)

Specifying functions

Consider the STL generic merge[40] function. Do you see how it can it be used more generally than the SGI description implies?

An effective way to describe the input-output relation computed by a function is in terms of preconditions, postconditions, and effects.

precondition: a predicate that states what condition (relation of input variables) must be satisfied by any call of the function; sometimes called a *requires* clause.

postcondition: a predicate that states what condition (relation of input and output variables) that is satisfied after the function call is completed, provided the precondition is satisfied; sometimes called an *ensures* clause.

effects: a list of changes the function makes on variables other than those considered to be outputs of the function; sometimes called a *side-effects* or *modifies* clause.

A function may throw an exception, in which it does not return in a normal manner and its input-output relation as described in the above terms does not apply. Any condition under which a function throws an exception is in general unrelated to the function's precondition:

- If the precondition is violated, an exception is not necessarily thrown. [Give an example.]
- Even if the precondition is satisfied, an exception may be thrown. [Give an example.]

---

[39] http://www.sgi.com/Technology/STL/
[40] http://www.sgi.com/Technology/STL/merge.html

When it may be necessary to explain design decisions

Sometimes, if some aspect of a component interface design is different from what is usually described in textbooks or seen in other libraries, it is helpful to the user to explain it. For example, the SGI documentation of the STL queue[41] adaptor takes the trouble to explain why the `pop` member function has a void return type.

Tips

Finally, you can help the user by including *tips* for use of components. Tips go beyond how to use a component to describe how to use it well. Example: in the SGI description of the STL vector[42] component, see the tips relating to storage management given in the notes at the end of the page.

Contents and organization of project final reports

Software documentation is often published in separate volumes called User's Guide, Reference Manual, and Design Document. Results of testing software are not usually published outside of an organization's development group, and (unfortunately for later maintainers) may not be documented at all. For the projects for this course, prepare a single document with the following sections:

- User's guide
- Reference manual
- Design issues
- Source code
- Case study

The user's guide contains the user level documentation and may contain tutorial material (such as extended discussion of examples), but for purposes of this course, tutorial material is optional.

The complete source code of a software product is not usually published, but in this case your project reports should include it. It is *required* that the code be presented in the literate programming style, using the NUWEB tool. The code that you compile and test must be the code that is extracted from the NUWEB source file for your project. You should submit your report containing the code and the other sections in both hardcopy form (resulting from running NUWEB source file → NUWEB → LaTeX → dvips → laser printer) and as an email message containing only the single NUWEB source file.

The case study should be a small example distributed system whose implementation makes nontrivial use of your new distributed software components.

The SGI web pages contain mainly overview and reference material, and some discussion of design issues; they contain only minimal tutorial material and lack any discussion of testing. They point to the code but do not use literate programming in presenting the coding details.

[41] http://www.sgi.com/Technology/STL/queue.html
[42] http://www.sgi.com/Technology/STL/Vector.html

User's guide

The User's guide should begin with an overview section. The overview section is a crucial part of the documentation and you should make sure it really does give an overview. Issues in writing a good overview include:

- Who is the intended audience.
- What is the right level of discussion for that audience, *in the overview* (you can change to a more detailed level in the later sections).
- What terms need to be defined to convey the main points to the intended audience.

Intended audience For the project reports for this course, the intended audience is *not* the course instructor.

- Aim for a broader audience such as the professional programming community, or, more likely, limited to C++ programmers (I leave some discretion here, but you should make clear how much knowledge of programming, C++, and STL or other generic libraries you assume).
- Thus the document should *not* have a title like "Distributed Software Components Course Project Report"; instead it should be descriptive of the library component(s) developed in the project.
- The documentation should *not* refer to earlier work on the project such as preliminary or progress reports; most readers won't care about the history of the project. You might include some information such as results of preliminary experiments with compiler portability or performance, but only if it helps justify design decisions of your final software (so it probably goes in the design document if it is included at all).

Level of discussion This is one of the most difficult problems of technical writing; it is tempting to dive into low level details in the overview long before the reader is ready for them. This can also be a problem in tutorial material (if any); you must introduce details a few at a time or else the reader will be confused.

One crucial issue is how much, if any, to say about design decisions in the overview. As a general rule, discuss design decisions in the overview only to the extent you believe that doing so will help convince your audience it is worth trying your software. Overviews of STL, for example, generally include discussion of many design decisions as part of the explanation of the advantages of generic programming.

Defining terms This will depend on your intended audience; the more background you assume, the more you restrict your audience, but spending a lot of space defining terms may turn off readers who are already knowledgeable.

Tutorial material After the overview, the User's Guide often contains tutorial material of various forms, particularly examples of use of the software and discussions of pitfalls to avoid. Writing good tutorials is important but is mostly beyond the scope of this project. Spend time on this part only if you have everything else completed and polished.

Reference manual

In How to use the STL documentation[43] note the descriptions of the format of concept pages, of type pages, and of function pages. You should take these as guidelines for writing the reference manual part of the documentation for your project.

Design issues

In this section describe the key design decisions. This includes interface and abstract semantics design decisions, and why particular algorithms and data structures were chosen, versus alternatives. Remember that this documentation is not for the end-user, and thus may be more technical than the User's Guide.

As much as possible, separate the design issues into an abstraction level and an implementation level:

- The *abstraction level design issues* concern the interface, semantics, and performance requirements.
- The *implementation level design issues* concern the choice of internal algorithms and data structures that meet the abstraction level requirements. Do not discuss details that would be obvious from reading the code, but if you have used nonobvious optimizations in your code then either here or the next section (source code) is the place to describe them.

Occasionally you may need to refer to some of the implementation issues in justifying your abstraction level design choices.

Some further points:

- It may be useful to make comparisons to other libraries or software packages. Though not essential, such comparisons can help readers who might have knowledge of other software put your library in perspective. Avoid comparisons to software that is not widely known, since only a few people would be able to appreciate the comparison. Also avoid denigrating competing software or design decisions with phrases like "an awful hack"; keep your comparisons on a technical basis.
- Give a detailed rationale for your design decisions if you think it will either (1) help convince readers to use your software, or (2) help future maintainers, re-implementors, or extenders of your software to avoid making other design choices that you know are inferior (because you tried them, or rejected them on the basis of careful analysis).
- By the same token, avoid rationalizing decisions for which you have little supporting evidence; it's better simply to say that other decisions were possible but were not explored.

Source code

As mentioned above, you are required to present the complete source code using the literate programming style. In some cases you may find it easier to present design decisions and source code if you combine them in a single section. This is permitted.

Keep in mind the flexibility the literate programming style permits: you can present the most important parts of your code first, followed by the details,

[43] http://www.sgi.com/Technology/STL/doc_introduction.html

assembling the parts into an order acceptable to the compiler in an "overview" or "outline" part (which can come either before or after the other parts). You may even relegate nitty-gritty details to an appendix so they don't make the main discussion too long.

Case study

The case study should be a small example distributed system whose implementation makes nontrivial use of your new distributed software components. [[More on this later.]]

Other essential steps in preparing your report

1. Pay attention to grammar, spelling, and punctuation. Any problems in these areas can distract readers from your main message, or, worse, drive them away entirely.
   (a) Read, or reread, Strunk and White, *The Elements of Style*. It's short, pleasant to read, and always helpful.
   (b) *Always* run your documents through a spelling checker before submitting them to anyone else to read.
   (c) Remember also to check manually for problems that a spelling checker can't detect (see below). Often it's very difficult to see such errors in your own writing, so have someone else proofread it too.
2. Document formatting errors can also be distracting. See the LaTeX section for help in avoiding common errors.
3. Write clearly and simply, and maintain a level of formality of style. In a team project, different parts of the report will typically be drafted by different members of the team, with inevitable differences in style. One team member should take the role of editor, resolving such differences to achieve a single writing style in the final document.

Ode To My Spell Chequer
Author Unknown

I have a spelling chequer,
It came with my PC.
It plainly marks for my revue,
Miss takes I can not sea.

I've run this poem threw it,
I'm sure your please too no.
Its letter perfect in it's weigh,
My chequer tolled me sew.

Project Reports

Graph Interface Proxy Components Using CORBA

by Shawn Bisgrove, Jan Kratky, and Cory Plock

This report[44] (postscript file, 428k, 135 pages) describes the implementation of a set of proxy components enabling the creation of a distributed graph system using CORBA.

For those still grappling with the mechanics of literate programming it may be useful to look at the Nuweb file, `graph_proxy.w`,[45] from which the document and all code were generated. The only other source files needed to regenerate the project report are some `*.txt` files containing some data gathered during testing. They are available, together with `graph_proxy.w`, in a zip file.[46]

See also the Nuweb section, where I've added a note about how to work around a bug in the local Nuweb implementation.

[44] http://www.cs.rpi.edu/~musser/dsc/Projects/Graph/graph_proxy.ps

[45] http://www.cs.rpi.edu/~musser/dsc/Projects/Graph/graph_proxy.w

[46] http://www.cs.rpi.edu/~musser/dsc/Projects/Graph/graph_proxy.zip

Record of Updates

4/21/99 Announcement: We will meet in **DCC 232** for the 4/22 and 4/29 classes, in order to have a network connection available for demos. Same time, 2:00pm to 3:50.

4/17/99 Added links in the Lecture notes section to the Powerpoint slides and some of the other files used by Dr. Bonneau in his presentation on COM components.

Also added a new section, Project Reports. The final versions of students' project reports will appear here. One report submitted early is already there, although it is yet to undergo final editing, since it may serve as a guidepost to other teams in writing up their project.

Also added a note at the end of the Nuweb section about how to work around a remaining bug in the modified Nuweb.

4/11/99 Changed due date for project submission to April 22. Added schedule for in-class project presentations in the (renamed) Project presentation schedule section. If your team isn't yet on the schedule, confer and send me email with your date/time preference.

4/7/99 Added new material in the CORBA, CORBA Services, and Lecture Notes sections.

3/25/99 Added a link in the Nuweb section to nuweb.el, an elisp file that provides Emacs with a Nuweb mode.

3/23/99 Added a section on Literate Programming, including a new version of the matrix-vector product example done in the literate programming style. Also added a section on Nuweb, a tool that supports the literate programming style.

3/22/99 Added a section on Projects, Teams, and Meeting Schedule. Check it to see which slots are available for your team to meet with me this week to discuss your project. Send me email to reserve a slot.

3/12/99 Removed the following paragraph from the description of the parallel version of the matrix-vector product example in the Lecture Notes section.

> The only problem I encountered in compiling with GNU C++ was getting multiple definition errors between the MICO "miniSTL" and the built-in STL header files (the `<vector>` header in particular). I got around the problem by not including `<vector>` when compiling with GNU C++, but this may not work if some parts of `<vector>` are needed but aren't supported by miniSTL.

This is no longer a problem because of a change that Shawn made in his MICO installation (he replaced the `miniSTL` directory with a symbolic link to `/usr/local/include/g++`).

3/7/99 Added notes and code from the parallel version of the matrix-vector product example (both the MICO and the omniORB versions) in the Lecture Notes section.

3/1/99 Added a link to a local copy of the CORBA Trading Object Service Specification, in the CORBA Services section.

2/24/99 Added information on omniORB and some examples of its use. A couple of the examples use the CORBA Naming Service, which is one of the topics for discussion in class on 2/25/99; see also Chapter 7 of the textbook. Another topic we'll likely get into is the Dynamic Invocation Interface (DII); see Chapter 15.

2/21/99 Separated CORBA information from the rest of the Languages and Tools section, and added a link to a local copy of the CORBA Collection Services Specification (briefly discussed in the 2/11/99 class) and a few WWW links to sites containing CORBA related information.

2/20/99 Added notes and code from the matrix-vector product examples in the Lecture Notes section.

Also added a link to Alan Bivens' web page,[47] which has information about the CORBA-Java binding provided by OrbixWeb.

Also added information on LaTeX in the Languages and Tools section.

2/11/99 Added new sections on Projects and Documentation requirements and guidelines.

2/10/99 Added links to Shawn Bisgrove's DSC web page[48] in the Lecture Notes section, at the end of the MICO section and in a new section on CORBA Java/C++ interoperability.

2/6/99 Added example code for the front office example in the Lecture Notes and Example Code section.

Also added there the small example IDL files for studying some of the IDL to C++ mappings.

Added to the MICO examples/account2 directory a `Makefile.win32`. This example builds and runs on both the FreeBSD and Windows platforms.

Also added information in the Utilities section about obtaining a free postscript viewer, Ghostview.

2/2/99 Added a link in the MICO section to a zip file containing MICO sources that compile with Visual C++ 6.0.

2/1/99 **Bulletin:** MICO is finally back up and working on the FreeBSD machines (at least monica and yoyo). It is the version that was there last week, 2.2.3. Nathan tried several times to build 2.2.4 but he's getting an internal compiler error message from gcc 2.8.1. Version 2.2.3 is still is very slow in linking but at least you can try some more of the small examples in the examples directory.

Also, added information about installing MICO on Windows 95/98/NT and Visual C++ in the MICO section.

1/27/99 Added lecture notes from last week on CORBA IDL.

[47] http://www.cs.rpi.edu/~bivenj

[48] http://www.cs.rpi.edu/~bisgrs/cs.html

1/27/99 Added information on access to, and experience so far with, a FreeBSD/Pentium-based installation of MICO. This is usable, and will be the basis of some discussion in class, but there is a problem with extreme slowness of the linker. Assuming the linking problem can be solved, it should be interesting to try some of the other examples in the MICO examples directory. In this week's class we will discuss dividing these up among the class members, so you might be looking at them and thinking about which one to choose.

1/18/99 Shawn Bisgrove reports, "I was unable to compile Mico 2.2.4 with Visual C++ 6.0, but the documentation is for Visual C++ 5.0 compiler. If anyone would like to build it and has Visual C++ 5.0, they could produce binary versions to use." Any takers?

1/14/99 Added a link to another CORBA implementation, MICO, in the Languages and Tools section. Thanks to Chris Parker and Shawn Bisgrove for tracking this down. Chris notes "The source tree includes a Win32 (95/98/NT) build with instructions on how to build it with Microsoft's VC++, although I have not yet tried it. I'm building for FreeBSD 3.0-RELEASE right now, and I'll move over to my NT box when that appears to have worked."