# *CISCA*
# A C++ Implementation of the Software Communication Architecture FileSystem

by

Greg Lanthier
Adviser: Dr. M. Barbeau
School of Computer Science
Carleton University

A thesis
presented to Carleton University
in partial fulfilment of the
requirement for the Degree of
Bachelor of Computer Science


Ottawa, Ontario, April 2003

I hereby declare that I am the sole author of this thesis.
I authorize Carleton University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Greg Lanthier

I authorize Carleton University to reproduce this thesis by photocopying or other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Greg Lanthier

# Abstract

The CORBA® specification falls short of specifying certain things that must run-time systems must use. CORBA® does not address file systems, logging, loadable modules, executable modules, or any of the other pieces that are described in detail by the 'Software Component Architecture' (SCA). A reference implementation for this architecture (called *SCARI*) was prepared by the Communications Research Council of Canada. However, this implementation only runs in a Java™ environment.

This project investigates the possibility of implementing the SCA core framework file system in standard C++ using widely available features of the CORBA® specification. The software developed during this project, *CISCA*, attempts to be as flexible as possible in terms of its software configuration, the operating systems it will run on, and in its ability to use as many ORBs as possible. In order to do so it only uses bare standard component of the CORBA® specification (like default servants) and a minimum of platform specific API calls.

The measurement of success for the implementation was a comparison between latency times during specific operations in both *SCARI* and *CISCA* as well as a comparison in the memory requirements of *SCARI* and *CISCA* when *CISCA* was run with different ORBs.

The results led the author to conclude that in most normal circumstances *CISCA* was indeed faster than *SCARI* for average message sizes but the timings and memory requirements for *CISCA* varied greatly from ORB to ORB. Additionally, various problems were encountered with different ORBs which required ORB specific workarounds. These workarounds were unavoidable with the chosen design and implementation and lead the author to conclude that while a generic C++ implementation for the SCA is theoretically possible designers and developers must have a thorough knowledge of the ORBs that are likely to be the main targets for such an implementation.

A purely generic C++ implementation that will run on any ORB and on any operating system requires a good deal of thought and design to become a reality.

**Acknowledgements**

Thanks to those involved with the SCARI reference implementation from the Communications Research Centre Canada who provided the impetus for this work.

Thanks also goes to the developers in the open software community working on the MICO and ACE+TAO ORBs.

The Windows® Makefiles for the software accompanying this project are modelled after those used by the MICO ORB.

The following software projects and tools were also used during the course of this project.

- Autoconf & Automake
  These tools helped with the cross-platform build and helped to organize building *CISCA* with different ORBs.
- CppUnit & JUnit
  These libraries provided a unit testing framework that is consistent between Java and C++
- Iona Technologies & Orbix 2000 v2.0
  Orbix 2000 v2.0 was one of the main ORBs that was used for testing *CISCA* since it ran on all the target operating systems.
- Borland & VisiBroker™ 5.2
- GCC & G++ & GNU Make
  This compiler and make tool was used on Linux and on Solaris® in producing *CISCA*.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Thesis Statement

A highly scalable and efficient FileSystem server can be implemented using the SCA[19] core framework and portable C++. By using standard and widely available features of CORBA® with standard POSIX API calls and standard language features, it is possible to implement a portable FileSystem server in C++. It should also be possible to make the implementation portable to different ORBs and different operating systems.

It is expected that such an implementation in C++ would be at least as efficient as the Java™ reference implementation prepared by the Communication Research Council Canada[10] (called *SCARI*[8]), and in some circumstances the C++ implementation would out-perform the Java™ reference implementation.

## 1.2 Motivation

The SCA was drafted to provide a common framework that manufacturers would use to build components that could be easily integrated with each other. Software-based radio is one of many possible applications that the SCA is well suited for.

The Communications Research Centre Canada[10] sponsored reference implementation of the SCA, dubbed the *SCARI*[8] project, provides an exciting first look at a fully functional example of what is possible with the SCA written in Java™.

While *SCARI* enjoys the inter-platform portability of Java™, it unfortunately concedes to some of Java™s limitations. These include the following:

- *no API for querying file system statistics*
  Unfortunately Java™ does not provide a generic means of determining how large a file system is or how much space is available. As a result, *SCARI* must resort to using a work-around that is effectively the equivalent of the C API call `FILE* popen(const char*command, const char* type)`[21] to spawn an instance of the "`df --block-size=1`" command on Unix™ or the "`cmd.exe /c dir /-c`" command on Windows® to get the space available on the host file system.

- *large runtime memory footprints*
  Often the size of a Java™ runtime can become quite large during the course of execution. It is suspected that this will be the case with *SCARI*. *CISCA* should also be able to maintain a smaller in-memory footprint by employing a design using default servants.

- *slow data marshaling*
  Explicit array boundary testing and type casting and checking are two examples of checks that Java™ run times perform on an application during execution. These and other housekeeping tasks take a certain amount of time. While each new version of Java™ makes progress towards minimizing this overhead, it still exists. A large part of CORBA® involves *marshaling* data. This means that a CORBA® application spends an appreciable amount of time transforming application data-types to and from wire-level representations for delivery to other machines using protocols like IIOP. Due to the runtime checks that Java™ performs during type casting and array operations marshaling becomes a computationally intensive operation.

Applications written in C or C++ can address any of these limitations. The intention of *CISCA* is to address all three while remaining as portable as possible. Java™ enjoys a reputation that can be summed up by Suns marketing phrase "Write once, run anywhere." With thoughtful planning and development it should be relatively straight forward to write C++ code that is portable to different operating systems, compilers, and ORBs.

It was the intent of this project to ensure that *CISCA* would run not only on different operating systems but with different ORBs as well. Open-source tools like autoconf[9][25][1] and automake[12][25] help developers build software packages that can easily ported to different flavours of Unix™. Similar techniques employed to build cross-platform source packages can be used to build software that will work with different orbs without code modification. Adding Windows® specific code in appropriate `#ifdef` blocks ensures that *CISCA* runs on Windows® as well as Unix™.

[1] A version of the book titled "GNU Autoconf, Automake, and Libtool" (often referred to as the GOAT book) is available on-line at `http://sources.redhat.com/autobook`

Addressing the items mentioned above is only one aim of this project. It is thought that *CISCA* might be able to improve on *SCARI* in selected areas. These include the following:

- *access OS information more easily*
  Java™ does not provide a generic way of access operating system specific information like space remaining on the file system, etc. APIs differ slightly across the various flavours of Unix™, but not enough to make it impossible to write adaptive code using `#ifdef`s. So to can the operating system specific code be wrapped in `#ifdef`s when compiling the *CISCA* software on Windows®.

- *see if default servants work better*
  *SCARI* doesn't use Default Servants. A File servant is allocated for each and every file that is opened on the host operating system, and all those servants are co-located with the file system servants. It is believed that this will lead to a system that does not scale well. *CISCA* uses Default Servants to see if that provides a system that scales better than *SCARI*.

  Applications like File Servers seem to be a perfect application for default servants since only one servant is instantiated on the server side while will take on the attributes of the separate file instances as they are created and used by the clients.

# Chapter 2

# Methodology

## 2.1 Design & Development

*CISCA* was developed with the hopes that it would run with as many different ORBs on as many different operating systems as possible. There were two main reasons for this desire. First, *SCARI* enjoys the portability that is afforded by using Java™ as an implementation language. In order to make *CISCA* a viable alternative it must be portable to different ORBs and operating systems. The second reason is due to the desire for code quality. Software that can be built with more than one compiler on more than one operating system typically enjoys a higher degree of quality[20]. Additionally, as *CISCA* ran with more and more operating system/compiler/ORB combinations more confidence in the correctness of its operation would be achieved.

With the goal of cross-platform and ORB independence in mind *CISCA* could only use common features of CORBA®, could not rely on operating system-specific APIs, and could not depend on features offered by a certain compiler.

Initial development was confined solely to RedHat 7.1 Linux 2.4.7-10 with MICO 2.3.8, but constant attention was paid to making sure that it would be as easy as possible to port *CISCA* to different operating systems and different ORBs once it was ready. Tools like autoconf[25] and automake[25] made it possible to take the single code base that was created on RedHat 7.1 Linux 2.4.7-10 and with slight changes using `#ifdef` blocks build and run *CISCA* on Solaris® using MICO 2.3.8.

The decision to use MICO 2.3.8 as the initial development ORB was made based on the fact that of all the ORBs that were to be used for testing, MICO 2.3.8 reported the lowest CORBA® standard compliance[1]. Any features that worked with MICO would then theoretically with all of the other ORBs.

[1]Refer to table 4.1 for more details of which specifications of CORBA® each of the tested ORBs supports.

Once *CISCA* was running on Linux and Solaris® with MICO 2.3.8 it was a straight forward modification to add support for additional ORBs. Table 2.1 is a matrix of the operating systems and ORBs with which *CISCA* was tested.

ORBs were selected for testing and development based on their availability and their reported compatibility with Linux, Solaris®, and Windows®. MICO[13] and ACE+TAO[6] were selected based on the fact that both of these ORBs were open-sourced and both run on at least two of the three operating systems that were to be tested. The author uses Orbix 2000 v2.0[7] for work related activities, so Orbix 2000 v2.0 was selected for testing and development. VisiBroker™ 5.2[1] was selected for testing because of VisiBroker™ 5.2s ability to run on all the target operating systems and because of a liberal trial license.

| | **ACE+TAO** | **MICO** | **Orbix 2000 v2.0** | **VisiBroker™ 5.2** |
|---|---|---|---|---|
| Linux | ✓ | ✓ | ✓ | ✓ |
| Solaris® | ✓ | ✓ | ✓ | |
| Windows® | ✓ | | ✓ | |

Table 2.1: Tested ORBs and operating systems

## 2.2 Testing & Correctness Evaluation

In order to have confidence that *CISCA* ran correctly and operated properly unit test cases were created using CppUnit 1.8[11] that enabled simple regression testing when changes and additions were made to *CISCA*. These unit test also ensured easy regression testing when *CISCA* was ported to different ORBs and operating systems.

Equivalent test cases were created for *SCARI* using JUnit[4] so that the behaviour of *SCARI* and *CISCA* could be compared.

## 2.3 Performance Evaluation

The tests conducted to compare the relative performance of *SCARI* and *CISCA* with each of the tested ORBs were based on those conducted by the Advanced Technology Labs at Lockheed Martin Corp[23].

In order to measure the performance of *CISCA* a test application was created that would measure the latency required for writing buffers to a file on the *CISCA* test server.

A connection to the `CF::FileManager` is created in the `sca_latency_test` program and a file is opened before any timing operations are started. A time stamp is recorded (using the

`gettimeofday(...)` on Unix™ and the `QueryPerformanceCounter(...)` function on Windows®) just before a previously allocated buffer is passed to the `CF::File::write(...)` function. Another time stamp is collected immediately after the `CF::File::write(...)` completes. The difference between these times is recorded in a log file along with a time that the tests occurred.

A similar test application was created to test *SCARI* using JNI[3] to call `gettimeofday(...)` on Unix™ and the `QueryPerformanceCounter(...)` function on Windows®.

Twenty-two different sized buffers were written to the server ten thousand times each. The size of each buffers was equal to $2^n$ where 0 <= n <= 22.

The mean latency times for each ORB were compared with the mean times for *SCARI* and are plotted in the figures shown in section 3. The relative difference in the performance is calculated according formula 2.1. The results of these comparisons are shown in figures 3.1, 3.2, 3.3, and 3.4.

$$\begin{aligned} \frac{P_{SCARI}}{P_{CISCA}} &= 1 + \frac{n}{100} \\ n &= \frac{100 \times (P_{SCARI} - P_{CISCA})}{P_{CISCA}} \end{aligned}$$

Figure 2.1: Equation used to calculate relative difference in latency times and memory consumption between *SCARI* and *CISCA*

While the latency tests were running a separate Perl script was run to record the resident and virtual size of the `sca_server` process as reported by the '`ps`' command. The resulting values were correlated with the latency testing results so that graphs of the resident and virtual sizes of the `sca_server` processes relative to buffer size could be drawn. The mean resident and virtual memory sizes of the *CISCA* `sca_server` process for each ORB was compared to the mean resident and virtual memory usage of *SCARI* using the equation shown in figure 2.1. The results of this comparison are shown in table 3.5.

**Sample Calculation**

The following is a sample of how the relative difference in latency times between *SCARI* and *CISCA* running with MICO was calculated.

Let the buffer size equal $2^{10}$, or 1024 bytes.

Table E.1 shows that the mean latency time for writing buffers of size $2^{10}$ bytes with *SCARI* is 2220.3990 $\mu$-seconds. Table E.3 shows that the mean latency time for writing the same sized

buffers with *CISCA* under MICO is 674.9735 $\mu$-seconds.

$P_{SCARI}$ = 2220.3990
$P_{CISCA}$ = 674.9735

$$\begin{aligned} n &= \frac{100 \times (P_{SCARI} - P_{CISCA})}{P_{CISCA}} \\ n &= \frac{100 \times (2220.3990 - 674.9735)}{674.9734} \\ n &= 228.9609 \end{aligned}$$

This result shows a positive relative difference of 228.96% between *CISCA* running under MICO and *SCARI*.

A positive result indicates that *CISCA* has a faster latency time than *SCARI* and negative results indicate slower latency times. When the calculations involve relative differences in resident and virtual memory usage positive results indicate *less* memory is being used by *CISCA* than by *SCARI*.

# Chapter 3

# Analysis

## 3.1 General Discussion of *CISCA* Analysis

One of the main goals of this project was to see if building a highly efficient SCA FileSystem is possible using only standard CORBA® features, portable C and C++, and by following the guidelines described in the SCA Core Framework. Consequently great attention has been given to analyzing the performance of *CISCA*.

The word 'performance' is used to describe many different things in the computer industry. Usually when someone talks about the *performance* of a system they are referring to their perception of how well a system operates. There are many different standards for measuring how well a system works but for the purposes of this project analysis will focus on *latency* and *memory usage*.

*Latency* in this context refers to the amount of time that passes between when a method is first invoked and when the method completes and returns a result.

*Memory usage* is measured by observing the *resident* and *virtual* memory sizes of the processes associated with the *CISCA* server during latency testing. Please note, however, that results for the resident and virtual memory sizes of the *CISCA* server processes will be restricted to those observed on RedHat 7.1 Linux 2.4.7-10. Differences between how memory usage is reported on different operating systems (including, but not limited to, Windows® 2000) precludes a fare comparison between memory usage with different operating systems. This report confines itself to comparing the memory usage of the *CISCA* server process when used with different middle-wares.

Section 2.3 talks about how the values for the graphs and tables in this section were gathered and calculated.

### 3.1.1 Latency times for different ORBs with co-located client and server processes

The results shown in figures 3.1, 3.2, and 3.3 compare the latency times for writing a buffer to a file with different ORBs. The client and server processes are collocated on the same machine. Refer to table 2.1 for the specific details of the machines used for testing. Tables 3.1, 3.2, and 3.3 show the relative difference in the speeds with which the write operations completed when compared with the time required for the same write operation in *SCARI*.

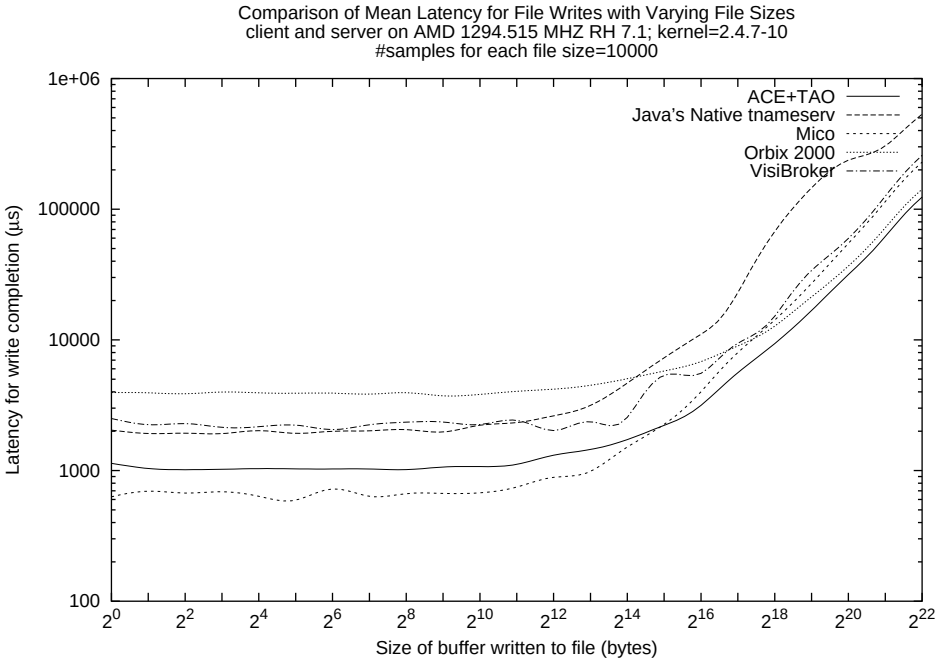


Figure 3.1: Comparison of latency times by ORB on RedHat 7.1 Linux 2.4.7-10

Figure 3.1 shows that the latency of the write operation in *CISCA* is highly dependant on the ORB that is used. For example, *CISCA* running under Orbix 2000 v2.0 proves to be the slowest of all the SCA implementations for write operations given buffers with sizes between approximately $2^{12}$ bytes and $2^{14}$ bytes. Please note, however, that in all the testing that was conducted when buffer sizes exceed $2^{14}$ bytes *SCARI* is always slower than even the slowest ORB running *CISCA*.

It is interesting to see that the results of the testing on Linux, Solaris®, and Windows® indicate that the open-sourced pure C++ ORBs MICO and ACE+TAO are consistently faster than the

commercial ORBs Orbix 2000 v2.0 and VisiBroker™ 5.2. In the case of MICO it could be said that the other ORBs incur slightly more overhead due to the fact that they support a larger subset of the CORBA® specification. MICO only goes as far as implementing CORBA® 2.3 while all of the other ORBs, with the exception of the Java™ tnamserv, support CORBA® 2.5. [1] More investigation is necessary to determine why ACE+TAO consistently had faster latency times than the commercial ORBs.

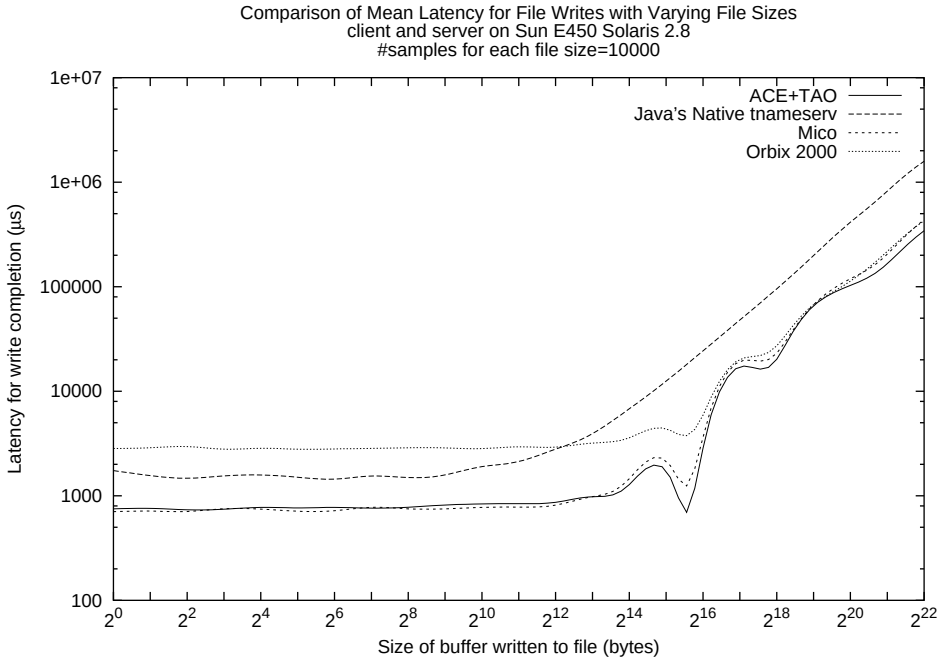


Figure 3.2: Comparison of latency times by ORB on Sun Enterprise 450 running Solaris™ 2.8

Similar results for latency times with *CISCA* running on Solaris® were observed as shown in figure 3.2. There was an appreciable improvement in the latency times for write operations involving buffers of size $2^{16}$ bytes. This inverted spike could be related to optimizations that exist within the networking module of the Solaris® kernel that the C++ ORBs could more easily take advantage of than the pure Java™ tnamserv could. More research is required to determine if this is the case. Alternatively more iterations of the latency tests may result in smoother curves.

A similar inverted spike is seen in figure 3.3 for ACE+TAO when the buffer being passed to the write operation is approximately $2^{16}$ bytes in size. Again, more research is required

[1] Refer to table 4.1 for a list of the vendor reported level of CORBA® compliance for the ORBs tested with this project

to figure out if this spike occurs because of optimizations within the networking layer of the Windows® 2000 kernel.

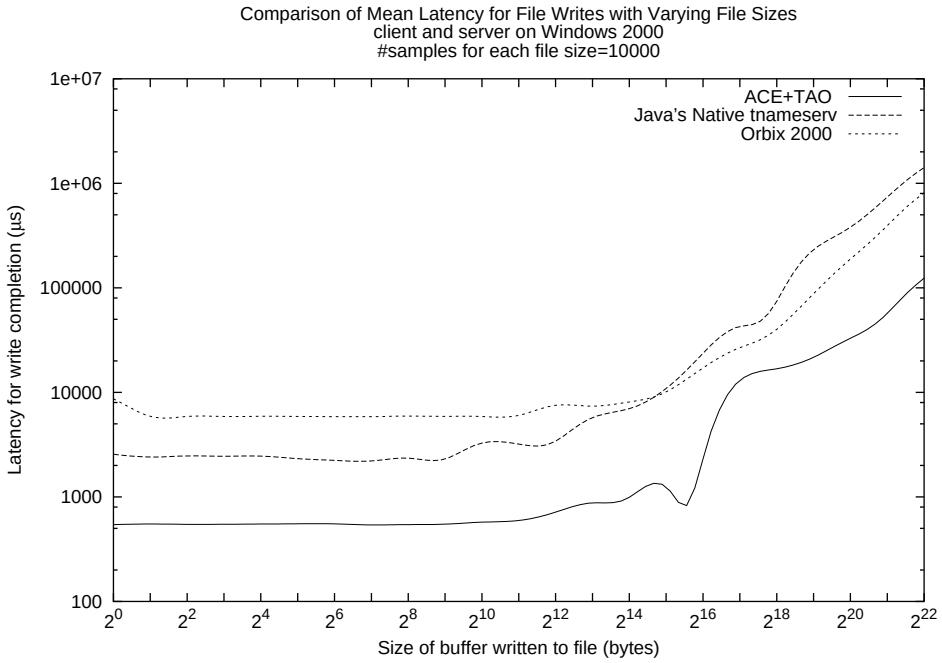


Figure 3.3: Comparison of latency times by ORB on Windows 2000

| File Sizes $(2^n)$ | ACE+TAO | Mico | Orbix 2000 | VisiBroker |
|---|---|---|---|---|
| 0 | 79.69% | 223.53% | -48.45% | -18.57% |
| 1 | 85.14% | 176.37% | -51.21% | -14.35% |
| 2 | 90.08% | 187.06% | -50.08% | -15.58% |
| 3 | 87.59% | 178.88% | -51.89% | -10.48% |
| 4 | 95.18% | 217.88% | -48.66% | -6.76% |
| 5 | 87.02% | 224.60% | -50.67% | -13.51% |
| 6 | 94.37% | 176.27% | -48.93% | -2.86% |
| 7 | 95.77% | 217.10% | -47.64% | -10.15% |
| 8 | 102.55% | 209.31% | -47.79% | -12.25% |
| 9 | 85.26% | 195.05% | -47.18% | -16.20% |
| 10 | 107.10% | 228.96% | -41.93% | -1.36% |
| 11 | 108.80% | 211.50% | -42.40% | -3.93% |
| 12 | 100.17% | 195.39% | -37.53% | 29.66% |
| 13 | 117.45% | 220.02% | -29.78% | 33.06% |
| 14 | 170.63% | 208.46% | -7.39% | 82.81% |
| 15 | 229.14% | 223.15% | 26.33% | 35.98% |
| 16 | 250.15% | 175.19% | 61.84% | 99.19% |
| 17 | 308.96% | 186.64% | 155.14% | 143.22% |
| 18 | 621.48% | 376.60% | 430.62% | 345.40% |
| 19 | 768.76% | 439.49% | 582.65% | 328.49% |
| 20 | 651.42% | 333.64% | 546.20% | 296.94% |
| 21 | 393.25% | 166.98% | 322.93% | 142.65% |
| 22 | 331.24% | 133.45% | 275.57% | 106.84% |

Table 3.1: Relative difference in latency times
*CISCA* under ACE+TAO, MICO, Orbix 2000 v2.0, VisiBroker™ 5.2 and *SCARI* on RedHat 7.1 Linux 2.4.7-10 (from figure 3.1).

| **File Sizes** $(2^n)$ | **ACE+TAO** | **Mico** | **Orbix 2000** |
|---|---|---|---|
| 0 | 132.21% | 147.14% | -38.59% |
| 1 | 105.65% | 118.38% | -45.86% |
| 2 | 100.10% | 107.91% | -50.24% |
| 3 | 108.48% | 106.14% | -44.51% |
| 4 | 104.24% | 112.11% | -44.43% |
| 5 | 96.55% | 111.37% | -46.06% |
| 6 | 86.31% | 100.68% | -48.59% |
| 7 | 102.79% | 99.46% | -45.63% |
| 8 | 92.16% | 98.91% | -48.13% |
| 9 | 94.13% | 111.19% | -44.85% |
| 10 | 127.27% | 145.62% | -32.80% |
| 11 | 152.84% | 172.90% | -27.61% |
| 12 | 222.53% | 243.50% | -4.44% |
| 13 | 300.77% | 300.03% | 22.92% |
| 14 | 430.80% | 366.07% | 89.35% |
| 15 | 617.82% | 476.74% | 185.55% |
| 16 | 752.43% | 570.01% | 311.42% |
| 17 | 179.30% | 147.99% | 138.25% |
| 18 | 372.21% | 313.28% | 251.28% |
| 19 | 202.35% | 194.61% | 193.97% |
| 20 | 301.48% | 246.95% | 266.20% |
| 21 | 394.21% | 301.20% | 276.57% |
| 22 | 361.40% | 261.70% | 270.50% |

Table 3.2: Relative difference in latency times
*CISCA* under ACE+TAO, MICO, Orbix 2000 v2.0 and *SCARI* on Sun Enterprise 450 running Solaris™ 2.8 (from figure 3.2).

| **File Sizes** $(2^n)$ | **ACE+TAO** | **Orbix 2000** |
|---|---|---|
| 0 | 370.75% | -70.56% |
| 1 | 337.14% | -59.13% |
| 2 | 351.49% | -58.08% |
| 3 | 347.89% | -58.32% |
| 4 | 346.44% | -58.42% |
| 5 | 319.65% | -60.55% |
| 6 | 305.03% | -61.83% |
| 7 | 309.08% | -62.35% |
| 8 | 332.22% | -60.37% |
| 9 | 320.13% | -60.92% |
| 10 | 471.77% | -44.27% |
| 11 | 437.95% | -46.73% |
| 12 | 376.85% | -54.73% |
| 13 | 553.71% | -22.27% |
| 14 | 605.95% | -13.60% |
| 15 | 771.03% | 7.92% |
| 16 | 923.04% | 37.95% |
| 17 | 227.48% | 59.60% |
| 18 | 342.00% | 85.61% |
| 19 | 973.89% | 164.33% |
| 20 | 1053.91% | 101.36% |
| 21 | 1194.02% | 87.41% |
| 22 | 1041.82% | 74.89% |

Table 3.3: Relative difference in latency times
*CISCA* under ACE+TAO, Orbix 2000 v2.0 and *SCARI* on Windows 2000 (from figure 3.3).

### 3.1.2 Latency times for different ORBs with client and server processes running on different machines

Latency tests were conducted across a network between a client running on RedHat 7.1 Linux 2.4.7-10 and a server running on a Sun Enterprise 450 running Solaris™ 2.8. Results are shown in figure 3.4 and table 3.4 shows the relative differences in latency times between the ORBs shown and *SCARI*.

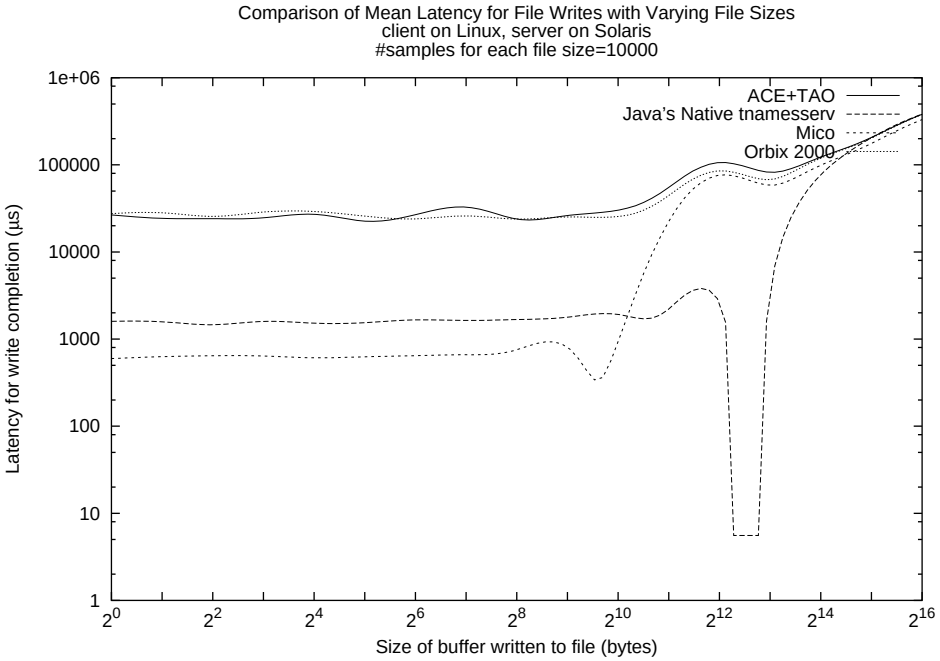


Figure 3.4: Comparison of latency times by ORB across a network

Figure 3.4 suggests that MICO is again more efficient than *SCARI* and the other ORBs for write operations involving buffers up to approximately $2^{10}$ bytes in size. Unexpectedly, Java™ and *SCARI* are the most efficient during write operations with buffers whose size is between $2^{10}$ and $2^{15}$ bytes. More research is required to determine the cause of the dramatic inverted spike observed for the *SCARI* latency times for buffer sizes between $2^{12}$ and $2^{13}$ bytes.

| **File Sizes** $(2^n)$ | **ACE+TAO** | **Mico** | **Orbix 2000** |
|---|---|---|---|
| 0 | -93.98% | 167.96% | -94.20% |
| 1 | -93.50% | 151.36% | -94.39% |
| 2 | -93.93% | 127.03% | -94.29% |
| 3 | -93.53% | 149.38% | -94.45% |
| 4 | -94.37% | 149.75% | -94.78% |
| 5 | -93.18% | 146.17% | -94.03% |
| 6 | -93.80% | 157.57% | -93.08% |
| 7 | -95.00% | 147.42% | -93.69% |
| 8 | -92.97% | 122.49% | -92.97% |
| 9 | -93.19% | 120.53% | -92.93% |
| 10 | -93.64% | 104.72% | -92.46% |
| 11 | -95.98% | -90.12% | -95.02% |
| 12 | -97.52% | -96.55% | -96.92% |
| 13 | -95.63% | -93.85% | -94.70% |
| 14 | -37.33% | -22.34% | -35.61% |
| 15 | -0.72% | 16.49% | 0.97% |
| 16 | 0.87% | 15.82% | 1.01% |

Table 3.4: Relative difference in latency times
*CISCA* under ACE+TAO, MICO, Orbix 2000 v2.0 and *SCARI* across a network (from figure 3.4).

### 3.1.3 Memory usage

The curves shown in figures 3.5 and 3.6 visually depict the resident and virtual memory size of the server processes during the tests which generated the results used in figure 3.1. Table 3.5 shows the relative difference in the resident and virtual memory sizes of the processes associated with the *CISCA* server and *SCARI* compared by buffer size.

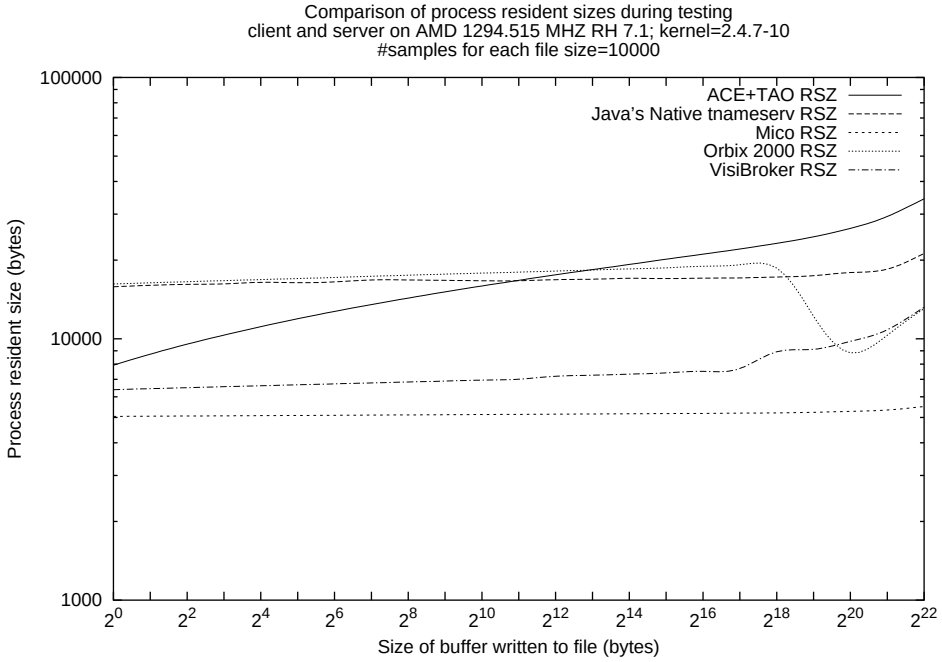


Figure 3.5: Comparison of process resident sizes by ORB on RedHat 7.1 Linux 2.4.7-10

Figures 3.5 and 3.6 both show that ACE+TAO increases in memory usage as the size of the buffer being passed to the write operation in *CISCA* increases. Ideal results would produce a perfectly flat line in both of these figures indicating that memory usage for the server process remains constant. If the results do not produce a flat line there should at least be a maximum amount of memory that the the process curve would be asymptotic to. The resident and virtual memory size results for ACE+TAO indicate that there is either a memory resource problem or that memory buffers are being handled in an inefficient manner.

As the curves in figures 3.5 and 3.6 move up and down the ORBs are busy allocating and deallocating memory. The process of memory management can be time intensive and lead to eventual problems with memory fragmentation[22] and excessive disk paging[22]. This results

in a degradation of performance which would eventual manifest itself in increased latency times on systems with constrained memory resources.

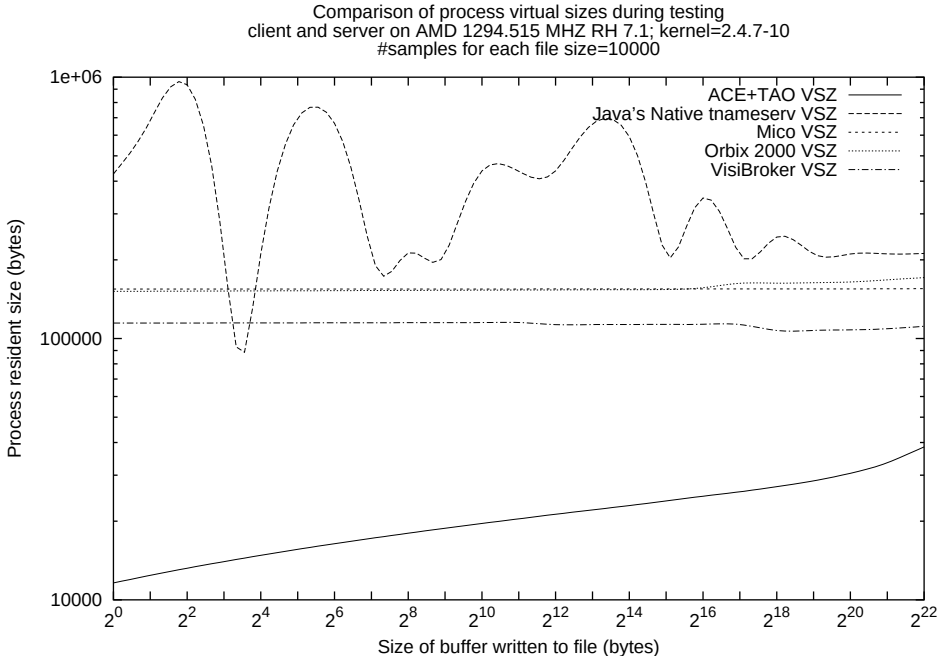


Figure 3.6: Comparison of process virtual memory sizes by ORB on RedHat 7.1 Linux 2.4.7-10

| **File Sizes** | **ACE+TAO** | | **Mico** | | **Orbix 2000** | |
|---|---|---|---|---|---|---|
| $(2^n)$ | rsz | vsz | rsz | vsz | rsz | vsz |
| 0 | 99.47% | 3571.73% | 213.25% | 175.56% | -2.48% | 181.27% |
| 1 | 83.43% | 5399.49% | 216.84% | 341.35% | -2.21% | 350.00% |
| 2 | 69.38% | 6977.40% | 219.00% | 504.76% | -2.30% | 516.03% |
| 3 | 57.04% | 1389.37% | 219.78% | 34.88% | -2.84% | 37.26% |
| 4 | 47.74% | 1322.28% | 223.68% | 36.17% | -2.41% | 38.44% |
| 5 | 37.39% | 4369.51% | 222.13% | 351.05% | -3.60% | 358.11% |
| 6 | 29.67% | 3958.57% | 223.82% | 330.45% | -3.86% | 336.75% |
| 7 | 24.23% | 1136.48% | 228.92% | 37.44% | -3.17% | 39.29% |
| 8 | 17.33% | 1082.17% | 228.43% | 37.49% | -4.10% | 39.18% |
| 9 | 10.68% | 1021.55% | 226.73% | 36.28% | -5.35% | 37.81% |
| 10 | 4.72% | 2132.91% | 225.07% | 182.92% | -6.50% | 185.83% |
| 11 | -0.37% | 2036.91% | 224.54% | 181.97% | -7.31% | 184.59% |
| 12 | -4.24% | 1959.84% | 227.11% | 183.06% | -7.30% | 185.41% |
| 13 | -8.13% | 2900.67% | 227.93% | 328.36% | -7.79% | 331.49% |
| 14 | -11.50% | 2480.56% | 229.91% | 282.81% | -7.96% | 285.20% |
| 15 | -15.71% | 774.31% | 228.48% | 35.06% | -9.03% | 35.75% |
| 16 | -19.05% | 1283.64% | 229.31% | 122.83% | -9.97% | 120.50% |
| 17 | -22.49% | 708.89% | 229.24% | 35.44% | -10.91% | 28.89% |
| 18 | -25.66% | 802.09% | 231.08% | 57.95% | -7.54% | 50.11% |
| 19 | -29.04% | 638.41% | 233.23% | 36.13% | 42.95% | 28.88% |
| 20 | -32.21% | 590.65% | 239.91% | 36.04% | 102.96% | 27.98% |
| 21 | -37.15% | 529.59% | 245.66% | 36.02% | 78.93% | 25.74% |
| 22 | -38.23% | 449.39% | 284.95% | 36.34% | 62.37% | 23.59% |

Table 3.5: Difference in resident and virtual memory sizes between the shown ORBs and *SCARI* shown in figure 3.5.

# Chapter 4

# Discussion

## 4.1 General Results

The results from chapter 3 demonstrate that a C++ implementation of the SCA is a viable alternative to *SCARI*. A C++ implementation of the SCA framework that tries to maintain the highest degree of portability and performance is not without its complications, however. Some careful work and planning needs to go into understanding both the ORB that is to be used, the compiler that is to be used to generate the application, and the environment in which the application will run. If each of these areas are well understood a developer would be able to deploy *CISCA* in any number of different configurations and situations.

## 4.2 Portability

Writing portable software means more than just writing software that can be compiled to run on more than one operating system. Portability, in the context of *CISCA*, means that software has to run on different operating systems with different orbs compiled by different compilers. This means that only features available in *all* ORBs could be used, no special libraries that were only available for Linux could be included, and that no special or magic features of certain compilers could be employed.

Initial development was confined solely to RedHat 7.1 Linux 2.4.7-10 with MICO 2.3.8, but constant attention was paid to making sure that it would be as easy as possible to port *CISCA* to different operating systems and different ORBs once it was ready. Tools like adjoin[25] and automake[25] made it possible to take the single code base that was created on RedHat 7.1 Linux 2.4.7-10 and with slight changes using `#ifdef` blocks build and run *CISCA* on Solaris® using MICO 2.3.8. Refer to appendix A for more details on how this process is implemented.

The decision to use MICO 2.3.8 as the initial development ORB was based on the fact that of all the ORBs that were to be used for testing, MICO 2.3.8 reported the lowest CORBA® standard compliance. Any features found and used with MICO would then theoretically exist in all of the other ORBs.

Once *CISCA* was running on Linux and Solaris® with MICO 2.3.8 it was a straight forward modification to add support for additional ORBs. Table 2.1 lists on which operating system and with which ORB *CISCA* was tested.

### 4.2.1 ORB Differences

*CISCA* was tested with four different orbs and during this testing process several minor differences between the ORBs were observed. This discussion will cover how the other ORBs were different from MICO and how those differences were accounted for.

Table 4.1 lists the vendor reported level of CORBA® compliance for each of the ORBs that were used during testing.

| **ORB** | **CORBA® compliance** | **Notes** |
|---|---|---|
| MICO 2.3.8 | 2.3 | Includes a full implementation of the CCM. |
| ACE+TAO 1.3 | 2.5 | |
| Orbix 2000 v2.0 | 2.5 | Support for version 3.0 of the CORBA® standard is included in some areas. |
| VisiBroker™ 5.2 | 2.5 | |
| Java™ 1.4.1 | 2.3 | According to Sun[14] the Java™ runtime used for testing only supports selected parts of the CORBA® 2.3 specification. [1] |

Table 4.1: Reported CORBA® compliance for ORBs used in development and testing

As table 4.1 shows each of the ORBs involved with this project should be compatible with MICO. That means any features used in *CISCA* that work with MICO should work with all the other ORBs shown in table 4.1 without majour modification.

For the most part this proved to be correct but there were several specific technical barriers encountered which prevented the code for *CISCA* from being compiled and run with all the ORBs straight out of the box. Three of the main issues encountered while porting *CISCA* to different ORBs are listed below.

```
142         /**
143          * Set up a POA for the File default servant.
144          */
145         CORBA::PolicyList pl;
146         pl.length(3);
147         pl[0] = poa->create_servant_retention_policy (PortableServer::NON_RETAIN);
148         pl[1] = poa->create_id_uniqueness_policy (PortableServer::MULTIPLE_ID);
149         pl[2] = poa->create_request_processing_policy(PortableServer::USE_DEFAULT_SERVANT);
150     #ifdef ORB_ORBIX2000
151         pl.length(4);
152         pl[3] = poa->create_id_assignment_policy (PortableServer::USER_ID);
153     #endif
```

Figure 4.1: Portion of the `sca_server_main.cpp` file

### Orbix 2000 v2.0 `PortableServer::_create_reference` workaround

For example, for some reason Orbix 2000 v2.0 was unable to create a reference to a CORBA® object when using a default servant. Whenever a file is created or opened by a `FileSystem` or `FileManager` a reference to a CORBA® object is created and returned to the caller. This is accomplished through the `PortableServer::_create_reference(...)` interface. With Orbix 2000 v2.0 a problem was encountered when a POA with a request processing policy of 'default servant' is asked to choose an id to assign to object references it creates. In other words, if the POA has the 'request_processing_policy' of 'default servant' and an id_assignment_policy of SYSTEM_ID (which is the default) an unknown exception is thrown when the PortableServer::_create_reference(...) is invoked.

To work around this *CISCA* will use the USER_ID id assignment policy and assign its own ids when creating object references. Figure 4.1 shows how policies are created when *CISCA* is built for Orbix 2000 v2.0. Figure 4.2 shows how *CISCA* creates object ids when working with Orbix 2000 v2.0. This is not a satisfactory implementation for a production system for many reasons including the need for an additional attribute of the `FileSystem` class called `_nFileNumber` that no other ORBs seem to require at all.

### VisiBroker™ 5.2 `resolve_init` workaround

VisiBroker™ 5.2 does not implement some of the more recent template type mappings from idl to C++ that have appeared in the CORBA® specification. As a result special calls to the `resolve_init(...)` template helper function[16] are made when *CISCA* is working with VisiBroker™ 5.2. An example of these calls is shown in figure 4.3.

```
178   ::CF::File_ptr FileSystem_impl::create( const char* fileName )
179       throw(
180           ::CORBA::SystemException,
181           ::CF::InvalidFileName,
182           ::CF::FileException)
183   {
184     if (exists(fileName)) {
185       CF::FileException e;
186       e.errorNumber = static_cast<CF::ErrorNumberType>(17);
187       e.msg = CORBA::string_dup("rats");//strerror(17));
188       throw e;
189     }
190
191     open_file_info_t _info;
192     _info._sFileName = fileName;
193
194     if ((_info._fp = ::fopen((_sBaseName + fileName).c_str(), "wb+")) == 0) {
195       CF::FileException e;
196       e.errorNumber = static_cast<CF::ErrorNumberType>(errno);
197       e.msg = CORBA::string_dup(::strerror(errno));
198       throw e;
199     }
200
201   #if defined (ORB_ORBIX2000)
202     char sFileId[100];
203   #if defined (HAVE_SNPRINTF)
204     snprintf(sFileId, 99, "%d", _nFileNumber++);
205   #elif defined (HAVE_SPRINTF)
206     sprintf(sFileId, "%d", _nFileNumber++);
207   #endif
208
209     PortableServer::ObjectId_var id(PortableServer::string_to_ObjectId(sFileId));
210     CORBA::Object_var obj = _file_poa->create_reference_with_id(id, "IDL:CF/File:1.0");
211   #else
212     CORBA::Object_var obj = _file_poa->create_reference("IDL:CF/File:1.0");
213   #endif
214
215     CF::File_var ref = CF::File::_narrow (obj);
216     assert (!CORBA::is_nil (ref));
217     PortableServer::ObjectId_var oid(_file_poa->reference_to_id(ref));
218     PortableServer::ObjectId_to_string(oid);
219     CORBA::String_var name = PortableServer::ObjectId_to_string(oid);
220     FileSystem_impl::_file_info[name.in()]  = _info;
221     return ref._retn();
222   }
```

Figure 4.2: Orbix2000 `PortableServer::create_reference(...)` workaround

```
126   #if defined(ORB_VISIBROKER)
127       poa = resolve_init<PortableServer::POA,
128                          PortableServer::POA_ptr,
129                          PortableServer::POA_var>(orb, "RootPOA");
130   #else
131       poa = resolve_init<PortableServer::POA>(orb, "RootPOA");
132   #endif
133       poaMgr = poa->the_POAManager();
134   #if defined(ORB_VISIBROKER)
135       nc = resolve_init<CosNaming::NamingContext,
136                         CosNaming::NamingContext_ptr,
137                         CosNaming::NamingContext_var>(orb, "NameService");
138   #else
139       nc = resolve_init<CosNaming::NamingContext>(orb, "NameService");
140   #endif
```

Figure 4.3: Portion of the `sca_server_main.cpp` file

**ACE+TAO exception stream insertion operators**

It was observed that after building ACE+TAO for the first time that `CORBA::Exception` classes could not be written to output streams using stream insertion operators. After some investigation it was discovered that the ACE+TAO code base is configured in an extremely complex way and needs to be told explicitly about what features are required. ACE+TAO does not include proper C++ exception handling by default. ACE+TAO was rebuilt enabling exceptions and the issue of being unable to write `CORBA::Exception` classes onto output streams was resolved.

### 4.2.2 Operating System Differences

The goal of *CISCA* was to produce a single tar-ball containing everything needed to compile with a minimum of external dependencies on any number of Unix™ and Windows® platforms. The development and testing effort during this project confined itself to RedHat Linux 7.1, Solaris® 2.8, and Windows® 2000. However, every effort was made to ensure that there was a *single* code base used to build the *CISCA* applications in keeping with the *Don't Repeat Yourself* or *DRY*[17] principle. The software can probably be ported to different platforms or made to support additional ORBs with only a little effort.

Differences in operating system APIs were accounted for using `#ifdef` blocks. In order to keep the main *CISCA* source files as uncluttered with operating system specific `#ifdef` blocks as possible a name space called `OSWrapper` containing generic functions for file manipulation and other file system operations was created. This means that instead of calling `opendir(...)` or `glob(...)` on different flavours of Unix™ or `_findfirst(...)` on Windows® directly from `FileManager` or `FileSystem`, calls to `OSWrapper::GetFileList(...)` are made instead.

### 4.2.3 Compiler & other tool differences

While working on a solution to the problem of making *CISCA* work with VisiBroker™ 5.2 on Solaris® it was discovered that g++ needs to be passed the command line option `-D_REENTRANT`. This option informs the compiler and linker that reentrant versions of the standard C API should be used. Such an option is not required on Linux and was initially omitted while working on Solaris®. Were this problem not corrected unexpected behaviour would have possibly affected *CISCA* when working with other ORBs on Solaris®.

The *CISCA* package depends on certain versions of the autoconf and automake tools. The versions that were initially used on Solaris® (autoconf version 2.13 and automake 1.5) would not configure *CISCA* correctly. Trials determined that while Linux requires autoconf version

2.13 and automake version 1.4-p5, Solaris® requires autoconf 2.53 and automake 1.5. Appendix A goes into more detail about reasons for this problem.

Some unexpected problems occurred with VisiBroker™ 5.2 on Windows® with Visual C++. Undefined behaviour was being exhibited when the *CISCA* server process was initializing the default servant POA. More investigation is required in order to determine the source of those problems, but as a result of those difficulties *CISCA* could not be tested on Windows®. It is possible that if a different request processing policy were selected that this undefined behaviour might not appear.

## 4.3 Additional Problems Encountered

There were some difficulties encountered with compiling and linking applications on Solaris® with VisiBroker™ 5.2. VisiBroker™ 5.2 does not ship link libraries that are compatible with gcc or g++. So while *CISCA* will compile properly on Solaris® with g++ and VisiBroker™ 5.2 it was not possible to generate executable binaries for this compiler/ORB/operating system combination.

Some unexpected problems occurred with VisiBroker™ 5.2 on Windows® with Visual C++. Undefined behaviour was being exhibited when the *CISCA* server process was initializing the default servant POA. More investigation is required in order to determine the source of those problems, but as a result of those difficulties *CISCA* could not be tested on Windows®. It is possible that if a different request processing policy were selected that this undefined behaviour might not appear.

As mentioned in the previous section, the *CISCA* package is quite specific about what versions of the autoconf and automake tools are present in order to configure and build properly. Some time was spent to track down a definitive list of tool versions that would work. Unfortunately the dependencies between the tools became quite complex and made compiling such a meaningful list difficult.

Finally, it was discovered that the idl file from the SCA makes use of symbol names that match those from the POSIX file `errno.h`[24]. With most compilers these symbol names are `#defined`. This problem results in the C preprocessor inserting invalid constants during the compilation phase. There does not appear to be an easy way around this problem. If the symbols in the `errno.h` file were defined as `const int`s in the `std::` name space there would be no problem. However, there did not appear to be a graceful and clean way to change these definitions as far as the compilers (both g++ 2.9* and MSVC++6.0) were concerned. It was therefore decided to prefix the error number symbol names in the idl with the string '`SCA_`'. This approach, while not ideal, ensured portability to all ORBs and operating systems tested.

There were some issues that appeared during the development of *CISCA* that prompted for the suggestion of additions to the SCA core framework.

## 4.4 Future work

There are many areas where this project could benefit from continued research.

- *Implement other modules from the SCA Core Framework.*
  It would be interesting to see how easy it would be to add additional components from the SCA core framework to *CISCA*. Of special interest is the question of whether additional components of the SCA core framework could be implemented without relying on any operating system or ORB specific features.

- *Improve ORB and POA integration.*
  It is less than ideal that special routines for accounting for minor differences in the way initial references to ORB objects are obtained and that special routines accounting for the Orbix 2000 v2.0 `PortableServer::_create_reference(...)` workaround are required. Is it possible to come up with a truly generic approach or is it possible to at least explain why such an approach is not possible?

- *Test and develop further the ability to manage distributed `CF::FileSystem` objects*
  A `CF::FileManager` should theoretically be able to mount a file system that is on a different machine in the network. In fact it should be possible for a `CF::FileManager` object to manager nothing but remote file systems. The testing in this project confined itself to situations where the `CF::FileManager` object and the `CF::FileSystem` objects that it managed were co-located on the same machine. Further testing into the possibility of managing remote file systems should be investigated.

- *Test inter-orb communication*
  Little or no testing was performed during the course of this project to determine how well ORBs were able to inter-operate. As far as theory goes it should be entirely possible for a *CISCA* client running on Linux in Orbix 2000 v2.0 to speak with an *CISCA* server running on Solaris® in MICO. Because of the change that was necessary to the error number symbol names to allow *CISCA* to compile under C++ it may not be possible for a *CISCA* client to communicate with a *SCARI* server. This fact should also be verified by additional testing.

# Appendix A

# The Software

## A.1 Building the binaries on Unix™

As mentioned earlier one of the intentions of *CISCA* was to prove that it was possible to produce an implementation of the SCA that was as portable as possible. To achieve this open-source project configuration tools like autoconf[9] and automake[12] were used for configuring the *CISCA* source for building on Unix™ platforms with different ORBs.

As a result building *CISCA* should be as straight forward as the standard 'configure and make' sequence used by many open-source projects.

```
$ tar -xzvf CISCA-1.0.tar.gz
$ cd CISCA-1.0
$ ./configure --with-orb=ORB [--with-boost-prefix=DIR] [--with-cppunit-dir=DIR]
$ make
```

Figure A.1: Example of build steps on Unix™

The '`--with-orb`' option must be specified. It is used to select which ORB *CISCA* is to be built with. The configure script selects certain orb-specific options and configures the *CISCA* source with the appropriate `#define`s and `#include` files. Table A.1 lists other custom options that have been added to the standard configure script.

The '`--with-boost-prefix=DIR`' option can be used if the boost library is installed on the host machine. The Boost[15] library is only used to provide a progress meter in the latency test program `sca_latency_test`. Note that the Boost library is not required and that the all programs will run without it.

The '`--with-cppunit-prefix=DIR`' option is used to specify the directory prefix that was used for building and installing the CppUnit[11] library. The CppUnit library is used for the unit-test program `sca_test`. This opting must be specified if the `sca_test` program is to be generated. Note that the `sca_server` and `sca_latency_test` programs do not have any dependencies on the CppUnit library itself. Only the `sca_test` program requires that the CppUnit library be installed.

| Configure Option | Used for ORB | Notes |
|---|---|---|
| `--with-orb=acetao` | ACE+TAO 1.3 | Must set the ACE_ROOT environment variable first. |
| `--with-orb=mico` | MICO | Must source the mico_setup.sh or mico_setup.csh script first. |
| `--with-orb=orbix2000` | Orbix 2000 v2.0 | Must set the IT_PRODUCT_DIR environment variable first. |
| `--with-orb=visibroker` | VisiBroker™ 5.2 | Must source the vbroker.sh or vbroker.csh script first. |
| `--with-boost-prefix=DIR` | *all* | Optional. DIR is root directory for where boost[15] is installed. |
| `--with-cppunit-prefix=DIR` | *all* | Optional. DIR is root directory for where CppUnit[11] is installed. |

Table A.1: *CISCA*-specific `configure` options

### A.1.1 Notes concerning ACE+TAO 1.3

The `ACE_ROOT` environment variable must be set to point to the full path to the `ACE_wrappers` directory *before* running the `configure` script in the *CISCA* source directory.

Figure A.2 shows the contents of a shell script that was used to build ACE+TAO on Solaris®.

```
#!/bin/sh
ACE_ROOT=/opt/orbs/ACE_wrappers; export ACE_ROOT
LD_LIBRARY_PATH=$ACE_ROOT/ace:/usr/local/lib:$LD_LIBRARY_PATH ; export LD_LIBRARY_PATH

cd $ACE_ROOT/ace ; make exceptions=1 static_libs_only=1
cd $ACE_ROOT/apps/gperf ; make exceptions=1 static_libs_only=1
cd $ACE_ROOT/TAO/tao ; make exceptions=1 static_libs_only=1 \
  TAO_ORBSVCS="Naming Naming_Service"
cd $ACE_ROOT/TAO/TAO_IDL ; make exceptions=1 static_libs_only=1 \
  TAO_ORBSVCS="Naming Naming_Service"
cd $ACE_ROOT/TAO/orbsvcs/orbsvcs ; make exceptions=1 \
  static_libs_only=1 TAO_ORBSVCS="Naming Naming_Service"
```

Figure A.2: Minimum ACE+TAO build steps on Solaris®

There are some issues using shared libraries generated with code compiled by `g++` and linked with Suns linker `ld`[18]. For this reason static libraries were used exclusively on Solaris®. The

`static_libs_only=1` option in the above script was removed when ACE+TAO was built on Linux.

### A.1.2 Notes concerning MICO

The environment variables `MICODIR` and `MICOVERSION` are used by the configure script when selecting the libraries that *CISCA* will link with as well as figuring out the location of MICO's `idl` compiler. These environment variables are set automatically by the `mico-setup.sh` or `mico-setup.csh` script.

If these environment variables are not set before running the configure script `configure` will report an error stating that the `MICODIR` variable is not set before terminating.

### A.1.3 Notes concerning Orbix 2000 v2.0

The `IT_PRODUCT_DIR` environment variable is used by the `configure` script to figure out where the Orbix 2000 v2.0 idl compiler, headers, and libraries are.

Prior to using the *CISCA* programs the shell script that is generated by the `${IT_PRODUCT_DIR}/orbix_art/2.0/bin/itconfigure` must be sourced. For example, if a configuration called `orbix2000` is created using `itconfigure`, the shell script `orbix2000_env` must be sourced before running `sca_server`.

### A.1.4 Notes concerning VisiBroker™ 5.2

The `VBROKERDIR` environment variable is used by the `configure` script to figure out where the VisiBroker™ 5.2 idl compiler, headers, and libraries are.

### A.1.5 Adding support for additional ORBs

It should be fairly easy to add support for additional ORBs without modifying the existing source code for *CISCA*. There is a short list of things that must be done in order to prepare the *CISCA* source base for use with an additional ORB. The steps required include:

- *figure out if any ORB specific configuration or environment scripts need to be sourced prior to using the ORB.*
  For example, MICO, Orbix 2000 v2.0, and VisiBroker™ 5.2 all provide environment scripts that can be sourced which set environment variables used to locate ORB specific components like the ORBs idl compiler, headers, and libraries. These environment variables can be used to simplify the next step of preparing the `configure` script.

```
284   AC_ARG_WITH(orb,
285     [  --with-orb=ORB         enable ORB support: ORB=[mico|acetao|orbix2000|visibroker]],
286     [orb_value=$withval],
287     [orb_value=""])
288
289   if test "x$orb_value" = "x"; then
290   AC_MSG_ERROR(You must choose a supported orb using the '--with-orb' option.)
291   elif test "x$orb_value" = "xmico"; then
292
293     #
294     # M I C O
295     #
296     if test "x$MICODIR" = "x"; then
297       AC_MSG_ERROR([The MICODIR environment variable is not set.  Did you source the mico-setup.sh?])
298     fi
299
300     if test "x$MICOVERSION" = "x"; then
301       AC_MSG_ERROR([The MICOVERSION environment variable is not set.  Did you source the mico-setup.sh?])
302     fi
303
304     AC_CHECK_FOR_ORB_HEADERS([$MICODIR/include], [mico/CosNaming.h])
305     AC_CHECK_FOR_ORB_LIBRARIES([$MICODIR/lib], [mico$MICOVERSION micocoss$MICOVERSION])
306
307     all_libraries="$all_libraries -ldl";
308
309     AC_CHECK_FOR_ORB_IDL([$MICODIR/bin/idl],
310                          [--c++-suffix cpp],
311                          [CF.cpp],
312                          [CF.cpp],
313                          [CF.h])
314
315     AC_DEFINE_UNQUOTED(ORB_MICO, 1)
316
317   elif test "x$orb_value" = "xorbix2000"; then
```

Figure A.3: Sample of ORB specific section in `configure`

- *add ORB specific section to the* `configure.in.in` *source file.*
  Figure A.3 shows the section of the included `configure.in.in` file which is used to configure the *CISCA* source base for use with MICO. The `if` blocks starting at line 296 and 300 simply test for expected environment variables (`MICODIR` and `MICOVERSION` in this case are both set by the `mico-setup.sh` script).

  The `AC_CHECK_FOR_ORB_HEADERS` macro is used to search for the ORB specific headers. The first argument is a white-space delineated list of directories to look in.[1] The second argument specifies the names of certain ORB headers which are included in the idl generated code. If the files can be found the macro automatically appends the appropriate include directory to the compiler flags.[2] If the files cannot be located the configure process halts with a message saying that the headers could not be located.

  The `AC_CHECK_FOR_ORB_LIBRARIES` macro is used to search for the ORB specific libraries. It works the same way as the `AC_CHECK_FOR_ORB_HEADERS` macro except that it will try to find `.so` and `.a` files based on the strings provided in the second argument.[3] Note that all of the libraries that need to be linked with the *CISCA* programs must be specified here. That means that the libraries providing the `CosNaming` and `PortableServer` code plus any supporting libraries must be included in the second argument to this macro.

  The `AC_CHECK_FOR_ORB_IDL` macro does several things. The first macro specifies the full path to the ORBs idl compiler. The second argument specifies any required options, including those required to generated skeleton and stub files with `.cpp` extensions. The third option specifies the name of all the server `.cpp` files. The fourth specifies the name of all the client `.cpp` files. The fifth and last argument specifies the names of any additional files generated by the idl compiler. This information will be used when a '`make clean`' is performed. If this macro succeeds it will cause dependencies to be created in the `Makefile`s so that the idl compiler will automatically generate these files when someone types '`make`' from the *CISCA* source directory.

---

[1] ORB headers may be dispersed across several different directories. Orbix 2000 v2.0 supports multiple compilers on one platform by separating the compiler specific headers and libraries in specific sub-directories. The generic headers that can be used by any compiler remain in a well known default directory. Refer to the `configure.in.in` script for more details.

[2] The `configure` variable `all_include` is used to hold on to the `-I<orb header dir>`.

[3] The `configure` variable `all_libraries` is used to hold onto the `-L<orb lib search dir>` in this case

```
19
20    #undef ORB_MICO
21    #undef ORB_ACETAO
22    #undef ORB_ORBIX2000
23    #undef ORB_VISIBROKER
24
```

Figure A.4: Portion of the `acconfig.h` file

```
36    #if defined (ORB_MICO)
37    /*-----------------------------------------------------------------------
38     * M I C O   2 . 3 . *
39     */
40
41    #include <CORBA.h>
42    #include <mico/CosNaming.h>
43    #include "sca/CF.h"
44
45    #elif defined (ORB_ACETAO)
```

Figure A.5: Sample of ORB specific section in `orb_server.h`

The `AC_DEFINE_UNQUOTED` macro on line 315 is used for the next step in configuring the *CISCA* source base for working with a new ORB.

- *add the names of the idl generated server header files to `orb_server.h` and the client header files to `orb_client.h`*

  If a new ORB specific section is added to the `configure.in.in` script the symbol that is defined by the `AC_DEFINE_UNQUOTED` macro must be added to the `acconfig.h` file near the section shown in figure A.4.

  The headers required by a server using *CISCA* should be included in the generic `orb_server.h` in appropriate `#ifdef` blocks. The client headers should be included in a similar manner

```
34    #if defined (ORB_MICO)
35    /*-----------------------------------------------------------------------
36     * M I C O   2 . 3 . *
37     */
38
39    #include <CORBA.h>
40    #include <mico/CosNaming.h>
41    #include "sca/CF.h"
42
43    #elif defined (ORB_ACETAO)
```

Figure A.6: Sample of ORB specific section in `orb_client.h`

in the `orb_client.h` header. These two headers are used to localize any ORB specific header files and other `#define`s that are required by ORB servers or clients.

These above steps are all that is required to add support for additional ORBs to *CISCA*. When the `configure.in.in`, `acconfig.h`, and the `orb_client.h` and `orb_server.h` files are modified as described all that should be required is the command `'make configure'` followed by the '`configure --with-orb=neworb; make`' sequence.

Of course, as mentioned earlier in section 4, some ORBs either do not adhere very tightly to the CORBA® specification or do not work as expected when tested.

## A.2 Building the binaries on Windows®

The method for building *CISCA* on Windows® is very similar to the one described for Unix™. The appropriate environment must be set prior to trying to build the software. The required environment variables are described in the previous section.

Once the tarball `CISCA-1.0.tar.gz` is expanded building *CISCA* should only requiring changing into the `CISCA-1.0` source directory and typing

```
nmake /f Makefile.win32 ORB="name"
```

where `ORB="name"` corresponds to one of the rows shown in table A.2. Note that the `Make-file`s on Windows® depend on an additional environment variable for finding the source files for *CISCA*: `SCA_HOME`. This variable should be set to point to the full path to the directory `CISCA-1.0`.

| Make Option | Used for ORB | Notes |
|---|---|---|
| `ORB=acetao` | ACE+TAO 1.3 | Must set the ACE_ROOT environment variable first. |
| `ORB=orbix2000` | Orbix 2000 v2.0 | Must set the IT_PRODUCT_DIR environment variable first. |
| `ORB=visibroker` | VisiBroker™ 5.2 | Must set the `VISIBROKERDIR` environment variable first. |

Table A.2: *CISCA* nmake options

## A.3 Compilers and other build tools

| | **compiler** | **autoconf** | **automake** | **make** |
|---|---|---|---|---|
| RedHat 7.1 Linux 2.4.7-10 | 2.96 | 2.13 | 1.4-p5 | GNU Make 3.79.1 |
| Sun Enterprise 450 running Solaris™ 2.8 | 2.95.2 | 2.53 | 1.5 | GNU Make 3.79.1 |
| Windows 2000 | MSVC++6.0 SP3 | *n/a* | *n/a* | nmake 6.00.8168.0 |

Table A.3: Compiler & tool versions

The *CISCA* package uses the autoconf[9] and automake[12] tool sets for configuring the source for builds on different operating systems and with different ORBs. Initial autoconf and automake development used scripts that were automatically generated by the KDevelop 2.1 ide.[5] The `configure.in.in` and other autoconf and automake files created by KDevelop 2.1 have very specific requirements for the versions of autoconf and automake that they support.

A certain amount of trial and error were required to find a satisfactory combination for the autoconf and automake tools for Solaris®.

Please take note that the more recent versions of KDevelop and the autoconf/automake tool sets do not suffer from these incompatibilities.

# Appendix B

# Software License

The software provided with this project is released under the GNU Public License. A copy of the GPL is contained in the distribution tar ball, `CISCA-1.0.tar.gz`.

# Appendix C

# Development & Testing Machines

*CISCA* was developed primarily on RedHat 7.1 Linux 2.4.7-10. The other machines used in development and testing are shown below.

## C.1 Linux

The primary development machine for this project was an I-Buddie XP DeskNote.

- *processor*
  AMD Mobile AMD Athlon™ XP 1500+ running at 1294.513 MHz
- *memory*
  512 MB
- *Operating System*
  Linux version 2.4.7-10 (bhcompile@stripples.devel.redhat.com) (gcc version 2.96 20000731 (Red Hat Linux 7.1 2.96-98)) #1 Thu Sep 6 16:46:36 EDT 2001

## C.2 Solaris®

Development and testing on Solaris® was conducted on a Sun Enterprise 450 running Solaris® 2.8.

- *processor*
  450 MHz

- *memory*
  2 GB

- *Operating System*
  SunOS mercury 5.8 Generic_108528-06 sun4u sparc SUNW,Ultra-4

## C.3 Windows®

All Windows® development was performed on an instance of Windows® 2000 Server running in a VMWare™ session installed on an I-Buddie XP DeskNote. Testing for Windows® was conducted using a seperate machine running Windows® 2000 Professional.

- *processor*
  667 MHz

- *memory*
  128 MB

- *Operating System*
  Windows 2000 Professional Version 5.0.2195 Build 2195

# Appendix D

# CD Contents

The CD that accompanies this report contains the following items:

- *ACETAO-1.3.tgz*
  This is the source distribution of the ACE+TAO 1.3 ORB that was used for a portion of the testing of CISCA. Instructions for building and running the ACE+TAO are contained in the distribution tar ball.

- *cppunit-1.8.0.tar.gz*
  This is the CppUnit test framework that was used for most of the regression testing of the CISCA software.

- *CISCA-1.0.tar.gz*
  This is the CISCA source tar ball itself. This includes all of the software that was developed for the *CISCA* project (in source form) along with the accompanying files neccessary to build the software on most flavours of Unix™ and Windows®.

- *mico-2.3.8.tgz*
  This is the distribution of MICO that was used for testing and most of the development of CISCA. Instructions for building, installing, and running MICO are contained in the distribution tar ball.

- *results.tgz*
  This file contains the testing results that were generated while evaluating the performance of CISCA.

- *thesis.pdf*
  This is a soft copy of this report (`thesis.pdf`).

- *cisca_ref.pdf*
- *cisca_ref.ps*
  PDF and Postcript formatted versions of documentation that was automatically generated from the source files of the CISCA package by the tool doxygen[2].
- *cisca_html.tgz*
  This is a tar ball of all the HTML documentation that was automatically extracted from the CISCA source by doxygen.

# Appendix E

# Results

This appendix contains a summary of the imperical results obtained during testing.

- *Table E.1*
  The statistical mean, median, mode, range, variance, and standard deviation of the results obtained while measuring the latency time for *SCARI* on RedHat 7.1 Linux 2.4.7-10.

- *Tables E.2, E.3, E.4, and E.5*
  The statistical mean, median, mode, range, variance, and standard deviation of the results obtained while measuring the latency time for *CISCA* on RedHat 7.1 Linux 2.4.7-10 with various ORBs.

- *Table E.6*
  The statistical mean, median, mode, range, variance, and standard deviation of the results obtained while measuring the latency time for *SCARI* on Sun Enterprise 450 running Solaris™ 2.8.

- *Tables E.7, E.8, and E.9*
  The statistical mean, median, mode, range, variance, and standard deviation of the results obtained while measuring the latency time for *CISCA* on Sun Enterprise 450 running Solaris™ 2.8 with various ORBs.

- *Table E.10*
  The statistical mean, median, mode, range, variance, and standard deviation of the results obtained while measuring the latency time for *SCARI* on Windows 2000.

- *Tables E.11 and E.12*
  The statistical mean, median, mode, range, variance, and standard deviation of the re-

sults obtained while measuring the latency time for *CISCA* on Windows 2000 with various ORBs.

- *Table E.13*
  The statistical mean, median, mode, range, variance, and standard deviation of the results obtained while measuring the latency time for *SCARI* with a client running on RedHat 7.1 Linux 2.4.7-10 and a server running on Sun Enterprise 450 running Solaris™ 2.8.

- *Tables E.14, E.15, and E.16*
  The statistical mean, median, mode, range, variance, and standard deviation of the results obtained while measuring the latency time for *CISCA* with a client running on RedHat 7.1 Linux 2.4.7-10 and a server running on Sun Enterprise 450 running Solaris™ 2.8.

| File Sizes($2^n$) | Mean | Median | Mode | Range | Variance | Standard Deviation |
|---|---|---|---|---|---|---|
| 0 | 2038.1084 | 1532.0778 | 1513.0043 | 275435.9245 | 27019137.8416 | 5197.9936 |
| 1 | 1919.8532 | 1507.9975 | 1500.0105 | 178467.0353 | 21307393.0491 | 4615.9932 |
| 2 | 1930.0698 | 1498.9376 | 1495.0037 | 102876.9016 | 22517638.7513 | 4745.2754 |
| 3 | 1918.0561 | 1497.0303 | 1495.0037 | 105980.0386 | 20877066.5407 | 4569.1429 |
| 4 | 2019.1437 | 1496.0766 | 1492.9771 | 209001.0643 | 31049905.1741 | 5572.2442 |
| 5 | 1925.7715 | 1501.0834 | 1502.9907 | 199608.9220 | 22400861.0596 | 4732.9548 |
| 6 | 1997.6029 | 1502.0370 | 1500.0105 | 171122.0741 | 26943176.5581 | 5190.6817 |
| 7 | 2012.9224 | 1497.0303 | 1484.9901 | 183101.0580 | 31892444.7415 | 5647.3396 |
| 8 | 2057.3538 | 1498.9376 | 1489.9969 | 209048.9864 | 34692168.6234 | 5890.0058 |
| 9 | 1970.8844 | 1515.0309 | 1514.4944 | 151226.0437 | 23188113.4093 | 4815.4038 |
| 10 | 2220.3990 | 1712.9183 | 1675.0097 | 119992.0177 | 21489254.5269 | 4635.6504 |
| 11 | 2328.5151 | 1783.9670 | 1778.0066 | 141530.9906 | 26177036.5116 | 5116.3499 |
| 12 | 2621.7239 | 1968.9798 | 1965.9996 | 141492.9628 | 34538604.8153 | 5876.9554 |
| 13 | 3150.7574 | 2387.0468 | 2341.9857 | 157088.8758 | 38947813.5008 | 6240.8183 |
| 14 | 4670.0179 | 3310.9188 | 3334.9991 | 116701.0069 | 62978754.3027 | 7935.9155 |
| 15 | 7296.5091 | 4884.0046 | 4830.0028 | 180112.0043 | 108537069.9921 | 10418.1126 |
| 16 | 11044.4814 | 8031.9643 | 7912.9934 | 229569.0775 | 213523929.4842 | 14612.4580 |
| 17 | 22946.2643 | 14176.0111 | 12449.0261 | 316592.9317 | 901492069.0224 | 30024.8575 |
| 18 | 67602.8372 | 25891.5424 | 25545.9944 | 481611.0134 | 6700435848.6055 | 81856.1900 |
| 19 | 145800.3260 | 119244.5159 | 153045.7437 | 539639.1153 | 7149595804.9934 | 84555.2825 |
| 20 | 237359.7925 | 235790.9679 | 189450.9792 | 915615.0818 | 8966819703.3087 | 94693.2928 |
| 21 | 304781.1516 | 305959.4631 | 232239.0079 | 717563.8676 | 5727703128.6790 | 75681.5904 |
| 22 | 534504.5483 | 513234.9730 | 493363.9765 | 1054041.0280 | 14985761448.0845 | 122416.3447 |

Table E.1: Statistical data for tests conducted with *SCARI* on RedHat 7.1 Linux 2.4.7-10

| File Sizes($2^n$) | Mean | Median | Mode | Range | Variance | Standard Deviation |
|---|---|---|---|---|---|---|
| 0 | 1134.2639 | 904.0000 | 902.0000 | 68215.0000 | 5490420.8861 | 2343.1647 |
| 1 | 1036.9921 | 904.0000 | 902.0000 | 80394.9999 | 3182437.4881 | 1783.9388 |
| 2 | 1015.3787 | 905.0000 | 902.0000 | 45299.9999 | 1962197.4699 | 1400.7846 |
| 3 | 1022.4644 | 906.0000 | 902.0000 | 46730.0000 | 2388979.5962 | 1545.6324 |
| 4 | 1034.4821 | 906.0000 | 902.0000 | 46048.0000 | 2908239.1463 | 1705.3560 |
| 5 | 1029.7002 | 904.0000 | 902.5000 | 44749.0000 | 2314463.8416 | 1521.3362 |
| 6 | 1027.7107 | 905.0000 | 902.0000 | 45623.0000 | 2398694.0520 | 1548.7718 |
| 7 | 1028.2229 | 902.0000 | 902.0000 | 44875.0000 | 2744795.8966 | 1656.7426 |
| 8 | 1015.7451 | 907.0000 | 909.0000 | 45629.9999 | 2139234.5914 | 1462.6122 |
| 9 | 1063.8221 | 929.0000 | 925.0000 | 53492.0000 | 2959530.3311 | 1720.3286 |
| 10 | 1072.1303 | 967.0000 | 964.0000 | 89390.0000 | 2693072.0935 | 1641.0582 |
| 11 | 1115.1832 | 988.0000 | 986.0000 | 46908.0000 | 2622664.5647 | 1619.4643 |
| 12 | 1309.7525 | 1163.0000 | 1162.0000 | 156118.0000 | 5045602.9349 | 2246.2420 |
| 13 | 1448.9706 | 1286.0000 | 1284.0000 | 50445.0000 | 3731590.6195 | 1931.7325 |
| 14 | 1725.5877 | 1530.0000 | 1528.0000 | 99275.0000 | 5132289.0455 | 2265.4556 |
| 15 | 2216.8309 | 1984.0000 | 1987.0000 | 153534.0000 | 7018569.7429 | 2649.2583 |
| 16 | 3154.2152 | 2894.0000 | 2882.0000 | 48377.0000 | 5511826.2279 | 2347.7279 |
| 17 | 5610.8939 | 5154.0000 | 5095.0000 | 47898.0000 | 10170350.3596 | 3189.0987 |
| 18 | 9370.0367 | 8658.0000 | 8661.0000 | 98798.0000 | 17631504.9925 | 4198.9886 |
| 19 | 16782.5326 | 15550.0000 | 15510.0000 | 60952.0000 | 29763576.9874 | 5455.6005 |
| 20 | 31588.3132 | 29269.0000 | 28634.0000 | 88612.0000 | 60591079.6868 | 7784.0272 |
| 21 | 61789.8258 | 57011.0000 | 55943.5000 | 188117.0000 | 153632735.3408 | 12394.8673 |
| 22 | 123944.7637 | 113823.0000 | 111852.8000 | 287001.0001 | 396051074.2379 | 19901.0320 |

Table E.2: Statistical data for tests conducted with ACE+TAO 1.3 on RedHat 7.1 Linux 2.4.7-10

| File Sizes($2^n$) | Mean | Median | Mode | Range | Variance | Standard Deviation |
|---|---|---|---|---|---|---|
| 0 | 629.9503 | 357.0000 | 358.0000 | 112144.0000 | 16895510.4453 | 4110.4149 |
| 1 | 694.6577 | 358.0000 | 359.0000 | 145673.0000 | 24862413.9309 | 4986.2224 |
| 2 | 672.3667 | 358.0000 | 359.0000 | 117786.0000 | 23030812.9077 | 4799.0429 |
| 3 | 687.7648 | 356.0000 | 357.0000 | 113332.0000 | 24849064.3608 | 4984.8836 |
| 4 | 635.1987 | 357.0000 | 358.0000 | 112599.0000 | 19737816.1154 | 4442.7262 |
| 5 | 593.2666 | 358.0000 | 359.0000 | 113845.0000 | 16195751.7556 | 4024.3946 |
| 6 | 723.0701 | 367.0000 | 367.0000 | 113639.9999 | 28142844.6799 | 5304.9830 |
| 7 | 634.7974 | 367.0000 | 367.0000 | 112110.0000 | 20536733.4894 | 4531.7473 |
| 8 | 665.1335 | 369.0000 | 367.0000 | 112327.0001 | 21268201.6114 | 4611.7460 |
| 9 | 667.9860 | 378.0000 | 375.0000 | 109585.0000 | 19816163.5938 | 4451.5350 |
| 10 | 674.9735 | 389.9999 | 390.0000 | 110226.9999 | 19683375.6711 | 4436.5951 |
| 11 | 747.5213 | 411.0000 | 412.0000 | 111949.0000 | 24260654.1820 | 4925.5106 |
| 12 | 887.5499 | 558.0000 | 559.0000 | 112465.0000 | 23108150.7954 | 4807.0938 |
| 13 | 984.5503 | 706.0000 | 704.0000 | 111737.0000 | 18108259.9028 | 4255.3801 |
| 14 | 1513.9922 | 1006.0000 | 1009.0000 | 113772.0000 | 37191241.0857 | 6098.4622 |
| 15 | 2257.9199 | 1601.0000 | 1594.0000 | 163660.0000 | 46081601.2323 | 6788.3430 |
| 16 | 4013.3348 | 2988.0000 | 2958.0000 | 164499.0000 | 73168050.5057 | 8553.8325 |
| 17 | 8005.3364 | 5792.0000 | 5765.0000 | 114804.0000 | 153951633.0127 | 12407.7247 |
| 18 | 14184.4730 | 10767.0000 | 10755.0000 | 118207.0000 | 241449228.7024 | 15538.6366 |
| 19 | 27025.5263 | 20107.0000 | 19960.0000 | 198855.0000 | 501355023.3273 | 22390.9585 |
| 20 | 54736.5106 | 39046.0000 | 38550.0000 | 307111.0000 | 1178052761.7720 | 34322.7732 |
| 21 | 114159.3554 | 77517.5000 | 76061.5000 | 386334.0000 | 2688462465.0187 | 51850.3854 |
| 22 | 228963.3909 | 256174.0000 | 147933.0000 | 457942.0000 | 5257239806.3505 | 72506.8259 |

Table E.3: Statistical data for tests conducted with MICO 2.3.8 on RedHat 7.1 Linux 2.4.7-10

| File Sizes($2^n$) | Mean | Median | Mode | Range | Variance | Standard Deviation |
|---|---|---|---|---|---|---|
| 0 | 3953.7385 | 2437.0000 | 2431.0000 | 231328.0000 | 211022129.4107 | 14526.6008 |
| 1 | 3935.2631 | 2436.0000 | 2431.0000 | 232510.0000 | 224666178.6425 | 14988.8685 |
| 2 | 3866.2103 | 2436.0000 | 2425.0000 | 210274.0000 | 208632992.4523 | 14444.1335 |
| 3 | 3987.1915 | 2438.0000 | 2423.0000 | 241128.0000 | 225172311.6199 | 15005.7426 |
| 4 | 3932.6378 | 2438.0000 | 2431.0000 | 232067.0000 | 220502415.0655 | 14849.3237 |
| 5 | 3904.1116 | 2440.0000 | 2431.0000 | 231731.0000 | 217080360.4615 | 14733.6472 |
| 6 | 3911.7047 | 2446.9999 | 2446.0000 | 238285.0000 | 214676208.8924 | 14651.8330 |
| 7 | 3844.1931 | 2454.0000 | 2445.0000 | 244491.0000 | 195859651.2935 | 13994.9866 |
| 8 | 3940.9018 | 2454.0000 | 2447.0000 | 210024.0000 | 220036348.5148 | 14833.6222 |
| 9 | 3731.3019 | 2458.0000 | 2453.5000 | 247739.0000 | 176560500.9729 | 13287.6070 |
| 10 | 3823.8334 | 2471.0000 | 2458.0000 | 230874.0000 | 185931198.7746 | 13635.6591 |
| 11 | 4042.3379 | 2500.0000 | 2492.0000 | 231400.0000 | 233407828.0298 | 15277.6905 |
| 12 | 4196.6343 | 2673.0000 | 2661.0000 | 222071.0000 | 224149967.0921 | 14971.6388 |
| 13 | 4487.2349 | 2813.0000 | 2798.0000 | 260194.0000 | 252627409.9130 | 15894.2571 |
| 14 | 5042.7909 | 3145.0000 | 3134.0000 | 250393.0000 | 288621549.9413 | 16988.8655 |
| 15 | 5775.9429 | 3609.0000 | 3593.0000 | 336547.0000 | 307605698.0703 | 17538.6915 |
| 16 | 6824.4596 | 4442.0000 | 4434.0000 | 250784.0000 | 321443125.5770 | 17928.8350 |
| 17 | 8993.6530 | 6322.0000 | 6286.0000 | 350900.0000 | 358007012.6238 | 18921.0732 |
| 18 | 12740.3465 | 9344.0000 | 9318.0000 | 302744.0000 | 441409480.0495 | 21009.7473 |
| 19 | 21357.8387 | 15336.5000 | 15137.0000 | 240576.0000 | 793834549.1446 | 28175.0696 |
| 20 | 36731.8402 | 27500.0000 | 27202.5000 | 287898.0000 | 868810257.4892 | 29475.5875 |
| 21 | 72064.4917 | 54168.5000 | 50927.0000 | 375925.9999 | 1825735530.7366 | 42728.6266 |
| 22 | 142318.8275 | 114971.0000 | 97615.5000 | 392900.0000 | 3952925047.0560 | 62872.2916 |

Table E.4: Statistical data for tests conducted with Orbix 2000 v2.0 on RedHat 7.1 Linux 2.4.7-10

| File Sizes($2^n$) | Mean | Median | Mode | Range | Variance | Standard Deviation |
|---|---|---|---|---|---|---|
| 0 | 2503.0061 | 735.0000 | 734.0000 | 303049.0000 | 223683598.8379 | 14956.0556 |
| 1 | 2241.6202 | 735.0000 | 734.0000 | 293249.0000 | 177646326.9977 | 13328.4030 |
| 2 | 2286.3034 | 736.0000 | 734.0000 | 282498.0000 | 193730099.5438 | 13918.6960 |
| 3 | 2142.5161 | 736.0000 | 734.0000 | 289863.0000 | 172331960.2890 | 13127.5268 |
| 4 | 2165.5209 | 736.0000 | 734.0000 | 264465.0000 | 170334965.5120 | 13051.2438 |
| 5 | 2226.5558 | 735.0000 | 734.0000 | 280599.0000 | 183317651.0703 | 13539.4849 |
| 6 | 2056.5057 | 742.0000 | 741.0000 | 289362.0000 | 153825754.0772 | 12402.6511 |
| 7 | 2240.3104 | 743.0000 | 742.0000 | 217270.0000 | 170841211.0740 | 13070.6240 |
| 8 | 2344.4900 | 747.0000 | 743.0000 | 279526.0000 | 187863608.8619 | 13706.3346 |
| 9 | 2351.9348 | 753.0000 | 750.0000 | 270669.9999 | 186047979.3632 | 13639.9406 |
| 10 | 2251.0830 | 762.0000 | 758.0000 | 278975.0000 | 170419430.4411 | 13054.4793 |
| 11 | 2423.8774 | 785.0000 | 786.0000 | 268785.0000 | 193398145.6235 | 13906.7662 |
| 12 | 2021.9802 | 1118.0000 | 1117.0000 | 283664.9999 | 101616433.3184 | 10080.4977 |
| 13 | 2367.8458 | 1219.0000 | 1216.0000 | 272516.0000 | 133731146.5686 | 11564.2184 |
| 14 | 2554.6072 | 1419.0000 | 1414.0000 | 277819.0000 | 138822648.7494 | 11782.3024 |
| 15 | 5365.6787 | 1816.0000 | 1812.0000 | 332527.0000 | 461130162.6014 | 21473.9415 |
| 16 | 5544.6086 | 2613.0000 | 2599.0000 | 343280.0000 | 379330508.8623 | 19476.4090 |
| 17 | 9434.4350 | 4544.0000 | 4510.0000 | 339135.0000 | 663766837.1138 | 25763.6728 |
| 18 | 15177.9252 | 7895.0000 | 7831.0000 | 482716.0000 | 1297781700.5064 | 36024.7373 |
| 19 | 34026.2335 | 14154.0000 | 13958.0000 | 411066.0000 | 2901605239.5097 | 53866.5503 |
| 20 | 59798.1367 | 27250.0000 | 26151.0000 | 460147.0000 | 5043330438.8947 | 71016.4096 |
| 21 | 125603.0564 | 71256.0000 | 51577.2500 | 459392.0000 | 9217652339.7478 | 96008.6056 |
| 22 | 258414.0056 | 270326.0000 | 102427.3000 | 643065.0000 | 19317080344.1575 | 138985.8998 |

Table E.5: Statistical data for tests conducted with VisiBroker™ 5.2 on RedHat 7.1 Linux 2.4.7-10

| File Sizes($2^n$) | Mean | Median | Mode | Range | Variance | Standard Deviation |
|---|---|---|---|---|---|---|
| 0 | 1741.6121 | 1234.0546 | 1212.0008 | 625825.0475 | 67260277.7433 | 8201.2364 |
| 1 | 1559.0725 | 1227.9749 | 1201.9873 | 620289.0873 | 48163498.3352 | 6939.9927 |
| 2 | 1471.5451 | 1225.9483 | 1199.0070 | 113240.1228 | 7114881.5351 | 2667.3735 |
| 3 | 1549.7700 | 1235.0082 | 1209.0206 | 621325.0160 | 47254489.9347 | 6874.1901 |
| 4 | 1583.1055 | 1238.9421 | 1199.0070 | 603967.0706 | 46497169.0733 | 6818.8833 |
| 5 | 1505.0310 | 1235.0082 | 1212.0008 | 98890.0661 | 8932005.2188 | 2988.6461 |
| 6 | 1443.3004 | 1232.9817 | 1206.9941 | 84077.0006 | 6951871.8198 | 2636.6403 |
| 7 | 1545.3250 | 1242.0416 | 1214.0274 | 651970.0289 | 50464702.3106 | 7103.8512 |
| 8 | 1493.5498 | 1240.0150 | 1217.0076 | 84432.0059 | 8367703.0262 | 2892.6982 |
| 9 | 1584.2553 | 1263.9761 | 1237.9885 | 706601.0237 | 58796249.3392 | 7667.8712 |
| 10 | 1900.3917 | 1542.9258 | 1525.9981 | 623358.9649 | 48686417.6566 | 6977.5653 |
| 11 | 2128.1737 | 1787.0665 | 1780.9868 | 552115.0827 | 39661467.3146 | 6297.7351 |
| 12 | 2792.6866 | 2315.0444 | 2297.9975 | 626028.1801 | 55438495.3317 | 7445.7031 |
| 13 | 3928.2934 | 3383.9941 | 3322.0053 | 663977.9806 | 66390574.5676 | 8148.0411 |
| 14 | 6797.4540 | 5833.9834 | 5769.0144 | 627335.0716 | 120058695.0579 | 10957.1299 |
| 15 | 12508.4085 | 10722.9948 | 10685.5035 | 712056.0408 | 266500867.8879 | 16324.8543 |
| 16 | 24344.1903 | 21095.9911 | 21003.0079 | 789124.0120 | 464860116.3340 | 21560.6149 |
| 17 | 48047.4720 | 41051.4474 | 40201.4852 | 769986.9871 | 1221743022.5974 | 34953.4408 |
| 18 | 95526.5177 | 83969.4738 | 78145.0272 | 862762.0935 | 2566229309.4295 | 50657.9639 |
| 19 | 198690.5094 | 171857.9531 | 166028.0228 | 941870.9278 | 5789152677.4443 | 76086.4816 |
| 20 | 413748.4029 | 412464.9763 | 295626.0443 | 1202801.9428 | 11524122158.1643 | 107350.4642 |
| 21 | 820133.5908 | 810931.0269 | 838148.4747 | 1480260.1337 | 19672403242.7807 | 140258.3446 |
| 22 | 1584239.1262 | 1538995.5044 | 1574931.9792 | 2449240.9229 | 48826926822.0137 | 220968.1579 |

Table E.6: Statistical data for tests conducted with *SCARI* on Sun Enterprise 450 running Solaris™ 2.8

| File Sizes($2^n$) | Mean | Median | Mode | Range | Variance | Standard Deviation |
|---|---|---|---|---|---|---|
| 0 | 750.0201 | 673.0556 | 663.9957 | 50562.9778 | 1690178.7158 | 1300.0687 |
| 1 | 758.1334 | 674.0093 | 666.0223 | 80156.9223 | 2162721.1520 | 1470.6193 |
| 2 | 735.4034 | 672.9364 | 666.0223 | 51589.0121 | 1219279.4320 | 1104.2099 |
| 3 | 743.3636 | 673.8901 | 666.0223 | 96897.0060 | 2242300.2035 | 1497.4312 |
| 4 | 775.1384 | 673.8901 | 666.0223 | 51025.9867 | 1914166.1574 | 1383.5339 |
| 5 | 765.7437 | 674.0093 | 666.0223 | 55926.9190 | 1490730.1501 | 1220.9546 |
| 6 | 774.6912 | 674.9630 | 666.9760 | 98505.9738 | 3755607.9051 | 1937.9391 |
| 7 | 762.0507 | 676.9896 | 669.0025 | 86536.8843 | 2198597.3414 | 1482.7668 |
| 8 | 777.2451 | 679.9698 | 671.9828 | 142933.9647 | 4334478.6476 | 2081.9411 |
| 9 | 816.0919 | 696.0630 | 689.9834 | 115417.0036 | 6529965.5392 | 2555.3797 |
| 10 | 836.1983 | 746.0117 | 738.9784 | 101196.1699 | 3548353.0446 | 1883.7073 |
| 11 | 841.6975 | 762.1050 | 753.9988 | 52007.0791 | 1675042.0924 | 1294.2342 |
| 12 | 865.8715 | 789.0463 | 779.9864 | 42830.9441 | 1615204.1121 | 1270.9068 |
| 13 | 980.1761 | 864.9826 | 856.9956 | 123349.9050 | 4755742.5735 | 2180.7665 |
| 14 | 1280.6023 | 1118.8984 | 1109.0040 | 531162.9772 | 31242316.7924 | 5589.4827 |
| 15 | 1742.5659 | 1574.9931 | 1572.0129 | 531157.9704 | 30961924.8827 | 5564.3441 |
| 16 | 2855.8468 | 2447.9628 | 2431.9887 | 531227.9463 | 109020946.8674 | 10441.3096 |
| 17 | 17202.8697 | 16430.9740 | 16433.0006 | 591421.0081 | 148710427.2777 | 12194.6885 |
| 18 | 20229.7577 | 18517.9710 | 18522.0242 | 590355.9923 | 280355505.5099 | 16743.8199 |
| 19 | 65715.2433 | 61787.9629 | 61771.9889 | 712700.9630 | 717073752.3388 | 26778.2328 |
| 20 | 103056.5538 | 94619.9894 | 94612.0024 | 701159.0004 | 1440125453.9233 | 37948.9849 |
| 21 | 165948.0212 | 151444.9120 | 151486.9928 | 785310.9837 | 3329842315.4094 | 57704.7859 |
| 22 | 343353.5564 | 314253.9859 | 308948.9937 | 801301.9562 | 5988486072.1811 | 77385.3091 |

Table E.7: Statistical data for tests conducted with ACE+TAO 1.3 on Sun Enterprise 450 running Solaris™ 2.8

| File Sizes($2^n$) | Mean | Median | Mode | Range | Variance | Standard Deviation |
|---|---|---|---|---|---|---|
| 0 | 704.6977 | 620.0075 | 617.0273 | 115997.9105 | 4427838.3346 | 2104.2429 |
| 1 | 713.9346 | 620.9612 | 617.9810 | 107613.0867 | 5347548.3632 | 2312.4767 |
| 2 | 707.7808 | 620.9612 | 617.9810 | 91979.0268 | 3673763.5333 | 1916.7064 |
| 3 | 751.7867 | 620.0075 | 617.9810 | 566644.0725 | 35228009.6347 | 5935.3188 |
| 4 | 746.3537 | 620.9612 | 617.9810 | 119106.0543 | 7475912.6040 | 2734.2115 |
| 5 | 712.0239 | 622.0341 | 620.0075 | 109583.0202 | 4156990.1988 | 2038.8698 |
| 6 | 719.1934 | 631.9284 | 627.9945 | 114740.9678 | 2961131.3020 | 1720.7938 |
| 7 | 774.7453 | 634.0742 | 630.9748 | 568082.9287 | 36224442.5261 | 6018.6745 |
| 8 | 750.8760 | 643.9686 | 640.9883 | 139091.9685 | 7244495.4290 | 2691.5600 |
| 9 | 750.1671 | 653.0285 | 651.0019 | 107913.9709 | 4796714.5600 | 2190.1403 |
| 10 | 773.7040 | 674.0093 | 674.0093 | 99028.1105 | 4245758.0530 | 2060.5237 |
| 11 | 779.8256 | 692.0099 | 692.0099 | 91665.1487 | 3965160.8947 | 1991.2712 |
| 12 | 813.0132 | 741.9586 | 741.0049 | 100324.0347 | 3040231.9502 | 1743.6261 |
| 13 | 982.0049 | 863.0753 | 856.9956 | 109351.8734 | 7175753.0568 | 2678.7596 |
| 14 | 1458.4707 | 1188.0398 | 1183.9867 | 488196.0154 | 39372233.4439 | 6274.7298 |
| 15 | 2168.8099 | 1801.9676 | 1796.0072 | 637725.9493 | 117690259.3371 | 10848.5142 |
| 16 | 3633.4047 | 3089.9048 | 3057.0030 | 486359.9539 | 54740721.2300 | 7398.6973 |
| 17 | 19375.0751 | 17776.9661 | 17771.0056 | 550264.0009 | 226231253.4465 | 15040.9858 |
| 18 | 23114.0113 | 19919.5147 | 19804.0009 | 574007.1535 | 384079068.9739 | 19597.9353 |
| 19 | 67440.8191 | 62751.8892 | 59777.4982 | 715101.9573 | 923616309.2550 | 30391.0564 |
| 20 | 119253.0917 | 104612.1120 | 104598.9990 | 716395.0205 | 2039149937.7330 | 45156.9478 |
| 21 | 204419.4537 | 177003.5028 | 171241.9987 | 983167.1715 | 4740879165.8074 | 68854.0425 |
| 22 | 437996.2261 | 395848.9895 | 358054.9955 | 1220790.9822 | 14399308947.3160 | 119997.1206 |

Table E.8: Statistical data for tests conducted with MICO 2.3.8 on Sun Enterprise 450 running Solaris™ 2.8

| File Sizes($2^n$) | Mean | Median | Mode | Range | Variance | Standard Deviation |
|---|---|---|---|---|---|---|
| 0 | 2835.9155 | 2544.9991 | 2537.0121 | 99757.9098 | 12299255.0740 | 3507.0294 |
| 1 | 2879.6357 | 2552.0325 | 2534.9855 | 116641.0446 | 17297900.2679 | 4159.0744 |
| 2 | 2957.0575 | 2545.9528 | 2506.0177 | 677137.0173 | 88170540.5102 | 9389.9170 |
| 3 | 2792.8792 | 2547.0257 | 2503.9911 | 104736.0897 | 11314718.7995 | 3363.7358 |
| 4 | 2849.0325 | 2544.9991 | 2503.9911 | 108125.9251 | 14858228.1399 | 3854.6372 |
| 5 | 2790.0239 | 2550.0059 | 2508.9979 | 100836.0386 | 11262341.7062 | 3355.9413 |
| 6 | 2807.3822 | 2544.9991 | 2508.9979 | 109472.9900 | 12191545.0298 | 3491.6393 |
| 7 | 2842.1034 | 2555.9664 | 2521.9917 | 123606.0858 | 13474494.1908 | 3670.7621 |
| 8 | 2879.4223 | 2557.0393 | 2514.0047 | 460161.9244 | 34948209.8887 | 5911.7011 |
| 9 | 2872.6614 | 2579.0930 | 2537.0121 | 114639.9975 | 13801953.2612 | 3715.0980 |
| 10 | 2827.7870 | 2575.9935 | 2539.9923 | 92064.9767 | 8932378.7841 | 2988.7085 |
| 11 | 2940.0153 | 2593.0405 | 2550.0059 | 719538.9271 | 64499545.4787 | 8031.1609 |
| 12 | 2922.4595 | 2626.0614 | 2583.9806 | 102115.9887 | 15025297.4751 | 3876.2479 |
| 13 | 3195.6866 | 2743.9594 | 2714.9916 | 729811.0723 | 123149051.8844 | 11097.2542 |
| 14 | 3589.8878 | 3262.5198 | 3214.0017 | 517736.9118 | 53717309.8715 | 7329.2094 |
| 15 | 4380.5065 | 3955.0066 | 3893.4946 | 516550.8986 | 45086274.5233 | 6714.6314 |
| 16 | 5917.1233 | 5362.0338 | 5313.9925 | 516098.0225 | 50601554.0305 | 7113.4769 |
| 17 | 20166.9233 | 16860.9619 | 16824.0070 | 634211.0634 | 216798907.3575 | 14724.0928 |
| 18 | 27193.8914 | 24883.0318 | 24857.9979 | 705903.8877 | 373250334.4679 | 19319.6877 |
| 19 | 67589.2351 | 61793.4465 | 61673.9988 | 691144.9432 | 449470055.1221 | 21200.7088 |
| 20 | 112985.1459 | 105278.0151 | 104492.0087 | 734011.8885 | 1179051455.1440 | 34337.3187 |
| 21 | 217793.1349 | 203180.4919 | 202911.4962 | 834197.0444 | 2573756174.8033 | 50732.2006 |
| 22 | 427593.1933 | 397508.5616 | 387452.0063 | 1681877.9707 | 6795796012.1079 | 82436.6182 |

Table E.9: Statistical data for tests conducted with Orbix 2000 v2.0 on Sun Enterprise 450 running Solaris™ 2.8

| File Sizes($2^n$) | Mean | Median | Mode | Range | Variance | Standard Deviation |
|---|---|---|---|---|---|---|
| 0 | 2556.4294 | 2314.4003 | 2168.7114 | 149655.8920 | 4364386.0849 | 2089.1113 |
| 1 | 2405.5258 | 2192.4574 | 2156.9781 | 52992.7686 | 1116536.1602 | 1056.6627 |
| 2 | 2467.2392 | 2236.7368 | 2165.0796 | 45289.2756 | 968620.9843 | 984.1854 |
| 3 | 2451.6775 | 2238.9717 | 2163.4034 | 9216.5345 | 567159.3829 | 753.0998 |
| 4 | 2456.3113 | 2224.8638 | 2165.0796 | 72688.2886 | 1579472.2196 | 1256.7706 |
| 5 | 2320.9601 | 2177.0923 | 1688.7621 | 30990.8103 | 518644.7301 | 720.1699 |
| 6 | 2237.9159 | 2130.0193 | 1692.3939 | 49916.1206 | 940875.8519 | 969.9876 |
| 7 | 2206.1412 | 1991.8733 | 1702.7304 | 28059.1528 | 426592.4468 | 653.1404 |
| 8 | 2349.4382 | 2185.1939 | 1714.7431 | 36038.3792 | 682247.7986 | 825.9829 |
| 9 | 2305.0231 | 2196.0892 | 1768.1018 | 76873.7367 | 1163791.6373 | 1078.7917 |
| 10 | 3278.8467 | 3216.0512 | 3377.3380 | 40540.6274 | 1342876.4685 | 1158.8255 |
| 11 | 3202.2283 | 3128.8893 | 2278.2225 | 63903.9319 | 1494598.5021 | 1222.5377 |
| 12 | 3409.3351 | 3320.2544 | 3252.3687 | 41366.9894 | 1048113.0237 | 1023.7739 |
| 13 | 5735.8246 | 5645.8293 | 5630.6039 | 122877.3489 | 4256303.0086 | 2063.0810 |
| 14 | 6997.9637 | 6691.4929 | 6463.1119 | 70735.8058 | 2965983.9446 | 1722.2032 |
| 15 | 10900.0307 | 10497.4236 | 10075.3029 | 418040.0023 | 24334191.9459 | 4932.9699 |
| 16 | 23632.9005 | 20996.8027 | 20193.8275 | 1124444.3079 | 421940230.0743 | 20541.1838 |
| 17 | 42741.1674 | 33903.1916 | 32939.8010 | 736549.7570 | 1049624019.9805 | 32397.9015 |
| 18 | 74365.5139 | 52409.0352 | 49205.2761 | 726610.2256 | 2399997950.9884 | 48989.7739 |
| 19 | 232462.5409 | 229851.0006 | 227796.4685 | 1097245.3203 | 623972647.0653 | 24979.4445 |
| 20 | 380035.9825 | 378813.6481 | 374784.7841 | 825122.4667 | 2184633410.9588 | 46740.0622 |
| 21 | 737427.6320 | 718257.9071 | 709867.0851 | 901827.7463 | 6844082595.9771 | 82728.9707 |
| 22 | 1415225.8782 | 1398516.7109 | 1407331.0993 | 1403405.4608 | 12840598742.8637 | 113316.3657 |

Table E.10: Statistical data for tests conducted with *SCARI* on Windows 2000

| File Sizes($2^n$) | Mean | Median | Mode | Range | Variance | Standard Deviation |
|---|---|---|---|---|---|---|
| 0 | 543.0596 | 522.4686 | 519.7960 | 14703.6006 | 47951.1894 | 218.9776 |
| 1 | 550.2847 | 554.6096 | 561.7194 | 5658.7918 | 9494.0907 | 97.4376 |
| 2 | 546.4635 | 549.0642 | 518.2782 | 3886.1643 | 11526.7160 | 107.3625 |
| 3 | 547.3883 | 549.4274 | 561.0769 | 5680.0515 | 13387.7130 | 115.7053 |
| 4 | 550.1948 | 555.0147 | 560.3170 | 5669.8826 | 12944.7470 | 113.7750 |
| 5 | 553.0723 | 556.9423 | 560.4902 | 5546.1239 | 16387.3897 | 128.0132 |
| 6 | 552.5295 | 555.9925 | 559.8198 | 6655.5945 | 20897.4920 | 144.5596 |
| 7 | 539.2925 | 526.3937 | 518.6972 | 5172.2216 | 16191.0312 | 127.2440 |
| 8 | 543.5760 | 523.8934 | 517.9150 | 3971.5103 | 14845.9345 | 121.8439 |
| 9 | 548.6393 | 544.9855 | 526.5753 | 5063.9676 | 19322.9278 | 139.0069 |
| 10 | 573.4586 | 574.0394 | 578.6490 | 6710.7970 | 21215.4575 | 145.6553 |
| 11 | 595.2602 | 585.0883 | 587.1137 | 6731.9450 | 28546.5028 | 168.9571 |
| 12 | 714.9687 | 688.7048 | 698.0077 | 6597.6262 | 37950.4260 | 194.8087 |
| 13 | 877.4256 | 818.5118 | 707.8833 | 86528.9304 | 1430240.2805 | 1195.9265 |
| 14 | 991.2844 | 991.0617 | 1020.6325 | 9633.4590 | 82479.3325 | 287.1922 |
| 15 | 1251.3908 | 1222.4598 | 1247.1278 | 31080.6820 | 231895.8852 | 481.5557 |
| 16 | 2310.0671 | 2161.2523 | 2341.7781 | 34492.5123 | 996322.3982 | 998.1595 |
| 17 | 13051.6607 | 12717.2448 | 12760.3648 | 71935.8466 | 8864818.1276 | 2977.3844 |
| 18 | 16824.9045 | 21754.6643 | 22531.6439 | 65780.7906 | 52596629.1585 | 7252.3534 |
| 19 | 21646.8572 | 23120.4664 | 23264.3255 | 59598.3288 | 17795753.4691 | 4218.5013 |
| 20 | 32934.7225 | 33046.7699 | 33390.9198 | 97315.0219 | 30270666.1283 | 5501.8784 |
| 21 | 56987.1405 | 55937.2630 | 55783.9753 | 100628.2363 | 73351426.3438 | 8564.5447 |
| 22 | 123944.7637 | 113823.0000 | 111852.8000 | 287001.0001 | 396051074.2379 | 19901.0320 |

Table E.11: Statistical data for tests conducted with ACE+TAO 1.3 on Windows 2000

| File Sizes($2^n$) | Mean | Median | Mode | Range | Variance | Standard Deviation |
|---|---|---|---|---|---|---|
| 0 | 8684.8157 | 5797.6640 | 5815.8230 | 227750.4540 | 325234088.9654 | 18034.2477 |
| 1 | 5885.6652 | 5783.9750 | 5815.8230 | 67401.3040 | 1508472.8313 | 1228.1990 |
| 2 | 5885.5623 | 5788.7250 | 5768.8900 | 69335.6280 | 1986409.5752 | 1409.4004 |
| 3 | 5882.4071 | 5785.3720 | 5773.0803 | 68342.4850 | 1843678.8655 | 1357.8214 |
| 4 | 5907.5953 | 5792.3560 | 5765.2580 | 71215.1960 | 2238086.2953 | 1496.0235 |
| 5 | 5883.2173 | 5760.2290 | 5731.1750 | 74521.2030 | 3040898.0124 | 1743.8171 |
| 6 | 5862.3003 | 5758.2740 | 5730.6170 | 65164.7070 | 1856135.3744 | 1362.4006 |
| 7 | 5859.7398 | 5750.7310 | 5726.1470 | 70773.2410 | 2323265.9639 | 1524.2263 |
| 8 | 5928.4533 | 5818.8960 | 5849.0675 | 71574.1810 | 1622008.4688 | 1273.5810 |
| 9 | 5897.7390 | 5790.6800 | 5843.2010 | 71323.8690 | 2167563.8812 | 1472.2649 |
| 10 | 5883.1416 | 5798.2230 | 5835.5180 | 76798.8670 | 2724098.0619 | 1650.4842 |
| 11 | 6011.6421 | 5899.7720 | 5947.1250 | 226165.0570 | 9209153.5242 | 3034.6587 |
| 12 | 7530.8096 | 7231.9250 | 7136.1020 | 167967.4380 | 9410106.6693 | 3067.5897 |
| 13 | 7378.8532 | 7272.0135 | 7224.3820 | 77544.4920 | 3153548.2384 | 1775.8233 |
| 14 | 8099.2318 | 7830.0455 | 7243.6580 | 77928.3400 | 6492385.1532 | 2548.0159 |
| 15 | 10100.1410 | 10079.7730 | 10055.7475 | 82188.9380 | 6498200.9081 | 2549.1569 |
| 16 | 17131.1383 | 15861.5130 | 15001.5474 | 428906.1880 | 48505326.1517 | 6964.5765 |
| 17 | 26780.1432 | 24968.1175 | 26180.1430 | 390157.6880 | 137454741.9273 | 11724.1094 |
| 18 | 40066.1720 | 38150.2395 | 36295.8840 | 922854.1620 | 287708134.3266 | 16961.9614 |
| 19 | 87944.1600 | 69343.1710 | 69134.4850 | 889084.5060 | 5390299274.6803 | 73418.6575 |
| 20 | 188738.5196 | 171383.2345 | 150694.5717 | 1319657.6660 | 5139827067.8545 | 71692.5873 |
| 21 | 393474.0817 | 391377.9545 | 389183.5596 | 846090.2150 | 4496989802.1076 | 67059.5989 |
| 22 | 809189.7778 | 804202.6295 | 810027.6412 | 1632148.5000 | 17717581858.1006 | 133107.4072 |

Table E.12: Statistical data for tests conducted with Orbix 2000 v2.0 on Windows 2000

| File Sizes($2^n$) | Mean | Median | Mode | Range | Variance | Standard Deviation |
|---|---|---|---|---|---|---|
| 0 | 1599.8300 | 1200.0799 | 1134.9916 | 164263.9637 | 14705000.7398 | 3834.7100 |
| 1 | 1578.2812 | 1153.9459 | 1137.0182 | 171276.9270 | 21876973.5440 | 4677.2827 |
| 2 | 1460.9367 | 1134.0380 | 1127.0046 | 394579.0529 | 37828468.7015 | 6150.4852 |
| 3 | 1593.3416 | 1147.0318 | 1132.0114 | 651900.0530 | 67469023.8383 | 8213.9530 |
| 4 | 1523.0086 | 1141.9058 | 1121.9978 | 404405.8323 | 26690083.2533 | 5166.2446 |
| 5 | 1540.9695 | 1155.0188 | 1127.0046 | 95731.9736 | 10592790.3003 | 3254.6567 |
| 6 | 1660.1252 | 1164.9728 | 1145.0052 | 692602.9921 | 75385205.0479 | 8682.4654 |
| 7 | 1635.7467 | 1175.9996 | 1163.0058 | 688773.9897 | 97900974.0587 | 9894.4921 |
| 8 | 1676.5504 | 1213.0737 | 1194.0002 | 651695.0130 | 85771989.9836 | 9261.3169 |
| 9 | 1784.1552 | 1296.0434 | 1278.9965 | 651317.0004 | 86585921.6355 | 9305.1556 |
| 10 | 1921.1252 | 1559.9728 | 1536.0117 | 392647.0280 | 25515005.4023 | 5051.2380 |
| 11 | 2215.2668 | 1678.9436 | 1657.0091 | 579302.0725 | 79932790.2710 | 8940.5140 |
| 12 | 2630.7568 | 1987.9341 | 1960.9928 | 903535.9621 | 149415303.7029 | 12223.5553 |
| 13 | 3584.7928 | 2622.0083 | 2557.9929 | 890997.8867 | 192951959.5217 | 13890.7149 |
| 14 | 76992.9815 | 5789.4588 | 4317.9989 | 31836715.9367 | 121980680230.2285 | 349257.3267 |
| 15 | 204974.0324 | 28170.5260 | 6433.0101 | 61231714.9639 | 1470171889534.2925 | 1212506.4493 |
| 16 | 383507.7264 | 247335.5532 | 12784.9579 | 63860027.0748 | 1845516144647.6792 | 1358497.7529 |

Table E.13: Statistical data for networked test conducted with *SCARI*

| File Sizes($2^n$) | Mean | Median | Mode | Range | Variance | Standard Deviation |
|---|---|---|---|---|---|---|
| 0 | 26567.3202 | 14448.0000 | 14267.0000 | 13161391.0000 | 23626777136.7894 | 153710.0424 |
| 1 | 24299.0816 | 14365.0000 | 14771.0000 | 5016067.0000 | 8500936042.3248 | 92200.5208 |
| 2 | 24086.9984 | 14351.0000 | 14166.0000 | 5063028.0001 | 8970223483.5167 | 94711.2638 |
| 3 | 24643.7278 | 14416.0000 | 14158.2500 | 5016359.0001 | 8396655863.8912 | 91633.2683 |
| 4 | 27060.2974 | 14389.5000 | 14831.0000 | 5017286.0000 | 11613046088.4650 | 107763.8441 |
| 5 | 22599.0595 | 14381.5000 | 14090.0000 | 917187.0000 | 1031735813.9105 | 32120.6447 |
| 6 | 26786.6993 | 14402.0000 | 14456.2000 | 5010899.0000 | 9159074562.4921 | 95703.0541 |
| 7 | 32704.3645 | 14824.0000 | 13805.0000 | 5031542.0000 | 12978858245.0844 | 113924.7921 |
| 8 | 23835.5302 | 14416.0000 | 13629.0000 | 5024288.0000 | 8709993471.8928 | 93327.3458 |
| 9 | 26194.2169 | 14713.0000 | 13864.0000 | 5010368.0000 | 7824946833.7107 | 88458.7296 |
| 10 | 30189.0427 | 16263.0000 | 13965.1429 | 5079925.0000 | 13851055077.6727 | 117690.5055 |
| 11 | 55067.7188 | 16160.5000 | 14090.0000 | 16307750.0000 | 54714575157.1476 | 233911.4686 |
| 12 | 106253.9010 | 16262.5000 | 14016.0000 | 31776220.9999 | 847063808125.0386 | 920360.6946 |
| 13 | 82068.0568 | 27998.5000 | 17119.0000 | 5397885.0000 | 32965100879.4455 | 181562.9392 |
| 14 | 122862.4006 | 35043.0000 | 16341.5714 | 5409063.0000 | 34930431912.0527 | 186896.8483 |
| 15 | 206468.5737 | 139531.5000 | 32715.5333 | 5781877.0000 | 77128103459.0351 | 277719.4690 |
| 16 | 380212.4991 | 295375.0000 | 198935.7733 | 6141690.0000 | 130641066580.8962 | 361443.0337 |

Table E.14: Statistical data for networked test conducted with ACE+TAO 1.3

| File Sizes($2^n$) | Mean | Median | Mode | Range | Variance | Standard Deviation |
|---|---|---|---|---|---|---|
| 0 | 597.0377 | 497.0000 | 490.0000 | 51097.0000 | 1255394.3341 | 1120.4438 |
| 1 | 627.9038 | 494.0000 | 490.0000 | 127220.0000 | 6073664.3697 | 2464.4805 |
| 2 | 643.4985 | 495.0000 | 490.0000 | 155433.0001 | 7378785.0174 | 2716.3919 |
| 3 | 638.9227 | 495.0000 | 491.0000 | 396900.0001 | 18227984.4187 | 4269.4244 |
| 4 | 609.8212 | 497.0000 | 492.0000 | 78219.0000 | 2125908.9115 | 1458.0497 |
| 5 | 625.9842 | 503.0000 | 498.0000 | 120660.0000 | 3676247.9017 | 1917.3544 |
| 6 | 644.5256 | 518.0000 | 513.0000 | 118630.0000 | 3237523.8855 | 1799.3121 |
| 7 | 661.1104 | 539.0000 | 535.0000 | 85954.0001 | 2684135.9956 | 1638.3333 |
| 8 | 753.5316 | 578.0000 | 573.0000 | 406505.0000 | 19250512.5630 | 4387.5406 |
| 9 | 809.0285 | 655.0000 | 650.0000 | 96026.0001 | 3792788.5253 | 1947.5083 |
| 10 | 938.4137 | 803.0000 | 796.0000 | 78060.0000 | 2809532.5117 | 1676.1660 |
| 11 | 22424.8397 | 994.0000 | 979.0000 | 1680632.0000 | 8848292918.1009 | 94065.3651 |
| 12 | 76296.8149 | 1198.0000 | 1145.0000 | 31768787.0000 | 664825571081.8645 | 815368.3653 |
| 13 | 58294.5848 | 2626.0000 | 1849.0000 | 3605921.0000 | 16822987653.3519 | 129703.4605 |
| 14 | 99144.2039 | 15955.0000 | 3775.0000 | 2109358.0000 | 27978943664.6485 | 167269.0756 |
| 15 | 175952.0868 | 120952.5000 | 7214.0000 | 2914690.0001 | 51496770523.8752 | 226928.9989 |
| 16 | 331137.9436 | 251632.5000 | 11471.4000 | 5708115.0000 | 110064772237.6688 | 331760.1125 |

Table E.15: Statistical data for networked test conducted with MICO 2.3.8

| File Sizes($2^n$) | Mean | Median | Mode | Range | Variance | Standard Deviation |
|---|---|---|---|---|---|---|
| 0 | 27562.4246 | 15753.5000 | 13203.0000 | 5062506.0000 | 18222766098.4282 | 134991.7260 |
| 1 | 28148.7837 | 15415.0000 | 13460.0000 | 5087606.0000 | 20230256739.8955 | 142233.1070 |
| 2 | 25604.6345 | 15571.0000 | 14256.0000 | 5016037.0000 | 8832237393.9067 | 93979.9840 |
| 3 | 28714.9828 | 15305.5000 | 14057.8571 | 5459783.0001 | 26305496442.5835 | 162189.6928 |
| 4 | 29158.2340 | 15650.0000 | 13044.0000 | 5035491.0000 | 25586303561.9083 | 159957.1929 |
| 5 | 25809.0630 | 15200.5000 | 13171.0000 | 5035443.0000 | 18382761051.7502 | 135583.0412 |
| 6 | 23988.0153 | 15086.0000 | 13362.0000 | 5015113.0000 | 10623712105.6214 | 103071.3932 |
| 7 | 25914.3141 | 15200.0000 | 13316.4000 | 5078979.0000 | 20583054327.1654 | 143467.9558 |
| 8 | 23838.4093 | 15472.5000 | 13982.7500 | 5038013.0000 | 5664986584.9571 | 75266.1052 |
| 9 | 25245.4766 | 15440.5000 | 13279.5000 | 5094021.0000 | 12233663070.4109 | 110605.8908 |
| 10 | 25492.0450 | 15784.0000 | 13294.0000 | 5024109.0000 | 15528391517.0526 | 124612.9669 |
| 11 | 44486.3227 | 16317.5000 | 15139.5000 | 5009970.0000 | 16780279127.4953 | 129538.7167 |
| 12 | 85499.1554 | 17017.0000 | 15758.6250 | 30484614.0000 | 448535333209.4174 | 669727.8053 |
| 13 | 67689.2779 | 17880.0000 | 16015.5000 | 5619798.0000 | 20177067165.7893 | 142046.0037 |
| 14 | 119572.9521 | 30491.0000 | 15961.0000 | 5017352.0000 | 35683320949.5925 | 188900.2937 |
| 15 | 203009.1772 | 136888.0000 | 21726.5000 | 5249592.0000 | 63531426412.2855 | 252054.4116 |
| 16 | 379671.4401 | 282891.0000 | 26401.0000 | 24593971.9999 | 184277193226.4561 | 429275.1952 |

Table E.16: Statistical data for networked test conducted with Orbix 2000 v2.0

# Notes

[1]Details about which parts of the CORBA® specification are supported by j2sdk.1.4.1 can be found at by visiting `http://java.sun.com/j2se/1.4.1/docs/api/org/omg/CORBA/doc-files/compliance.html`.

# Bibliography


[1] *Borland - Enterprise Server VisiBroker Edition taking CORBA applications to the Web*, [web page] <http://www.borland.com/besvisibroker> [Accessed February, 2003].

[2] *Doxygen*, [web site] <http://www.doxygen.org> [Accessed March, 2003].

[3] *JNI - Java(TM) Native Interface*, [web page] <http://java.sun.com/products/jdk/1.2/docs/guide/jni> [Accessed February, 2003].

[4] *JUnit*, [web site] http://junit.sourceforge.net [Accessed January, 2003].

[5] *KDevelop - Home*, [web site] <http://www.kdevelop.org> [Accessed January, 2003].

[6] *Real-time CORBA with TAO (The ACE ORB)*, [web site] <http://www.cs.wustl.edu/~schmidt/TAO.html> [Accessed January, 2003].

[7] *Welcome to IONA - Solutions*, [web page] <http://www.iona.com/products> [Accessed January , 2003].

[8] *Software Defined Radio - SCARI Project*, 1999, [web site] <http://www.crc.ca/en/html/scari/home/home> [Accessed January, 2003].

[9] *Autoconf - GNU Project - Free Software Foundation (FSF)*, 2002, [web site] <http://www.gnu.org/software/autoconf> [Accessed February, 2003].

[10] *Communications Research Centre Canada / Centre de recherches sur les communications Canada*, 2002,
[web site] `http://www.crc.ca/` [Accessed January, 2003].

[11] *CppUnit - The Unit Testing Library*, 2002,
[web site] `<http://cppunit.sourceforge.net>` [Accessed January, 2003].

[12] *GNU Automake - GNU Project - Free Software Foundation (FSF)*, 2002,
[web site]`<http://www.gnu.org/software/automake>` [Accessed February, 2003].

[13] *MICO - Mico Is COrba*, 2002,
[web site] `<http://www.mico.org>` [Accessed January, 2003].

[14] *Official Specifications for CORBA support in J2SE 1.4*, 2002,
[web page] `<http://java.sun.com/j2se/1.4.1/docs/api/org/omg/CORBA/doc-files/compli%ance.html>` [Accessed March 30th, 2003].

[15] *Boost C++ Libraries*, 2003,
[web site] `<http://www.boost.org/>` [Accessed January, 2003].

[16] Michi Henning and Steve Vinoski, *Advanced CORBA Programming with C++*, Addison-Wesley, 1999.

[17] Andrew Hunt and David Thomas, *The Pragmatic Programmer*, Addison-Wesley, 1999.

[18] Peter Kurpis, *Peter Kurpis - Re: using a static library in a shared library?*, 2001,
[web posting] `<http://gcc.gnu.org/ml/gcc-help/2002-07/msg00139.html>` [Accessed March, 2003].

[19] Modular Software programmable Radio Consortium, *Software Communication Architecture Specification v 2.2*, (2001).

[20] Richard G. Russell, *Defining Software Quality - An Essay* , 2002,
[web page] `<http://www.io.com/~richardr/writing/Engineering/SoftwareQuality>` [Accessed April 5, 2003].

[21] W. Richard Stevens, *Advanced Programming in the UNIX Environment*, Addison Wesley Longman, Inc., 1993.

[22] Andrew S. Tanenbaum, *Modern Operating Systems*, Prentice Hall, Upper Saddle River, NJ 07458, 1992.

[23] Gautam H. Thaker, *Comparing Various Distributed Computing Technologies*, [web site] <http://www.atl.external.lmco.com/projects/QoS/compare/dist_oo_compare.%html> [Accessed January, 2003].

[24] The IEEE and The Open Group, *The Open Group Base Specifications Issue 6 IEEE Std 1003.1, 2003 Edition: errno.h*, [web page] <http://www.opengroup.org/onlinepubs/007904975/basedefs/errno.h.html> [Accesed February, 2003].

[25] Gary V. Vaughan, Ben Elliston, Tom Tromey, and Ian Lance Taylor, *GNU Autoconf, Automake, and Libtool*, New Rider, 2000.