



### The ATON Project

Center for Research in Electronic Art Technology  
University of California, Santa Barbara  
Computer Vision and Robotics Research Laboratory  
University of California, San Diego  
CalTrans Test-Bed Center For Interoperability



# ATON Report 2001.06.4: CORBA Measurements and Programmer Support

Frode Holm, Ahmi Wolf, and Francisco Iovino  
{frode, ahmi, francisco}@create.ucsb.edu  
CREATE, UCSB, June, 2001

## Contents

Preface . . . . .	1
Introduction . . . . .	2
CORBA Interoperability and Performance Measurements . . . . .	2
ORB- and Platform-Independent Code (OPIC) C++ Templates . . . . .	4
Summary . . . . .	7

## Preface

Within the ATON Project, we carried out two main tasks related to the use of CORBA ORBs for multimedia applications. First, in order to support real-time distributed systems better, we need to measure the run-time performance of one or more ORBs using a portable benchmark suite. Next, to allow us to move applications between ORBs and HW/OS platforms, we had to develop a set of source code “template” files that isolate the programmer from the differences between ORBs and platforms. This report describes these two tasks.

## Introduction

Our previous work with TCFI in the IDOT (Impact of Distributed Object Technology Using CORBA) and DRIVE (Distributed Real-Time Interactive Virtual Environments) projects had used several different CORBA ORB products with mixed success. We had already identified several issues with standards compliance and run-time performance among the three ORBs we used in 1996-7 (VisiBroker, OrbixWeb, and SmalltalkBroker). We decided for ATON to revisit the performance issues, and to create a more comprehensive ORB benchmarks to test the ORBs we intended to use this time (TAO ORB, MICO, SmalltalkBroker, and Orbix).

## CORBA Interoperability and Performance Measurements

The goal of the performance-measurement sub-task was to develop a CORBA test suite that would give us insight into the interoperability and performance characteristics of different ORBs. Ideally we wanted to know, from a selected set of ORBs, which one(s) would be the best adapted for serving as a base for a such an ambitious real-time distributed multimedia project as ATON (as well as for other projects at CREATE).

We have thus to:

- select a starting set of ORBs that we consider worth being measured;
- write CORBA benchmark code in C++ and IDL that measures the data throughput on each ORB and between ORBs; and
- run the benchmark suite on the different platforms and collect the results.

The main criteria for choosing a set of ORBs to test were (1) their availability on a wide range of platforms (Solaris, IRIX, MS-Windows NT/2000, Linux), and (2) (optional) the availability of an open source distribution. The TAO ORB was chosen because of its wide support and its implementation of the new CORBA real-time and A/V streaming standards. The MICO ORB was chosen because of its compact and transparent system, clear design, well written documentation, small footprint, and general easy of use. At the request of CalTrans we also initially included Orbix, but had to abandon this ORB because of problems with licenses and ORB portability.

These machines and platforms were used for the tests:

- Windows/PC: NEC Powermate 8100, 450Mhz Pentium II, Windows NT (Waltz)
- Sun/UNIX: Sun Ultra 10, Solaris 5.8, (hinv service turned off) (Belly)

These machines have similar clock speeds and cost about the same price (in the range of \$2500 fully configured—with maximum academic discount), though the PC is older, and faster Pentium processors are now available. The machines are connected through a relatively “quiet” switched 100Base-T LAN.

Because of our project-related biases (and previous experience), the benchmark code is oriented toward measuring ORB data marshalling performance, and is partitioned into three components: a manager, a producer and a consumer.

The producer makes calls to the consumer with a variable size buffer as the argument. The buffer itself can be one of three different types: boolean, long and struct. The consumer just

executes a null method and returns. The test for structs was only performed for buffer sizes up to 256 kB, as the time taken to complete the test became prohibitively high. The struct is defined in CORBA IDL as follows:

```
struct MyStruct {
    boolean b;
    string s;
    long l;
};
```

The manager records the round-trip time between producer and consumer, averaged over a large number of iterations.

We only show results for MICO because it turned out that TAO crashed in all runs of this particular example due to a memory leak bug on both WindowsNT and Solaris. We spent a considerable amount of time trying to solve this problem, but it proved to be intractable given the available time and resources. The tests were performed in four different configurations, two locally on a single machine and two over the network.

- 1: PC (Producer) -> PC (Consumer)
- 2: Sun (Producer) -> Sun (Consumer)
- 3: PC (Producer) -> Sun (Consumer)
- 4: Sun (Producer) -> PC (Consumer)

The plots of the results are shown below. The vertical axis corresponds to throughput (in kB/sec) and the horizontal axis corresponds to the size of buffer. Note that the vertical axes differ between the graphs.

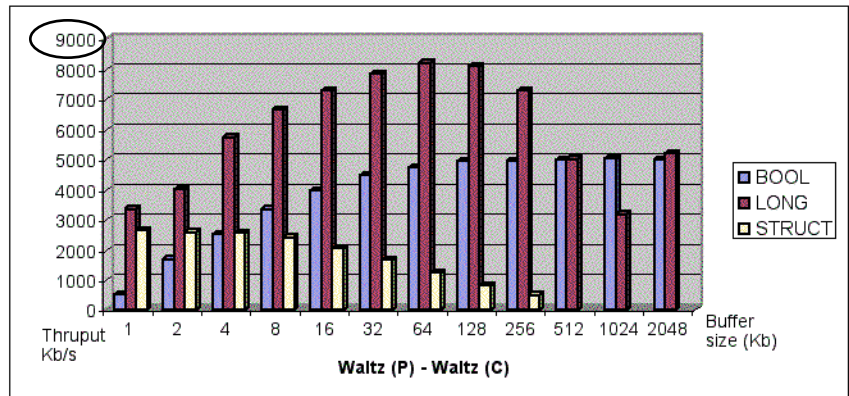
The first thing one notices is that, while the graphs for the first two tests look very similar, the scales are quite different. The Sun machine is consistently faster than the Pentium by a factor of between 2.0 and 2.5. This implies that even a new 1 GHz pentium would still somewhat slower than last year's lowest-end Sun UltraSPARC-based server.

More importantly, if we realize that most streaming media data will be large blocks of simple (i.e., unstructured) data, these results (and those that follow) tell us that we can easily use CORBA as-is for passing data between processes in an HPDM application. The lowest inter-machine throughput for medium-sized (98 kB) blocks is approximately 5 MB/sec. The inter-machine benchmarks in tests 3 and 4 show us that the ORB has a latency in the range of 1-2 msec, which is in the range that we deemed to be acceptable. There are other benchmarks in the literature that affirm this.

As expected, the cost of marshalling was highest with structs, since this type has three different items that need serializing. In all scenarios, the highest throughput was achieved with the long type and buffer sizes of 64 or 128 kB. Presumably, this configuration allowed the marshaller to avoid otherwise necessary overhead. Unfortunately, we were not able to measure the effect of the network properly, since we didn't have any set of 2 identical machines.

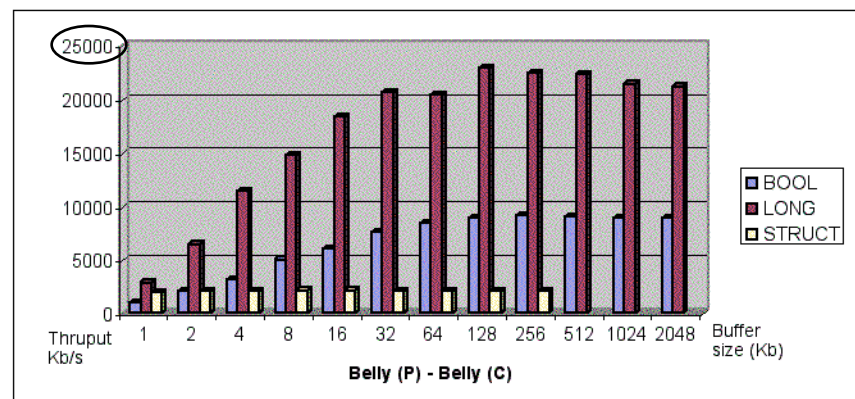
Waltz: MICO (P) -- Waltz: MICO (C)

PC	PC	LONG	STRUCT
	BOOL		
1	500	3333	2618
2	1667	4000	2564
4	2500	5714	2528
8	3333	6612	2391
16	3960	7273	2043
32	4444	7805	1663
64	4706	8184	1209
128	4916	8091	789
256	4916	7283	478
512	4954	4993	
1024	5003	3158	
2048	4978	5180	



Belly: MICO (P) -- Belly: MICO (C)

Sun	Sun	LONG	STRUCT
	BOOL		
1	1000	2857	1969
2	2083	6452	2062
4	3125	11429	2110
8	5000	14815	2128
16	6061	18391	2128
32	7547	20645	2105
64	8466	20382	2066
128	8889	22898	2050
256	9156	22437	2044
512	9033	22309	
1024	8895	21432	
2048	8945	21186	



**Figure: Results for ORB Benchmark 1 and 2 (performance of CORBA calls between processes on the same machine)**

As we expected, the inter-machine ORB tests are somewhat slower than intra-machine results given in the first two graphs, and the direction of the data transfer (Sun-to-PC or PC-to-Sun) is not a significant factor. We cannot easily explain at present why the PC-to-Sun version is consistently about 15% faster than the reverse direction.

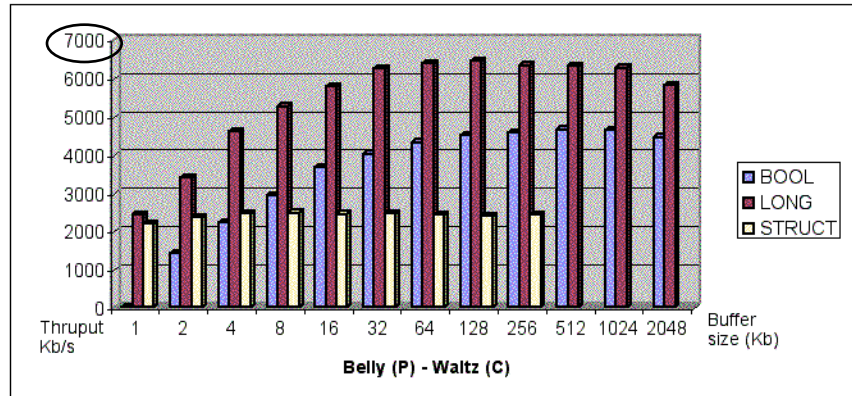
The longer-term utility of these tests (and the others in the suite) is that we can now easily evaluate other ORB products that we might consider using in the future in terms of their scalability and appropriateness for real-time multimedia applications. Because it uses the OPIC templates (see below), the benchmark code should be relatively easily portable to other ORBs or development platforms.

## ORB- and Platform-Independent Code (OPIC) C++ Templates

As part of our effort to standardize as much of our code base as possible, we attempted to create a set of basic ORB examples and "template" files that would work unchanged on any combination of different ORBs and platforms. We were largely successful in this endeavor, which is due in part to the fact that CORBA code has started to come of age, so that several commonly accepted conventions have been adopted, even in areas where such conformance has not been mandated by the CORBA standard itself.

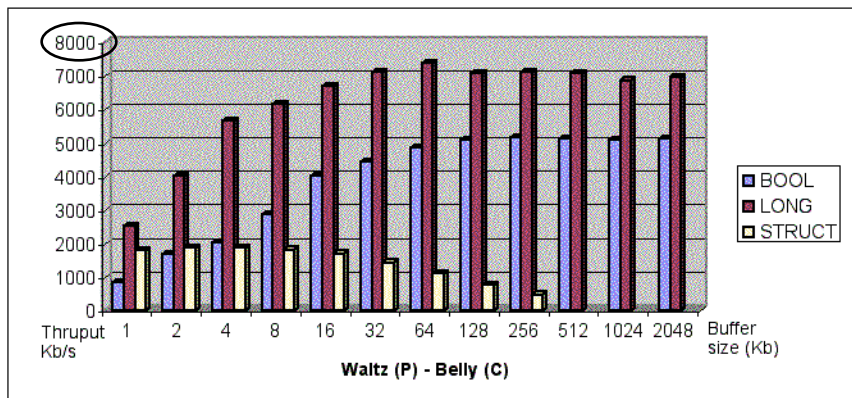
**Belly: MICO (P) -- Waltz: MICO (C)**

Sun	PC	BOOL	LONG	STRUCT
1	3	2439	2193	
2	1429	3390	2358	
4	2222	4598	2463	
8	2941	5263	2477	
16	3670	5776	2443	
32	4020	6262	2448	
64	4324	6400	2426	
128	4494	6455	2405	
256	4578	6338	2416	
512	4660	6321		
1024	4653	6269		
2048	4458	5802		



**Waltz: MICO (P) -- Belly: B21MICO (C)**

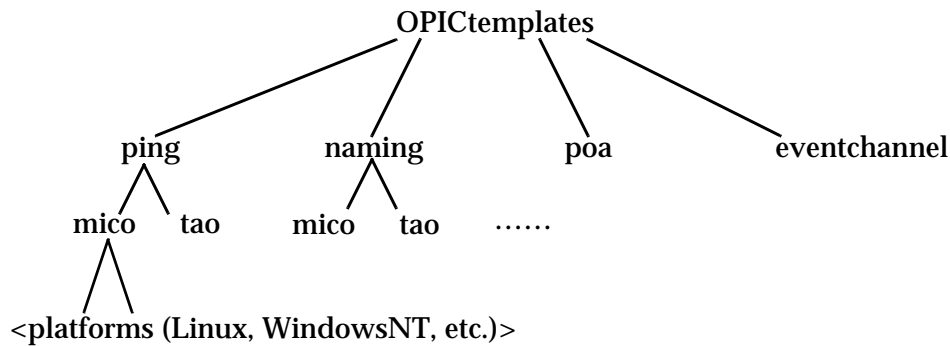
Sun	PC	BOOL	LONG	STRUCT
1	833	2500	1779	
2	1667	4000	1848	
4	2000	5634	1867	
8	2857	6154	1808	
16	4000	6667	1689	
32	4420	7095	1424	
64	4848	7348	1095	
128	5071	7060	752	
256	5153	7082	468	
512	5094	7062		
1024	5072	6849		
2048	5113	6942		



**Figure: Results for ORB Benchmark 3 and 4 (throughput of CORBA calls between different machines)**

In the templates we have provided, there are only two small areas where the source code differ between the two ORBs we tested (MICO and TAO). The first is the naming of the implementation classes and corresponding files, which are generated from the IDL source. This would be easy to normalize as both ORBs have IDL compiler command line arguments that allow for overriding the default names (i.e., we can fix this in our makefiles). The second is the naming of non-editable include files, also generated by the IDL compiler, that is unfortunately not available for change. It would be possible to hide this detail in a common header file, so that, in principle, just one set of source files could be used unmodified on any combination of ORBs and OS platforms.

We did not take it quite this far, so in the examples, the MICO and TAO source files have been given separate directories. Compiler and platform dependent files (makefiles, scripts etc.) are in further subdirectories. The overall structure of the "OPICtemplates" directory is as shown in the Figure below.



**Figure: OPIC Code Directory Structure**

*Ping*

This is the most basic and simple example of a CORBA program possible. One object responds to a ping() method by incrementing a counter and returning its value. In this example the client and server resolves the object reference through an IOR (Interoperable Object Reference) written to a file.

*Naming*

This the same as the ping example, but uses the CORBA Naming Service for resolving object references.

*POA*

This example is intended to set up and demonstrate a few of the basic capabilities of the POA Manager (Portable Object Adapter). The code sample shows how to create child POAs, necessary to exploit features not available in the root manager. The main feature shown is activation “on demand.” In this scenario the servant object is not created until an invocation is made upon it. This is done by a special Servant Manager (Activator) class (user written) that knows how to create the appropriate object.

In other respects this example is identical to the Naming example.

*EventChannel*

This is work in progress and contains adapted code from the MICO example code. It consists of an event-push example with supplier/consumer classes (which was only tested on one ORB).

*Discussion*

Except for the areas mentioned above, the OPIC code templates run unaltered on all combinations of the ORBs (MICO and TAO) and platforms (Linux, Solaris, WindowsNT) that we use. The OPIC templates greatly enhanced our productivity as we built the various CORBA servers and clients we needed for the ATON virtual environment.

## Summary

The work described here supports the rest of our CORBA based work, including the input servers for the ATON virtual environments, and the DPE distributed processing environment. We are using the OPIC templates now for our C++ Development on several platforms, and can use our benchmark suite to evaluate new ORBs or network infrastructures in the future.

The initial result of the benchmarks is that we have demonstrated that some off-the-shelf ORBs are indeed suitable for our class of applications, but that there are wide variations in platform performance and ORB implementations.