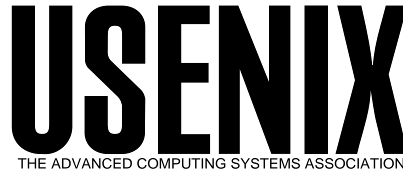


The following paper was originally published in the

5th USENIX Conference on Object-Oriented Technologies and Systems (COOTS '99)

San Diego, California, USA, May 3–7, 1999

Performance Patterns: Automated Scenario-Based ORB Performance Evaluation

S. Nimmagadda, C. Liyanaarachchi, D. Niehaus
University of Kansas

A. Gopinath, A. Kaushal
Sprint Corporation

© 1999 by The USENIX Association
All Rights Reserved

Rights to individual papers remain with the author or the author's employer. Permission is granted for noncommercial reproduction of the work for educational or research purposes. This copyright notice must be included in the reproduced paper. USENIX acknowledges all trademarks herein.

For more information about the USENIX Association:
Phone: 1 510 528 8649 FAX: 1 510 548 5738
Email: office@usenix.org WWW: http://www.usenix.org

Performance Patterns: Automated Scenario Based ORB Performance Evaluation*

S. Nimmagadda, C. Liyanaarachchi, A. Gopinath†, D. Niehaus, A. Kaushal†
Information and Telecommunication Technology Center
Electrical Engineering and Computer Science Department
University of Kansas

†Sprint Corporation

Abstract

The performance of CORBA *(Common Object Request Broker Architecture)* objects is greatly influenced by the *application context* and by the performance of the ORB *endsystem*, which consists of the middleware, the operating system and the underlying network. Application developers need to evaluate how candidate application object architectures will perform within heterogenous computing environments, but a lack of standard and user extendable performance benchmark suites exercising all aspects of the ORB endsystem under realistic application scenarios makes this difficult. This paper introduces the *Performance Pattern Language* and the *Performance Measurement Object* which address these problems by providing an automated script based framework within which extensive ORB endsystem performance benchmarks may be efficiently described and automatically executed.

1 Introduction

The Common Object Request Broker Architecture (CORBA)[15] is emerging as an important open standard for distributed-object computing, especially in heterogenous computing environments combining multiple platforms, networks, applications, and legacy systems[27]. Although the CORBA specifications define the features of a compliant ORB, they do not specify how the standards are to be implemented. As a result, the performance of a given application supported by ORBs from different vendors can differ greatly, as can the performance of different applications supported by the same ORB.

A number of efforts have been made to measure the performance of ORBs, often comparing with performance of other ORBs [23]. These efforts generally measure only specific aspects of ORB performance in isolation. While performance of specific ORB functions is important, it is also important to realize that superior results in a few simple tests *does not ensure* that the aggregate performance of ORB A is better than ORB B for a particular application object architecture. The performance of ORB based applications implemented as a set of objects is greatly influenced by the application context and by the architecture and performance of the ORB endsystem. The endsystem consists of the ORB middleware, the operating system and the underlying network. An application's performance is determined by how well these components cooperate to meet the particular needs of the application.

Current benchmark suites and methods tend to concentrate on a specific part of the endsystem. Operating system benchmarks concentrate on component operating system operations, but may say comparatively little about how well the operating system will support ORB middleware. ORB benchmarks concentrate on component operations of the middleware, but are less effective at pinpointing problems at the application, operating system, and network layers. Developers considering non-trivial ORB based applications need the ability to evaluate, in some detail, how well a given ORB and endsystem combination can support candidate application object architectures. They need this information before implementing a significant portion of the entire application. Such developers should begin with a set of standard performance benchmark suites exercising various aspects of the ORB endsystem under realistic application scenarios, but they also require the ability to create test scenarios which specifically model their candidate application architectures and behavior in the endsystem context.

Current benchmarking methods and test suites do not adequately solve the real problem developers face be-

*This work was supported in part by grants from Sprint Corporation.

cause current methods concentrate on only a part of the application and endsystem in isolation and thus do not enable the developer to consider how implementation decisions at various levels interact. An effective and efficient tool set supporting an integrated performance evaluation methodology should support ORB, endsystem, and application oriented tests, should be automated, and should make it easy for the user to extend and modify the set of tests performed. Only such an integrated tool set and benchmark test suite supporting realistic application scenarios and capable of collecting information from all layers of the endsystem can enable developers to effectively evaluate candidate application object architectures *before* implementation.

This paper describes how a combination of tools developed at the University of Kansas (KU) can address this challenge. This integrated tool set represents a significant advance in support for performance evaluation of ORB based applications because it increases the range and complexity of tests that a benchmark suite can contain, it extends the types of performance information which can be gathered during an individual test, and its support for automated test execution significantly extends the number of tests that a practical benchmark suite can contain. The NetSpec tool provides a control framework for script driven automation of distributed performance tests. The Data Stream Kernel Interface (DSKI) provides the ability to gather time stamped events and a variety of other performance data from the operating system as part of a NetSpec experiment. The Performance Measurement Object (PMO) provides the ability to conduct NetSpec based experiments involving CORBA objects, and the Performance Pattern Language (PPL) provides a higher level language for describing NetSpec based experiments involving sets of CORBA objects more succinctly.

NetSpec has been used by a number of research projects at the University of Kansas (KU) and elsewhere. It provides the automation and script based framework supporting experiments including a wide range of conditions, component behaviors, and data collection [12, 16]. NetSpec is designed to be extended and modified by the user through the implementation of *daemons*. Test daemons support basic network performance tests and supply background traffic in other NetSpec based experiments. Measurement daemons gather information during an experiment but contribute no traffic or behavior beyond that required to gather data.

The DSKI is a pseudo-device driver which enables a NetSpec experiment, through the DSKI measurement daemon, to specify and collect the set of operating system level events of interest which occur during the experiment [1]. The PMO is a NetSpec test daemon designed to support CORBA based performance experiments. A NetSpec PMO script can specify the creation of CORBA objects, their execution time behavior, and the relations that hold among the objects. Using existing traffic related NetSpec test daemons, the DSKI, and PMO, a user can write a script specify a set of interacting objects, a set of network background traffic providing a context within which the objects exist, and gather operating system level information about network and operating system level events affecting performance.

A practical drawback of the NetSpec PMO support is that the language is defined at a low level of detail, and PMO scripts for scenarios with many objects are thus long and repetitive. The PPL addresses this by defining a higher level language for more compactly describing application level object interaction scenarios, which abstract the performance aspects of commonly used implementation strategies. We have called these scenarios *performance patterns* to draw a direct analogy to design patterns which the definitive book *Design Patterns* defines on page 3 as "descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context"[6].

A performance pattern is a set of objects exhibiting a set of behaviors, relationships, and interactions typical of an application architecture or class of application architectures. This pattern can be customized through parameter specification or user extension to match the intended application behaviors and architecture as closely as required. The PPL compiler emits NetSpec PMO scripts implementing the specified performance pattern.

It is important to realize that the PPL approach is quite general and is not ORB or even CORBA specific. The PPL could easily be used to create object based performance scenarios given support from a NetSpec daemon of the correct type. The PMO is CORBA specific, but it would be straightforward to implement and analogous NetSpec daemon for DCE or DCOM based performance evaluation. The PMO is not ORB specific and has been ported with minimal effort to four ORBs: The ACE ORB (TAO)[20], OmniORB[23], ExperSoft's CORBAplus[5], and ILU[11]. We currently focus on TAO, OmniORB, and CORBAPlus for project specific reasons. The range of experiments which can be supported is a function, in part, of the set of possible object behaviors supported by the PMO. PMO behaviors are implemented by routines linked into the PMO, and it has been designed to make adding new behaviors simple, thus supporting user extension.

The rest of the paper first discusses related work in Section 2, and then describes the implementation of the PMO and PPL in Section 3. Section 4 presents examples of PMO and PPL use, while Section 5 presents our conclusions and discusses future work.

2 Related Work

A number of efforts have been made to measure the performance of ORBs, often comparing with performance of other ORBs [23]. Earlier studies on the performance of CORBA objects focussed mainly on identifying the performance constraints of an Object Request Broker (ORB) alone. Schmidt analyzed the performance of Orbix and VisiBroker over high speed ATM networks and pointed out key sources of overhead in middleware ORBs [7, 8]. This paper complements Schmidt's work by demonstrating an integrated and automated approach which is capable of simultaneously measuring the influence of the ORB, the operating system and the underlying network on the performance of CORBA objects.

Studies have also been conducted on IDL compiler optimizations that can improve overall performance of the ORB. One such effort is the Flick project[4]. It is claimed that Flick-generated stubs marshal data between 2 and 17 times faster than stubs produced by traditional IDL compilers, resulting in an increased end-to-end throughput by factors between 1.2 and 3.7. While clearly addressing an important topic, the Flick work also clearly concentrates on one specific facet of endsystem performance. Our work complements such efforts by providing a platform within which the effect of such efforts on endsystem performance can be evaluated.

TAO is the ACE ORB being developed at Washington University [17]. This project focuses on: (1) identifying the enhancements required to standard ORB specifications that will enable applications to specify their Quality of Service (QoS) requirements to ORBs, (2) determining features required to build real-time ORBs, (3) integrating the strategies for I/O subsystem architectures and optimizations with ORB middleware, and (4) to capture and document key design patterns necessary to develop, maintain and extend real-time ORB middleware. The work described in this paper compliments these goals by providing a way to capture, document, and evaluate performance aspects of ORB based design patterns.

In addition to providing a real-time ORB, TAO is an integrated ORB endsystem architecture that consists of a high-performance I/O subsystem and an ATM port inter-connect controller (APIC). They have developed a wide range of performance tests which include throughput tests[7], latency tests[8] and demultiplexing tests[8]. They have used these performance tests to test TAO [17] and other CORBA2.0 compliant ORBs. Their tests formed a basis for several of the basic tests in the automated framework described in this paper.

A commercially available CORBA test suite is the VSORB from X/Open [28]. VSORB is implemented under the TETware test harness, a version of the Test Environment Toolkit (TET), a widely used framework for implementing test suites [24]. It is designed for two primary uses: (1) testing ORB implementations for CORBA conformance and interoperability under formal processes and procedures, and (2) CORBA compliance testing by ORB implementors during product development and quality assurance. This work differs from ours in that it concentrates on compliance rather than performance, but clearly shares the goal of creating a general framework for large scale evaluation tests.

The Manufacturing Engineering Laboratory at the National Institute of Standards & Technology(NIST) takes a different approach towards the benchmarking of CORBA in their current work on the Manufacturer's CORBA Interface Testing Toolkit(MCITT) [13]. They use a emulator-based approach in which the actual servers are replaced by test servers and the person doing the testing only needs to specify the behaviors that are important for the specific scenario being examined. The approach provides an extremely simplified procedural language, the Interface Testing Language, for specifying and testing the behavior of CORBA clients and servers. This work is similar to ours with respect to its abstraction of the object behavior, but it does not explicitly integrate endsystem evaluation, concentrating only on the application and ORB middleware.

The Distributed Systems Research Group at Charles University, Czech Republic, have done a comparison of three ORBs based on a set of criteria including dispatching ability of the ORB, throughput provided for the invocation of different data types, scalability, and performance implications of different threading architectures [2, 19]. The criteria address different aspects of the ORB functionality and the influence of each criterion has been discussed with respect to specific ORB usage scenarios. They have also developed a suite of benchmarking applications for measurement and analysis of ORB middleware performance. This is a strong effort, but the drawback to this approach, in our view, is that it is restricted to evaluating ORB level performance and specific predefined application scenarios. This is significant because application behaviors will vary and their method does not appear, by our understanding, to be designed to support user specified test scenarios.

Performance evaluation is an important topic in many areas of computer system design and implementation, and significant related work exists which does not consider ORB performance. Data bases provide some of the best developed examples of benchmarks addressing application scenarios. It is interesting to observe that both data bases and ORBs support applications by assuming the role of middleware. As such, performance evaluation of data bases is most meaningful and useful to potential users when it considers application scenarios.

The Wisconsin Benchmark is an early effort to systematically measure and compare the performance of relational database systems with database machines[3]. The benchmark is a single-user and single-factor experiment using a synthetic database and a controlled workload. It measures the query optimization performance of database systems with 32 query types to exercise the components of the proposed systems. This is similar to our effort in that it abstracts the application scenario and considers a range of system functions. Our work differs, however, in that we also provide for placing the set of ORB based objects in an endsystem context including background load and traffic.

The ANSI SQL Standard Scalable and Portable Benchmark (AS3AP) models complex and mixed workloads, including single-user and multi-user tests, as well as operational and functional tests [25]. There are 39 single-user queries consisting of utilities, selection, join, projection, aggregate, integrity, and bulk updates. The four multi-user modules include a concurrent random read test and a pure information retrieval (IR) test[26]. The concurrent random write test is used to evaluate the number of concurrent users the system can handle updating the same relation. The mixed IR test and the mixed OLTP test are to measure the effects of the cross-section queries on the system with concurrent random reads or concurrent random writes. This effort has a stronger similarity to ours in that it considers a wider range of activity as well as multiple users. It does not, to our knowledge, provide support for users to specify application based test scenarios.

3 Implementation

We have implemented *an integrated tool-based approach* for performance measurement of ORB endsystem performance. The single most important aspect of our system is that it measures performance within the target environment, rather than relying on published data that may be inaccurate, or which accurately describes aspects of performance under a different environment. The main features of this approach are:

1. A script based approach for conducting performance tests which promises better expressiveness of experiments.

2. The ability to study the performance of CORBA objects in the context of different operating system loads and network traffic.

3. The ability to study the influence of different components of the CORBA endsystem including the middleware, the operating system, and the network on the performance of CORBA objects.

4. The ability to measure the performance of objects in heterogeneous distributed systems from a single point of control.

5. The flexibility and scalability to specify a wide range of distributed tests and behavior patterns. This includes scalability in time, number of objects, and number of hosts supporting the pattern.

6. The ability to measure latencies, throughput and missed deadlines among a wide range of performance metrics.

7. An automated highly scalable framework for performance measurement. This is a crucial feature because it enables practical us of much larger benchmarking suites than non-automated approaches.

The performance metrics which best predict application performance depend, in part, on the properties of the application. This is one reason why a pattern based and automated framework is required. The pattern orientation enables the user to describe scenarios with a rich and varied set of behaviors and requirements, closely matching the proposed application architecture. Automation enables testing on a large scale, permitting the user to test a wide range of parameters under a wide range of conditions, which permits the user to *avoid* making many potentially unjustified assumptions about what aspects of the application, ORB, and endsystem are important in determining performance.

The metrics which will be crucial for important classes of applications include: throughput, latency, scalability, reliability and memory use. The system parameters which can affect application performance with respect to these metrics include: multi-threading, marshalling and demarshalling overhead, demultiplexing and dispatching overhead, operating system scheduling, integration of I/O and scheduling, and network latency. Our approach currently enables us to examine the influence of many of these aspects of the system on performance, and further development will enable us to handle all of them.

Figure 1 shows our integrated benchmarking framework supporting performance evaluation tests. The experiment description expressed in the PPL script is parsed by the PPL compiler which emits a PMO NetSpec script implementing the specified experiment. The NetSpec parser processes the PMO based script and instructs the NetSpec controller daemon to create the specified sets of daemons on each host used by the distributed experiment. Note that Figure 1 illustrates a generic set of daemons, rather than those supporting a specific test. The PMO daemon interfaces the

CORBA based objects on that host to the NetSpec controlling daemon. An additional PMO object is sometimes used, and communicates with the PMO daemon, because CORBA objects can be created dynamically. Note that the line between the PMO objects represents their CORBA based interaction, which is the focus of the experiment. The DSKI measurement daemon, if present, is used to gather performance data from the operating system. It is a generic daemon and is not CORBA based. The traffic daemon is also not CORBA based, but is used to create a context of system load and background traffic within which the CORBA objects exist.

Our approach integrates several existing tools and adds significant new abilities specifically to support CORBA. The tools integrated under this framework are *NetSpec*[12, 16], the *Data Stream Kernel Interface* (DSKI)[1], the *Performance Measurement Object* (PMO)[10, 9], and the *Performance Pattern Language* (PPL). The rest of this section discussed each component in greater detail.

3.1 NetSpec

NetSpec has been used by a number of research projects at the University of Kansas (KU) and elsewhere. It provides the automation and script based framework supporting experiments including a wide range of conditions, component behaviors, and data collection [12, 16]. NetSpec is designed to be extended and modified by the user through the implementation of daemons supporting specific component roles in experiments. Test daemons are used as active components, traffic sources and sinks, while measurement daemons are passive with respect to the experiment since they only collect measurements. Existing NetSpec daemons support network level performance tests with many simultaneous connections and traffic load profiles, as well as data collection from both hosts and network nodes using measurement daemons. A wide range of NetSpec daemons exist, providing a range of behaviors and functions, including: TCP/UDP traffic load, ATM signaling load, SNMP data collection, and DSKI data collection from the operating system.

3.2 Data Stream Kernel Interface

The DSKI is a pseudo-device driver which enables a NetSpec experiment, through the DSKI measurement daemon, to specify and collect a series of time-stamped operating system level events of interest which occur during the experiment [1]. This is particularly useful when considering interactions among the application, middleware, and operating system levels of the endsystem. The primary target platform for the DSKI is Linux, but we have also ported it to DEC UNIX, and as a pseudo-device driver it can be ported relatively easily to any version of UNIX. The DSKI supports a range of data collection options with differing in level of detail and overhead. One particularly powerful feature is the ability to associate and arbitrary *tag* with an event. For example, when the tag is a packet ID or buffer address this enables post processing to track the progress of specific messages through the protocol stack. In the CORBA context, post processing of the event stream shows the amount of time spent by messages in different portions of the operating system when making object request calls to and from the ORB. We are currently working to create a similar ability to create, configure, and process streams of events from the CORBA and application levels.

3.3 Performance Measurement Object

The PMO is a NetSpec test daemon designed to support CORBA based performance experiments. The PMO enables a NetSpec script to specify the creation of CORBA objects, their execution time behavior, and the relations that hold among the objects. The PMO control layer parses the instructions from the NetSpec controller specifying its role within an experiment. These objects can exhibit a variety of behaviors, and are capable of exchanging a wide range of CORBA data with each other.

The PMO provides all the basic abilities required to conduct CORBA based evaluation experiments, but experience has shown that it is not always the best way. The reason for this is that NetSpec's method of ensuring user extendibility and portability also ensures that NetSpec scripts are very long. The best analogy is to consider the NetSpec scripting language an architecture independent assembly language. It is thus possible to describe any desired experiment, but sometimes tedious. The PMO level is appropriate for describing basic CORBA component tests, but can be unwieldy when used for application level object interaction scenarios. The PPL addresses this problem, and is discussed in Section 3.4

The PMO NetSpec script language describes an experiment in terms of sets of daemons. Each daemon specification provides a complete list of parameter-value pairs describing that daemons role in the experiment. Groups of daemons are created and executed by the NetSpec controller either in *serial* or in *parallel*. These simple constructs make it possible to describe a wide range of sophisticated application level behaviors. Additional constructs make it possible to have sets of distributed subordinate controller daemons for large scale distributed experiments. The details of the NetSpec syntax are described elsewhere [12, 16], but the examples

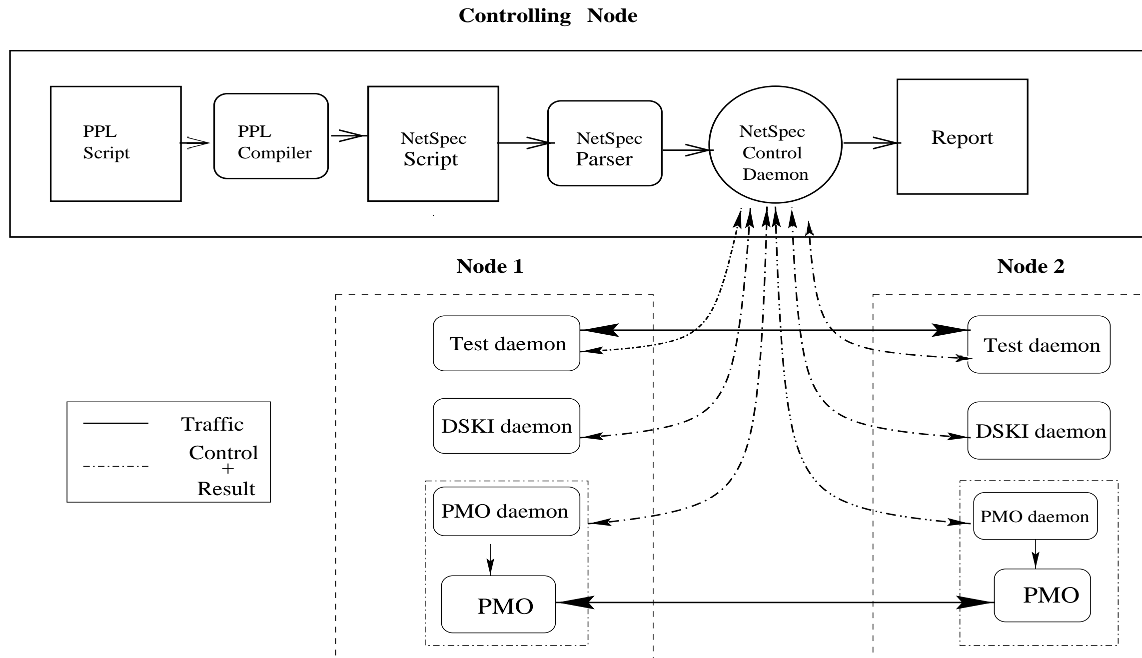


Figure 1. The Integrated Benchmarking Framework

described in Section 4 should provide a clear idea of how the system works.

3.4 Performance Pattern Language

The PPL was designed as a higher level language for describing such application level object interaction scenarios[14, 6] in terms of *performance patterns*. Within each pattern, the user describes objects, object behaviors, test types and relations among the objects that influence the performance of the pattern as a whole. For convenience, the PPL also permits the user to define parameter blocks describing aspects of object behavior which are referenced by object definitions using the same set of parameters. After the patterns associated with the experiment are specified, the *schedule* for their execution is given. Currently the only schedule supported is a simple sequential execution of one pattern at a time. However, we are working on extending this to permit flexible pattern composition and dynamic time dependent behavior to better support application scenario based testing. The correspondence between PPL constructs and the PMO level scripts produced by the PPL compiler is illustrated by the examples in Section 4.

The combination of the PMO and PPL provides a powerful and efficient way for developers to describe and conduct a wide range of application scenario based performance evaluation experiments for CORBA systems. The method is applicable to any ORB and has been ported with minimal effort to four ORBs: The ACE ORB (TAO)[20], OmniORB2[23], ExperSoft's CORBAplus[5], ILU[11]. The range of experiments which can be supported is a function, in part, of the set of possible object behaviors supported by the PMO. PMO behaviors are implemented by routines linked into the PMO, and it has been designed to make adding new behaviors simple.

The scalability of our method is important in two ways. First, script driven automation of the experiments makes it fairly easy to describe tests at a scale representative of the final application. Second, the script driven automation makes it possible to conduct an acceptably large and comprehensive set of tests in an acceptably short period of time. For example, sets of tests producing graphs discussed in Section 4 are fully automated and execute in periods ranging from a few seconds to almost an hour. Scalability is important because the number of properties of an ORB which can significantly affect performance of a particular application is large, requiring a large test suite for adequate evaluation.

4 Evaluation

This section illustrates current capabilities as well as the potential of our automated script driven and application scenario based performance evaluation methods

and tools. The examples show how the tests used in current benchmarks are supported by the framework, and how these can be used as components of more sophisticated scenario based performance patterns. This section presents results of two types of tests under two patterns to illustrate our methods. Section 4.1 presents results under the simple client-server pattern and behaviors for the *cubit* and *throughput* test types. Section 4.2 presents the results under the *proxy* pattern for the same behaviors and test types. Section 4.3 demonstrates the use of the DSKI to reveal the components of the system support overhead for the client-server pattern using a simple request-response behavior. We also demonstrate the portability of our method by presenting results for both Linux and Solaris. Table 1 presents the Linux testing environment for the *cubit* and *throughput* behavior tests, while Table 2 presents that for Solaris. Note that the sending machine is slightly slower than the receiving machine. We originally used identical machines, but a machine failure forced us to use a different receiving machine for tests presented here.

Name of ORB	omniORB2, TAO
Language Mapping	C++
Operating System	Redhat Linux 5.1 kernel 2.1.126
CPU info	Pentium Pro 200 MHz 128 MB RAM
Compiler info	egcs-2.90.27 (egcs-1.0.2 release) no optimizations
Thread package	Linux-Pthreads 0.7
Type of invocation	static
Measurement method	getrusage
Network Info.	ATM

Table 1. Operating Environment Used for the Tests on Linux Platform

Significant further development of our approach is desirable, and is proceeding, but the current capabilities of the tools generally meet and modestly exceed some aspects of current practice. It is important to note that the framework is explicitly designed for user extension precisely because no single developer or authority can know every significant aspect of ORB evaluation. Accumulation of the sum of the CORBA community's collective wisdom concerning ORB evaluation would significantly advance the state of the art. The script based automated approach described here is designed to support such a collective effort.

Parameter	Description
Name of ORB	omniORB2, TAO, CORBAplus
Language Mapping	C++
Operating System	Solaris 2.6
CPU info	Ultra Sparc-II 296 MHz (S) Ultra Sparc-IIi 350 MHz (R) 128 MB RAM
Compiler info	SUN C++ 4.2 no optimizations enabled
Type of invocation	static
Measurement method	getrusage
Network Info.	ATM

Table 2. Operating Environment Used for the Tests on Solaris Platform

4.1 Simple Client-Server Pattern

This example illustrates the basic elements of the PPL and PMO in the context of a simple client-server pattern, which reflects current conventional benchmarking practice. Listed below is the PPL script corresponding to the scenario of Figure 2. The client and server in this case are Sender and Receiver respectively. The information regarding the parameters required for the testing between these two CORBA objects is provided in the *object blocks* of the PPL script and the kind of relation between the objects is specified in the *relation block* of the PPL script. Execution of the pattern is specified by the one line schedule.

The PPL compiler takes the script as input, analyzes the object definitions and relations, and generates the NetSpec PMO script shown in Figure 3. The first thing to note is that the PMO script has two major sections, one defining the client as a *corba* daemon running on the machine *marcus*, and the server as a *corba* daemon running on the machine *zeno*. The other major point is that the parameter block is specified explicitly for each daemon. The main point is that the PMO script defines each object separately and that the relations among them are more difficult to discern in the PMO language.

Figure 4 shows the performance of the Client-Server pattern supporting the *cubit* test type for OmniORB and TAO on a Linux platform, while Figure 5 shows the results for OmniORB, TAO and CORBAplus on a Solaris platform. The CORBAPlus ORB is not currently available for Linux, but should be soon. The flexibility of the script driven approach is demonstrated by the observation that the TAO based tests were repeated for the OmniORB by replacing *orb_name = TAO* with *orb_name = OmniORB* in the PPL script. The *cubit* test emphasizes basic communication performance because it involves

```
pattern CUBIT-TESTS {
  param_block param1 {
    test_type = cubit; orb_name  = TAO;
    minsize   = 512;   maxsize   = 8192;
    predelay   = 5;    postdelay = 5;
    duration = 10;     multiples = 2;
    protocol = iiop;         qos = normal;
    criteria  = latency;
  }

  object Sender  {
    machine_name = marcus; interface = eth;
    behavior = client;        param  = param1;
    numsamples   = 250;
  }

  object Receiver {
    machine_name = zeno;   interface = eth;
    behavior     = server; param     = param1;
    numsamples   = 250;    port_num  = 22222;
  }

 relations {
    (TAO-sender,TAO-receiver);
  }
}

/* Execution Schedule */

CUBIT-TESTS;
```

Figure 2. Simple Client-Server Pattern

```
cluster {
  corba  marcus {
    NameOfORB  = TAO;
    TypeOftest = cubit;
    TestParams = (
      numsamples = 250,  minsize    = 512,
      maxsize    = 8192, multiples  = 2,
      predelay   = 5,    postdelay  = 5,
      duration   = 10 );
    protocol = iiop;
    objname  = Sender;
    role     = client;
    relations = server{Receiver};
    criteria = latency;
    qos      = normal;
    own      = marcus (interface = eth);
  }

  corba  zeno {
    NameOfORB  = TAO;
    TypeOftest = cubit;
    TestParams = (
      numsamples = 250,  minsize    = 512,
      maxsize    = 8192, multiples  = 2,
      predelay   = 5,    postdelay  = 5,
      duration   = 10 );
    protocol = iiop;
    objname  = Receiver;
    role     = server;
    relations = client{Sender};
    criteria = latency;
    qos      = normal;
    own      = zeno (interface = eth, port = 22222);
  }
}
```

Figure 3. Corresponding Client-Server PMO NetSpec Script

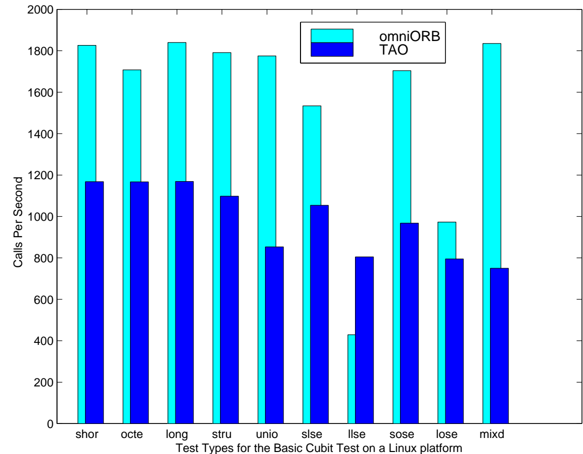


Figure 4. Client-Server Cubit for OmniORB and TAO on Linux

small packets, and a simple computation (cube a number) on the server side. The *cubit* behavior thus focuses on the time spent by each packet in the system layers and the middleware for a CORBA call invocation. The results shown are the average values for 250 invocations of the basic operation for each of several CORBA data types, and are presented in terms of calls per second.

There are several points of interest in these results. First, is the fact that even such a simple test reveals differences between ORB implementations, and between operating system platforms. The most striking difference is that while TAO performance is essentially constant on both Linux and Solaris, OmniORB performance on Solaris is roughly double that on Linux for many data types but not all. Another observation is that OmniORB generally outperforms the other ORBs, but that its performance for the "llse" (long long sequence) data type is substantially below that of TAO on Linux.

Determining why these observed behaviors occur will take further study, but this demonstrates the important point that our compact PPL script describes a test which can be run automatically in a matter of seconds, revealing significant differences in ORB behavior, and providing a convenient and efficient foundation for further experimentation. The flexibility of the PPL approach is further illustrated by changing the test type from *cubit* to *throughput* in the client-server pattern, producing the Linux throughput results for TAO illustrated in Figure 6 and Figure 7 for OmniORB. A data file, essentially CORBA "char" data type, ranging from 1 MB to 64 MB is sent using buffer sizes ranging from 512 bytes to 16 KB. This test shows that throughput for both ORBs is constant with the total amount of data, but that throughput is significantly affected by the

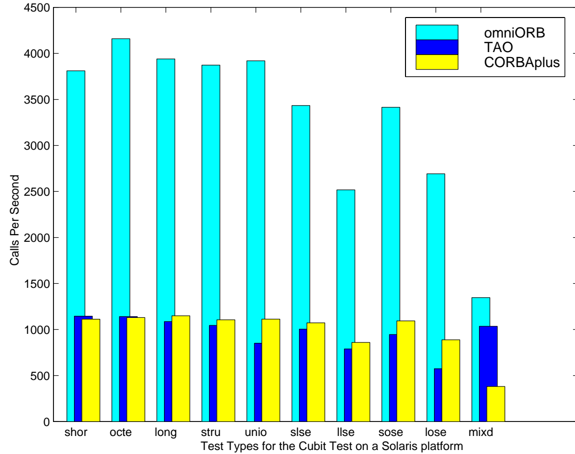


Figure 5. Client-Server Cubit for OmniORB, TAO and CORBAplus on Solaris

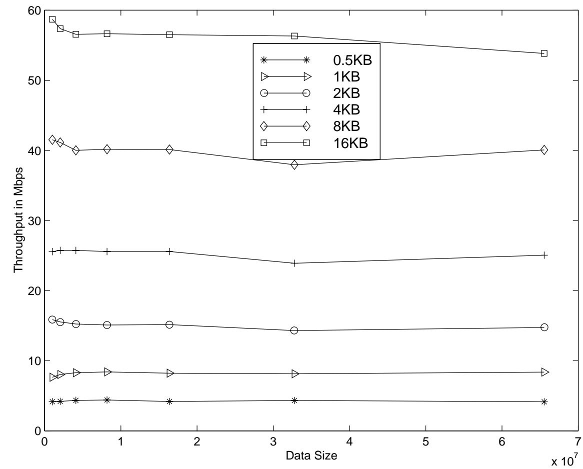


Figure 6. Client-Server Throughput for TAO on Linux

buffer size used for each data transfer session. The TAO throughput increased with buffer size, indicating that the packet transfer rate was limited, but not the packet size. The OmniORB throughput varied in a much less obvious way, and was significantly greater for 4KB buffers. Determining why OmniORB performance varies so haphazardly with buffer size would require gathering data from the operating system layer, as discussed in Section 4.3. The throughput tests for a single client-server pair ran under NetSpec control in an elapsed time of approximately 15 minutes.

4.2 Proxy Pattern

This section discusses a more complex CORBA application scenario, the Proxy Pattern [22], in which the proxy object acts as an interface between the CORBA clients and CORBA servers as shown in Figure 8 for three client and server objects, with the PPL script implementing this pattern for the *cubit* test type under OmniORB. The proxy pattern uses the basic client-server pattern as a component, extending it to a group of client-server pairs communicating through a proxy object. In this case we use three client server pairs under the proxy pattern which exhibit the client and server behaviors, respectively, while executing the *cubit* and *throughput* test types. The client contacts the corresponding server at run-time either by passing the object reference, or the server's name registered with the CORBA Naming Service, to the Proxy object which forwards the client request to the appropriate server. The data type used for the transfer of information between the clients and the proxy Object is CORBA "Any".

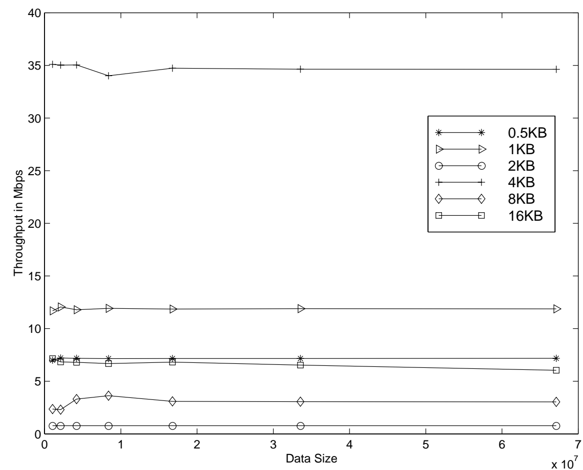
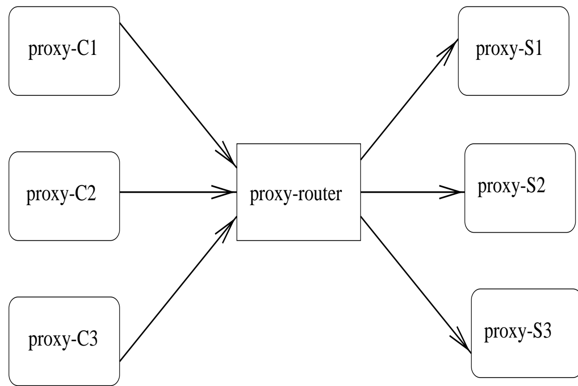


Figure 7. Client-Server Throughput for OmniORB on Linux




```
pattern Proxy  {
  param_block client-param {
    orb_name = omniORB2; test_type = cubit;
    numsamples = 250;    minsize   = 1;
    maxsize    = 1;      multiples = 1;
    predelay   = 3;      postdelay = 3;
    protocol = iiop;          qos = normal;
    criteria = latency;  interface = eth;
    machine_name = marcus;
  }

  param_block server-param {
    orb_name = omniORB2; test_type    = cubit;
    predelay = 3;        postdelay    = 4;
    interface = eth;     machine_name = zeno;
  }

  object proxy-C1 {
   behaviour = client; param = client-param;
  }
  object proxy-C2 {
   behaviour = client; param = client-param;
   predelay  = 5;      postdelay  = 3;
  }
  object proxy-C3 {
   behaviour = client; param = client-param;
   predelay  = 7;      postdelay  = 3;
  }
  object proxy-S1 {
   behaviour = server; param = server-param;
   port_num  = 10000;
  }
  object proxy-S2 {
    behaviour = server; param = server-param;
    port_num  = 20000;
  }
  object proxy-S3 {
    behaviour = server; param = server-param;
    port_num  = 30000;
  }
  object proxy-router {
    behaviour = proxy ; param = server-param;
    port_num  = 30003;
  }

  relations {
    (proxy-C1,proxy-router); (proxy-C2,proxy-router);
    (proxy-C3,proxy-router); (proxy-router,proxy-S1);
    (proxy-router,proxy-S2); (proxy-router,proxy-S3);
  }
}

/* Execution Schedule */
Proxy;
```

Figure 8. Proxy Pattern and PPL Script

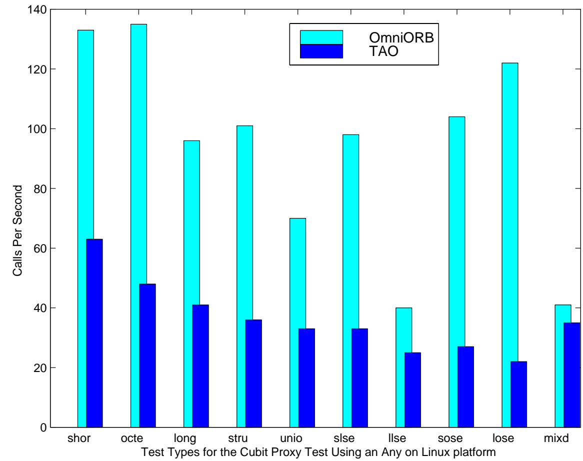


Figure 9. Proxy Cubit Results for OmniORB and TAO on Linux

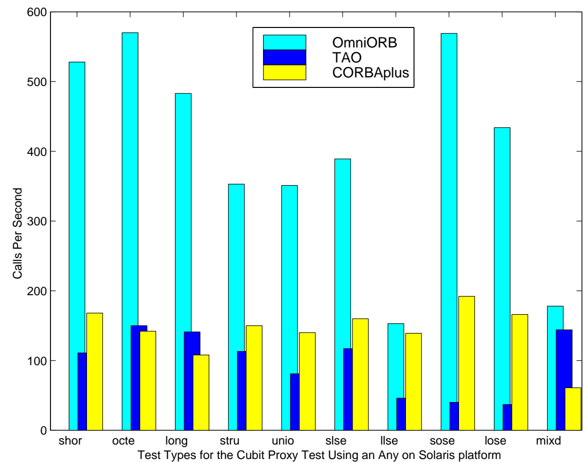


Figure 10. Proxy Cubit Results for OmniORB, TAO and CORBAplus on Solaris

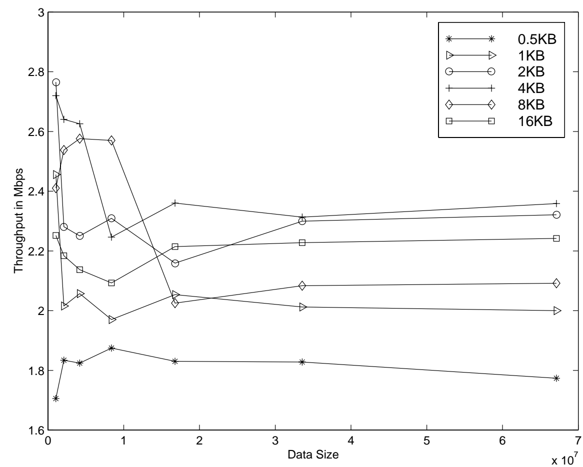


Figure 11. Proxy Throughput for OmniORB on Solaris

The proxy object can be used in two modes. In the first, the proxy only plays a role when establishing a connection between the client and server. In the other mode, the proxy actually routes the data between the objects, with a significant effect on performance. We present results for the second mode.

Figure 9 shows the performance of the omniORB and TAO client objects using the *cubit* test type under the proxy pattern on Linux, while Figure 10 shows the performance of clients under those ORBS as well as under CORBAplus on Solaris. The number of calls per second shown in Figure 9 are the average of the numbers of the three clients in both OmniORB and TAO. There was some non-trivial variance among clients for some tests and some ORBs, which would be another interesting point for further investigation. However, we illustrate the use and utility of our methods using the average results, within which there are several points of interest.

The most obvious point is that using the proxy object to mediate data transfer between client and server significantly impacts performance, reducing it to approximate 10 percent of that for the simple client-server pattern. Some impact is certainly expected due to the use of three concurrent client-server pairs, and reduction to 30 percent of the single pair performance would be plausible. Clearly, using a proxy object has a significant additional impact on performance. While not particularly surprising, this result emphasizes the importance of application scenario based testing. This pattern was, for example, discussed in a popular magazine [22] and is used by one of our colleagues as the basis for a WWW meta-search engine. Clearly, any developer contemplating such an architecture would be grateful to know the likely impact before implementing the software.

The second point of interest is that both TAO and OmniORB enjoy a significant performance increase in moving from Linux to Solaris, while the TAO performance for the client-server pattern was relatively constant between the two systems. A third significant observation is that the magnitude of the performance increase for OmniORB in moving from Linux to Solaris is much greater, increasing three to five fold in most cases. Finally, the difference between TAO and CORBAplus performance under Solaris is greater under this pattern than under the client-server pattern.

These observations support our assertion that application performance scenarios, performance patterns, should be part of any comprehensive benchmark. The comparative performance between different ORBs on the same operating system and between the same ORB on different operating systems changed significantly with the change in pattern. This also supports our idea that developers using performance results to select an ORB and operating system as an implementation platform should use test results for object architectures, performance patterns, which faithfully represent their proposed application.

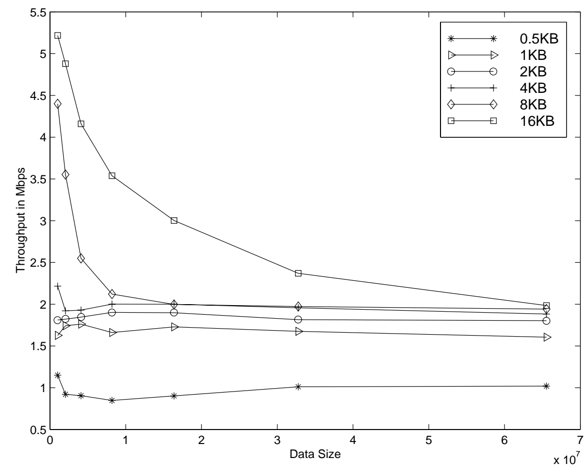


Figure 12. Proxy Throughput for TAO on Solaris

We also changed the test type, as we did for the client-server pattern, to test the *throughput* performance among the object pairs. Figure 11 shows results for the OmniORB under Solaris while Figure 12 presents the throughput results for TAO. Both tests show throughput is reduced from five to ten fold. The TAO results still show an orderly increase in throughput with buffer size, although this converges to a level of 2 Mb/s for all but the smallest buffer size. OmniORB performance, in contrast, does not vary in nearly as orderly a manner with buffer or data set size, and does not converge to similar throughput for most buffer sizes. The performance using 8 KB, 4 KB, and 2 KB buffers is particularly interesting. As with the client-server pattern, the 4 KB buffer size provides the best performance, but 8 KB buffers do substantially better for small data set than large. This could easily be due to system level buffering effects.

Determining why the throughput varies in these ways with data and buffer size will require gathering information from the operating system layer to see if the networking protocols play a role, and gathering information from the ORB layer to see if there is an influence at that level. Section 4.3 illustrates how we might use the DSKI to gather protocol layer information, but discusses a simpler example.

4.3 Using the DSKI

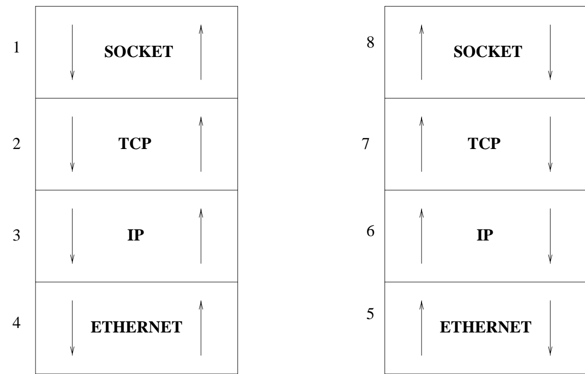
This section briefly illustrates the use of the DSKI to gather performance information from the operating system during a test using the simple client-server pattern

discussed in Section 4.1. As discussed in Section 3 the DSKI creates a stream of time stamped records for each occurrence of a predefined event in the operating system kernel. The set of event records produced can be post-processed to calculate the time spent in providing different types of system services. In this case, the time spent in various portions of the TCP/IP stack can be calculated because we have defined a set of events capable of tracing the progress of packets through the TCP/IP stack. Figure 13 presents the NetSpec PMO script implementing the experiment.

```
cluster {
  corba testbed2 {
    NameOfORB  = omniORB2;
    TypeOftest = rrstring;
    TestParams = (
      numsamples = 1, minsize    = 1,
      maxsize    = 2, multiples  = 2,
      predelay   = 3, postdelay  = 3,
      duration   = 30 );
    protocol  = iiop;
    objname   = omni-receiver;
    role      = server;
    relations = client{omni-sender};
    criteria  = throughput;
    qos       = normal;
    own       = testbed2(interface = eth,
                         port = 41777);
  }

  corba testbed1 {
    NameOfORB  = omniORB2;
    TypeOftest = rrstring;
    TestParams = (
      numsamples = 1, minsize    = 1,
      maxsize    = 2, multiples  = 2,
      predelay   = 3, postdelay  = 3,
      duration   = 30 );
    protocol  = iiop;
    objname   = omni-sender;
    role      = client;
    relations = server{omni-receiver};
    criteria  = throughput;
    qos       = normal;
    own       = testbed1 (interface = eth);
  }

  dstream testbed1 {
    type = active (numevents = 100, port=40778,
                                    duration=30);
    ds_tcpip = all;
  }

  dstream testbed2 {
    type = active (numevents = 100, port=40778,
                                    duration=30);
    ds_tcpip_read = all;
  }
}
```

Figure 13. PMO NetSpec Script using DSKI

Figure 14 presents the packet flow in and out of the Sender and Receiver hosts, as well as the performance figures from the kernel obtained using the DSKI. The socket, TCP, IP, and Ethernet layers are numbered 1 through 4 on the sending host, and 5 through 8 from the bottom up on the receiver host. The path of a packet sent



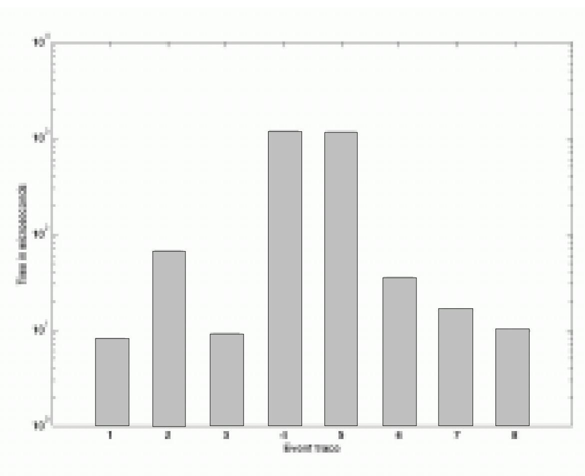


Figure 14. Time trace in the Operating System Layers Obtained Using DSKI

from the sender to the receiver in the diagram would thus flow through layers in numerical order. The bar graph in Figure 14 shows the time in microseconds spent by a packet in each layer, by number.

This experiment demonstrates that the DSKI provides the ability to gather fine grain operating system level time stamped events. Note that the Y-axis in Figure 14 is a logarithmic scale, showing that we were able to monitor time intervals ranging from 10 microseconds to 10 milliseconds. It also illustrates an important feature of the DSKI which we call *tagging*. Each event in the operating system can include a context dependent *tag* when it logs the event. In the case of tracing TCP/IP performance we used the port and sequence numbers to uniquely identify each packet as it moves among protocol layers on each machine.

A similar approach would be used to investigate how buffer size affects throughput under the proxy pattern. DSKI events could be used to monitor when and how

packets are transmitted, combined or broken into fragments, and what size system buffers are used to hold them. Tagging would be used to determine the progress of each packet, and would tell us if the buffer size influenced protocol behavior. Monitoring at the system level might also tell us something about the middleware, even without explicit instrumentation. If, for example, buffers of one size were given to the CORBA layer, but buffers of a different size were passed onto the system layer we would know that the middleware was manipulating the data buffers.

The DSKI can thus be used to investigate a wide range of interactions between the operating system and software layers using its services. This is by no means a simple endeavor, since it requires access to the system source code and sufficient knowledge of the system to enable the investigator to define a reasonable set of events, and then to interpret the results. However, for the investigator willing to learn how to use it well, it can provide an important new source of detailed information from the operating system layer which plays an important role in determining what aspects of endsystem architecture limit application performance under various sets of execution conditions.

5 Conclusions and Future Work

The performance of CORBA based applications implemented as sets of objects is greatly influenced by by the *application context* and by the performance of the ORB *endsystem*. Application developers need to evaluate how candidate application object architectures will perform within heterogenous computing environments, but a lack of standard and user extendable performance benchmark suites exercising all aspects of the ORB endsystem under realistic application scenarios makes this difficult. This paper introduced the *Performance Pattern Language* and the *Performance Measurement Object* which address these problems by providing, under NetSpec control, an automated script based framework within which extensive ORB endsystem performance benchmarks may be efficiently described and automatically executed.

The tools described are implemented, and the viability of the framework they provide has been demonstrated by implementation of small but non-trivial sets of performance evaluation scripts. The examples presented show that the full range of evaluation information can be gathered and a rich set of performance scenarios examined. The automated nature of the script driven framework is also important because it makes it possible to describe and conduct a large set of evaluation experiments covering a adequately diverse and detailed set of scenarios and performance metrics.

Performance evaluation of CORBA based distributed applications, and of candidate object architectures, is an extremely important and difficult problem. Current benchmarking and testing methods are not as comprehensive as they might be because the scale and complexity required is daunting. The tools described here make a significant increase in the scale, complexity, and level of detail of performance evaluation studies possible, thus significantly advancing the state of the art.

Our future work will include creation of new test types and performance patterns. We are particularly interested in extending this approach to testing to include execution of applications under real-time constraints. We will use this set of tests to drive an investigation of what kinds of system support can be used to improve real-time performance of ORB based applications. We will concentrate on a time constrained event service and integration of operating system scheduling, I/O, and ORB level operations to improve time constrained communication among objects.

Availability

NetSpec and many daemons developed for various types of performance evaluation are publicly available. The PMO and the PPL have been developed with support from Sprint, and are not yet publicly available, but should be soon. The work described here is intended as a contribution to the CORBA community and are intended for full availability on the WWW. For further details check:

`www.ittc.ukans.edu/~niehaus/research.html`

References

[1] Buchanan, B., Niehaus, D., Sheth, S., and Wijata, Y. The Data Stream Kernel Interface. Technical Report ITTC-FY98-TR-11510-04, University Of Kansas, December 1997.

[2] CORBA Comparison Project. http://nenya.ms.mff.cuni.cz/thegroup/COMP/index.html.

[3] DeWitt, D. J. The Wisconsin Benchmark: Past, present, and Future. In *The Benchmark Handbook for Database and Transaction Processing Systems*, Ed. by Jim Gray, Morgan Kaufmann, Inc., 1993, pp. 269-316.

[4] Eide, E., Frei, K., Ford, B., Lepreau J. and Lindstorm, G. Flick: A Flexible, Optimizing IDL Compiler. In *ACM SIGPLAN'97 Conference on Programming Language Design and Implementation*, June 1997.

[5] Expersoft Corporation Corbaplus. http://www.expersoft.com/.

[6] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design Patterns*. Addison-Wesley, 1994.

[7] Gokhale, A., and Schmidt, D. C. Measuring the Performance of Communication Middleware on High-Speed Networks. In *SIGCOMM*, August 1996.

[8] Gokhale, A. and Schmidt, D. C. Evaluating CORBA Latency and Scalability Over High-Speed ATM Networks. In *ICDCS*, 1997.

[9] Gopinath, A. Performance Measurement and Analysis of Real-time CORBA Endsystems. Master's thesis, University Of Kansas, June 1998.

[10] Gopinath, A., Nimmagadda, S., Liyanaarachchi, C. and Niehaus, D. Performance Measurement Of CORBA Endsystems. Technical Report ITTC-FY99-TR-14120-01, University Of Kansas, june 1998.

[11] Inter Language Unification. ftp://ftp.parc.xerox.com/pub/ilu/ilu.html.

[12] Lee, B. O., Frost, V. S., and Jonkman, R. Netspec 3.0 Source Models for telnet, ftp, voice, video and www Traffic, January 1997.

[13] MCITT. http://www.mel.nist.gov/msidstaff/flater/mcitt.

[14] Mowbray, T. J. and Malveau, R. C. *CORBA Design Patterns*. John Wiley & Sons Inc., 1997.

[15] Object Management Group. The Common Object Request Broker: Architecture and Specification v 2.2, February 1998.

[16] Roel Jonkman . Netspec: Philosophy, Design and Implementation. Master's thesis, University of Kansas, Lawrence, Kansas, February 1998.

[17] Schmidt, D. C., Bector, R., Levine, D. L., Mungee, S. and Parulkar, G. TAO: A Middleware Framework for Real-Time ORB Endsystems. In *IEEE Workshop on Middleware for Distributed Real-Time Systems and Services*, December 1997.

[18] Schmidt, D. C., Mungeee, S., Flores-Gaitan, S., and Gokhale, A. Alleviating priority inversion and non-determinism in real-time corba orb core architectures. In *Real Time Applications Symposium*, June 1998.

[19] Schmidt, D. C. Evaluating Architectures for Multi-threaded Object Request Brokers. *Communications of the ACM*, 41(10):54–60, 1998.

[20] Schmidt, D. C., Levine, D. L., and Mungee, S. The Design and Perfromance of Real-Time Object Request Brokers. In *Computer Communications*, volume 21, pages 294–324, April 1998.

[21] Schmidt, D. C., Gokhale, A., Harrison, T. H. and Parulkar, G. A High Performance Endsystem Architecture for Real-Time CORBA. *IEEE Communications Magazine*, 14(2), February 1997.

[22] Smith, A. Distributed obsever chains. *Java Report*, 3(9):49–58, 1998.

[23] The Olivetti & Oracle Research Laboratory. Omniorb2, http://www.orl.co.uk/omniorb/.

[24] The Open Group. http://tetworks.opengroup.org/datasheet.html.

[25] Turbyfill, C., Orji, C. and Bitton, D. AS3AP - A Comparative Relational Database Benchmark, In *Proceedings of the IEEE COMPCON*, 1989, pp. 560-564.

[26] Turbyfill, C., Orji, C. and Bitton, D. AS3AP: An ANSI SQL Standard Scaleable and Portable Benchmark for Relational Database Systems, In *The Benchmark Handbook for Database and Transaction Processing Systems* , Ed. by Jim Gray, MorgaKaufmann, Inc., 1993, pp. 317-358.

[27] Vinoski, S. CORBA : Integrating Diverse Applications Within Distributed Heterogeneous Environments. *IEEE Communications Magazine*, 14(2), February 1997.

[28] X/Open Company Ltd. VSORB Test Suite release 1.0.0, April 1997.