Building Blocks for Achieving Quality of Service with Commercial Off-the-Shelf (COTS) Middleware

Andreas Polze

May 1999

TECHNICAL REPORT
CMU/SEI-99-TR-001
ESC-TR-99-001

Carnegie Mellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

Building Blocks for Achieving Quality of Service with Commercial Off-the-Shelf (COTS) Middleware

CMU/SEI-99-TR-001
ESC-TR-99-001

Andreas Polze

May 1999

COTS-Based Systems

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

Mario Moya

Mario Moya, Maj, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense and by Humboldt University. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This document is available through Asset Source for Software Engineering Technology (ASSET): 1350 Earl L. Core Road; PO Box 3305; Morgantown, West Virginia 26505 / Phone: (304) 284-9000 or toll-free in the U.S. 1-800-547-8306 / FAX: (304) 284-9001 World Wide Web: http://www.asset.com / e-mail: sei@asset.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center / Attn: BRR / 8725 John J. Kingman Road / Suite 0944 / Ft. Belvoir, VA 22060-6218 / Phone: (703) 767-8274 or toll-free in the U.S.: 1-800 225-3842.

Table of Contents

List of Figures

Abstract

To date, most of the fault-tolerant, real-time systems have been implemented in embedded settings, and there is an urgent need to open up this type of computing technology to a larger number of people who use heterogeneous distributed computing environments. Today’s transportation, manufacturing, and communication systems require the integration of multiple embedded real-time control systems with standard distributed computing environments in a predictable fashion. Humboldt University has developed the concept of *composite objects* as a filtering bridge between standard middleware platforms and software frameworks providing services with certain quality-of-service (QoS) guarantees. Current research focuses on the Common Object Request Broker Architecture (CORBA) middleware platform; however, composite objects are also applicable to platforms like the Distributed Component Model (DCOM) and distributed computing environments (DCEs). Key concepts in Humboldt’s approach are analytic redundancy, noninterference, interoperability, and adaptive abstraction. These concepts originated in SEI work on the Simplex architecture and have been reapplied to extend the reach of commercial off-the-shelf (COTS) software technologies into demanding application settings (such as those found in military and industrial applications). Here, we discuss building blocks and techniques for fault-tolerant, real-time applications based on CORBA.

1 Introduction

Most of the fault-tolerant, real-time systems have been implemented in embedded settings, and there is an urgent need to open up this type of computing technology to a larger number of people who use heterogeneous distributed computing environments [Soley 98].

The Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) is an important and popular technology that supports the development of object-based, distributed applications. The benefits of abstraction promised by CORBA (location transparency, heterogeneity, dynamic configuration, etc.) are appealing in many application domains, including those that satisfy real-time and fault-tolerance requirements (such as manufacturing, process control, and transport systems). However, CORBA is focused on facilitating general computing environments, and the specification of timing behavior and quality-of-service (QoS) parameters, such as communication latency and acceptable processor utilization, is beyond the scope of the latest version of CORBA.

The Computer Architecture and Organization Group at Humboldt University is studying numerous aspects relating to responsive computing (in particular, techniques for timely delivery of computing results even in the face of faults). In the context of this work, the Responsive CORBA Unified Environment (RESCUE) Project is developing a CORBA-based distributed framework for responsive (fault-tolerant, real-time) services, which exploits consensus for synchronization, reliable communication, and fault diagnosis among replicated server objects. The technical foundation for RESCUE is provided by composite objects, which act as a filtering bridge (see Figure 1-1) between CORBA (which does not provide quality-of-service guarantees) and responsive services (which do provide such guarantees). Composite objects provide CORBA clients with higher predictability regarding timely and reliable method execution.

Here, we present the composite objects approach for predictable integration of CORBA with real-time requirements. We discuss data replication and weak memory consistency as the key concepts for implementing the composite objects approach. We evaluate the timing behavior of composite objects and demonstrate the value of our technique in a number of proof-of-concept scenarios.

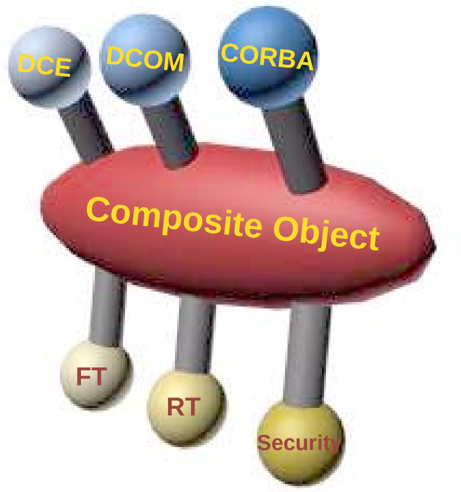


Figure 1-1: Bridging Middleware and QoS-Sensitive Services with Composite Objects

The rest of this report is organized as follows: Section 2 discusses the composite objects approach for predictable integration of CORBA and real-time fault-tolerant computing. Section 3 presents the problems of the producer/consumer/viewer and the "unstoppable robots" models and evaluates their timing behavior. In Section 4, we outline our key concepts for CORBA-based fault-tolerant computing and present example scenarios for fault-tolerant Netscape and the Java-based fault-tolerant maze. In Section 5, we summarize related work with the objective of extending CORBA with real-time and fault-tolerance capabilities. Finally, we present our conclusions and some directions for future work in Section 6.

2 Composite Objects: Making Tradeoffs Explicit

The composite objects approach allows the programmer to make an explicit tradeoff between an application's predictable resource utilization and its communication latency. Therefore, composite objects make implementation details that are usually hidden and abstracted away by CORBA visible and explicit.

The implementation of the composite objects approach follows three basic design rules:

1. *noninterference:* General-purpose computing and responsive computing should not burden each other.
2. *interoperability:* Services exported by general-purpose computing objects and by responsive computing objects can be used by each other.
3. *adaptive abstraction:* Lower level information and scheduling details are available for real-time objects, but are transparent to non-real-time objects.

Since composite objects provide functionality to react on CORBA method invocations, they can be seen as descendants of a class that implements object adaptor functionality (*_skel* in many implementations). On the other hand, composite objects have the capability to create real-time programming abstractions such as prioritized threads and real-time communication channels. As depicted in Figure 2-1, they can be seen as descendants of a class that implements real-time servers.

Composite objects consist of a real-time part and a non-real-time part. Design time and runtime guarantees can be given for execution of real-time methods. In contrast, methods in the non-real-time part are executed following a best-effort approach. Composite objects establish timing firewalls [Polze 97] between real-time and non-real-time (CORBA) computing, so that the non-real-time part cannot violate the real-time scheduling rules that are needed by the real-time part [Sha 94].

.

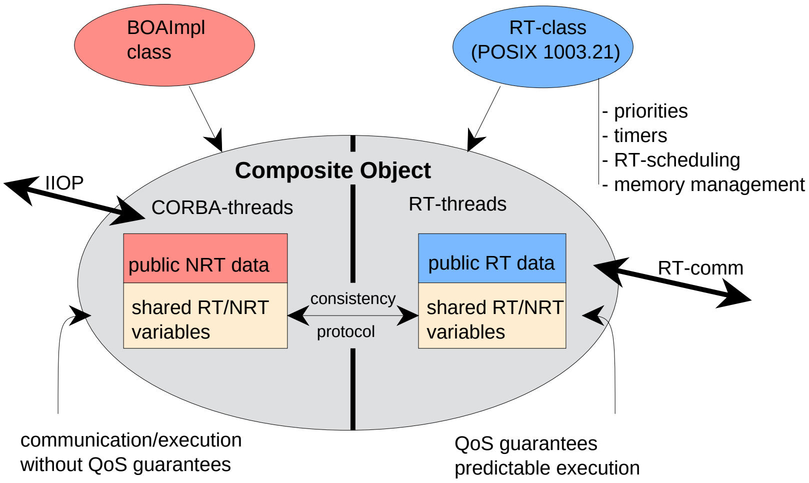


Figure 2-1: Structure of a Composite Object

Data replication is the key to independent data access from real-time and non-real-time threads inside a composite object. We split an object's data into two parts: one that is statically locked in memory to fulfill real-time scheduling assumptions (RT data), and another that is treated like a standard non-real-time user's process data (NRT data) and is subject to memory management such as paging. A pair of RT/NRT variables can be viewed as a replicated variable. Composite objects implement a consistency protocol to update both replicas and create the impression of a shared variable.

The consistency protocol for shared RT/NRT variables as well as resource allocation [central processing unit (CPU) cycles, memory] and call admission for CORBA clients are crucial for implementing the concept of composite objects. Based on the Mach (NeXTSTEP 3.3) and rtLinux [Barabanov 97] operating systems, we have used pipes, shared buffers, and message queues (Mach IPC) for our implementation of the composite objects approach as shown in Figure 2-2.

Figure 2-2 describes the data flow during *read/write* accesses to a composite object's variables. *read* requests in both parts of the composite object are always satisfied by accessing the local copy of a replicated, shared RT/NRT variable. In contrast, *write* requests not only result in updating the local copy, but the value is also stored in an interprocess communication (IPC) data structure (i.e., *pipes*, *rt-fifos*). Handler threads periodically copy values out of the IPC data structures into the local replica of the shared RT/NRT variable. The update rate for those handler threads is programmable.

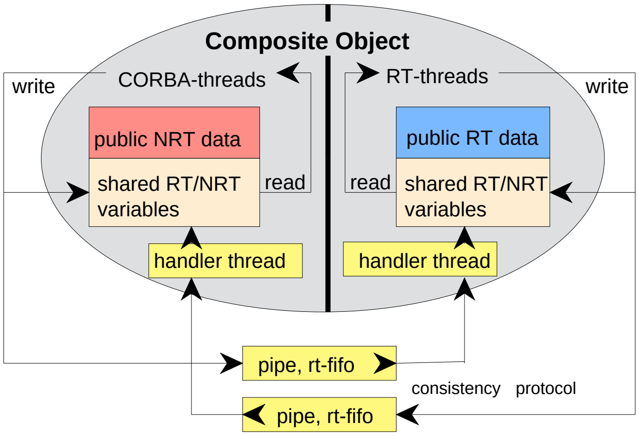


Figure 2-2: Consistency Protocol and Communication Inside a Composite Object

Alternatives to the implementation outlined here include use of *shared memory* or *message passing* (Mach IPC) for data transfer between the CORBA part and real-time part of the composite object. In our experiments, we have evaluated implementations of the composite objects approach based on pipes and shared memory.

Another design alternative is an event-triggered consistency protocol in contrast to the time-triggered, periodic protocol described above. In that case, a *write* operation would include signaling the arrival of a new data value to the handler thread on the opposite side of the composite object. We have implemented consistency protocols following both schemes.

To evaluate the overhead introduced by the composite objects approach, we have investigated a scenario where a real-time data source is accessed by a CORBA client through a composite object that implements an event-triggered consistency protocol.

Figure 2-3 shows communication delays for the described scenario. The lower, dotted curve represents the latency of real-time communication between the composite object and the data source (Mach Interprocess Communication). The middle curve shows the time needed for the consistency protocol inside the composite object plus real-time communication. The distance between the lower and middle curves represents the overhead imposed by our implementation of the composite objects approach. It is roughly 1 ms (millisecond) (10% of the overall communication latency). One should notice that both curves are fairly stable, indicating that introduction of a composite object as a mediator will not disturb an application’s predictable timing behavior.

.

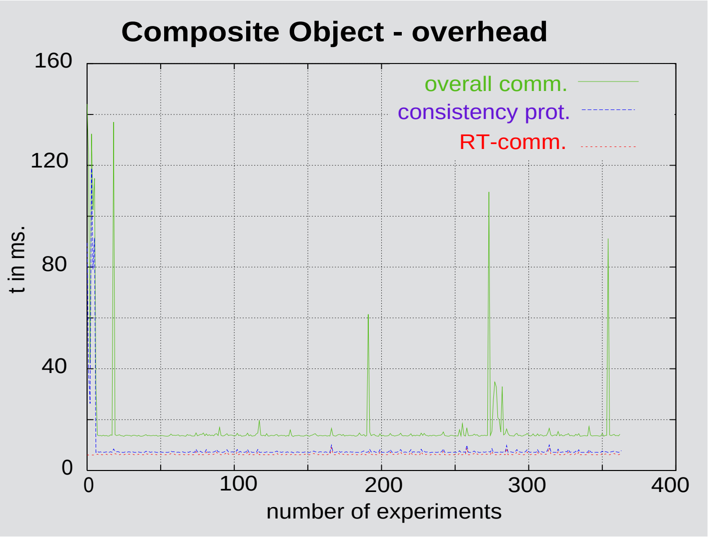


Figure 2-3: Overheads Imposed by Composite Objects

In contrast, the curve at the top of the diagram, which represents the overall communication delay including CORBA, shows clearly visible peaks. Although the average CORBA communication latency is around 12 ms (the client was run remotely), even in the unloaded case it varies occasionally by one order of magnitude. This indicates that the direct execution of CORBA calls from within a time-critical application will significantly disturb the application's predictability.

3 Real-Time Case Studies

3.1 Java Interface to a Soft Real-Time Legacy Application

As a proof-of-concept vehicle, we have created a responsive service by extending a CORE/SONiC (COncensus for REsponsiveness/Shared Objects Net-interconnected Computer) soft real-time legacy application with CORBA interfaces that are accessed by two Java applets.

The "balancing robots" [Werner 96] (shown in Figure 3-1) simulate the following scenario: Robots are moving on top of a plate that is kept in unstable balance. Robots are constantly using energy and, therefore, occasionally have to visit a fuel station, which is located outside of the plate's center. Consensus among robots is achieved prior to every move, with robots that have high levels of fuel acting as a counterbalance to those moving toward the fuel station. Collision avoidance is also implemented by consensus.

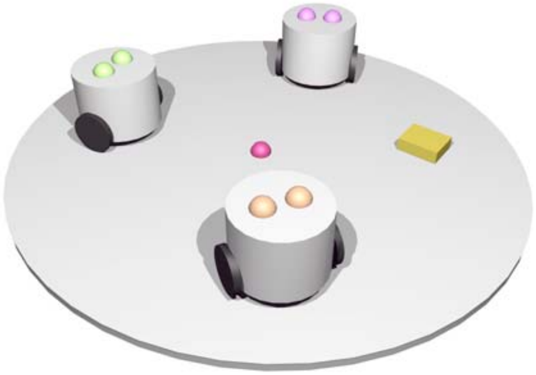


Figure 3-1: Balancing Robots Simulation

The application demonstrates tolerance of the robots' and controllers' crash, omission, and computation faults. Control is subject to soft real-time constraints. Calculations of the controllers' and robots' moves are executed with a frequency of 4 Hz. The application of composite objects in a scenario with more stringent timing requirements is given in Section 3.2.

Using the composite objects technique, we have modified two of the application's interfaces to use CORBA. Using the Java language binding for CORBA, one applet has been implemented to monitor the soft real-time simulation and periodically obtain status information. Another applet implements the robots' control algorithm in Java and interacts with the real-time simulation's native controllers during consensus rounds. These interactions have fixed deadlines (inner control loop) that must be met despite CORBA's and Java's variations in communication latency and execution times. Both Java applets are connected to the "balancing robots" via gateways implemented as composite objects. Figure 3-2 shows the communication structure of the complete application.

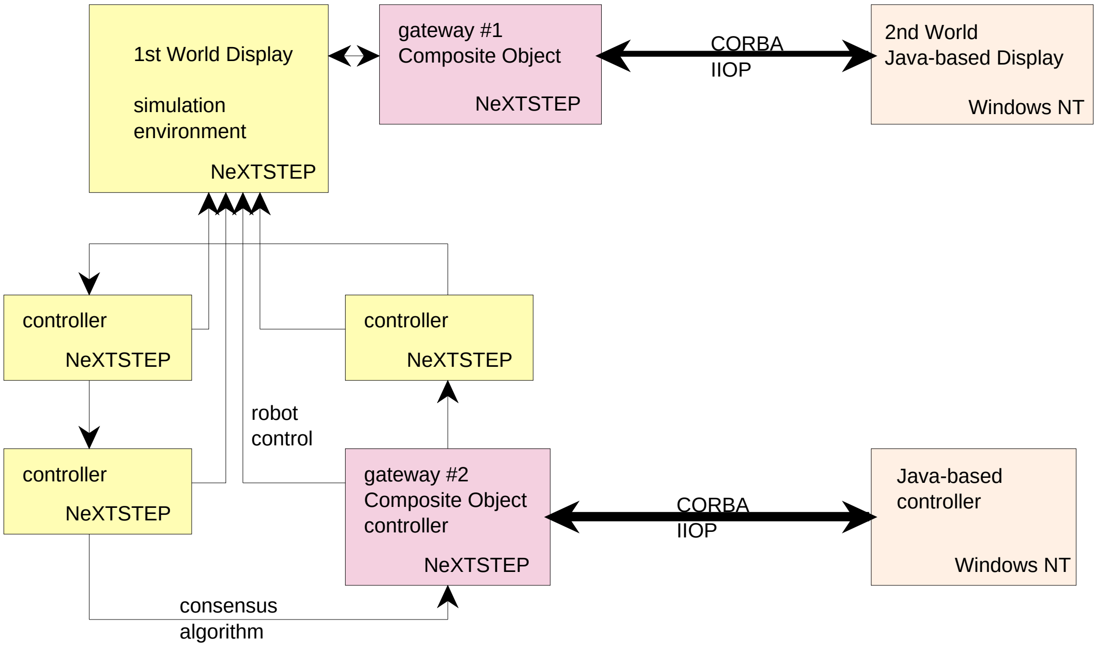


Figure 3-2: Communication Structure of the Balancing Robots

Problematic in our scenario is the potentially varying number of clients (Java applets) accessing the service. This may result in an unacceptably high load on the real-time system and, in turn, missed deadlines (indicated by stopped/crashed robots that ran out of fuel). To circumvent these problems, we use the following three strategies inside the composite objects bridging between CORBA and the soft real-time application:

1. *caching:* Due to the periodic operation of the soft real-time system, it is sufficient to read status information once per period. Thus, the composite object serves as a data cache, whose contents are valid for a limited time (the length of one period). This results in a bounded load for the soft real-time simulation.
2. *call admission:* Even when using the caching approach mentioned above, incoming CORBA requests place some load on the real-time system (e.g., scheduling overheads, processing of network protocols). To bound these loads, the CORBA ORB's dispatch rou-

tine can be executed under control of the scheduling server [Polze 97] effectively restricting its percentage of CPU usage. We have used this technique to implement call admission within the composite objects used by the application in our examples.

3. *fallback procedures:* The Java-based controller allows the interactive user to forward input to the real-time simulation. However, variations in communication and execution times due to CORBA and Java affect the soft real-time simulation's inner control loop and may result in missed deadlines. We have used the technique of fallback procedures depicted in Figure 3-3 to deal with those cases. In our approach, the fallback procedure is invoked simultaneously with the CORBA invocation of the Java control algorithm. In the case of a late reply, the fallback procedure's result is used instead of the Java algorithm's. However, in contrast to the Java algorithm, the fallback procedure's resource requirements and execution times are known and, therefore, can be factored into the real-time application's scheduling analysis.

.

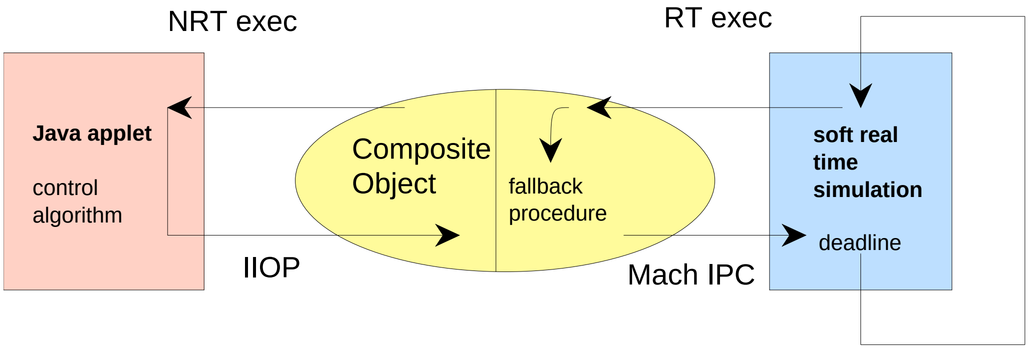
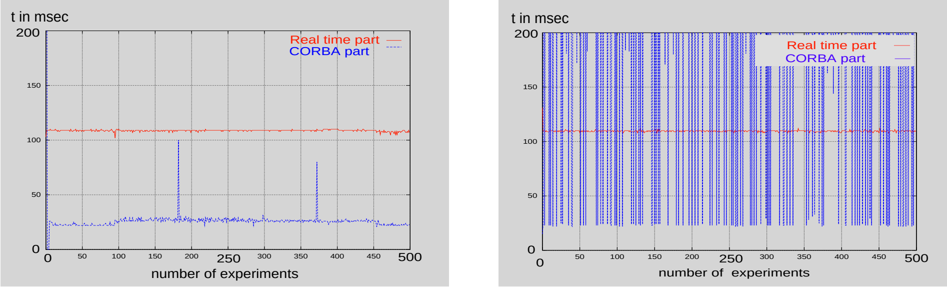


Figure 3-3: Java Controller: Execution Scheme with Fallback Procedure

In Figure 3-4, we compare the latency of CORBA requests sent to the Java-based controller with the "balancing robots" simulation's period. We see that when an unloaded system runs the Java controller, CORBA latency is well below the real-time simulation's period in most cases. Even on occasional delayed calls, the result is available before the simulation's next period begins.

In contrast, when a loaded system runs the Java controller, execution results are often late. Without applying a fallback procedure, the "balancing robots" behavior would have been significantly disturbed. However, our experiments indicate stable behavior of the simulation and demonstrate the ability of our technique to deal with variations in communication and execution times typical for middleware-based distributed computing environments.

The “balancing robots” Web-service has been implemented on top of the Mach-based NeXTSTEP 3.3 operating system. On NeXTSTEP we have used the Inter-Language Unification (ILU) 2.0 alpha 12 CORBA implementation, whereas the Java-applet connecting to our Web service has been implemented based on the Java object request broker (ORB) of OmniBroker [OOC 99]. Figure 3-5 shows screen dumps of the Java components.



unloaded Windows NT
system running external controller

loaded Windows NT
system running external controller

Figure 3-4: Execution of Java Controller Versus Simulation’s Period

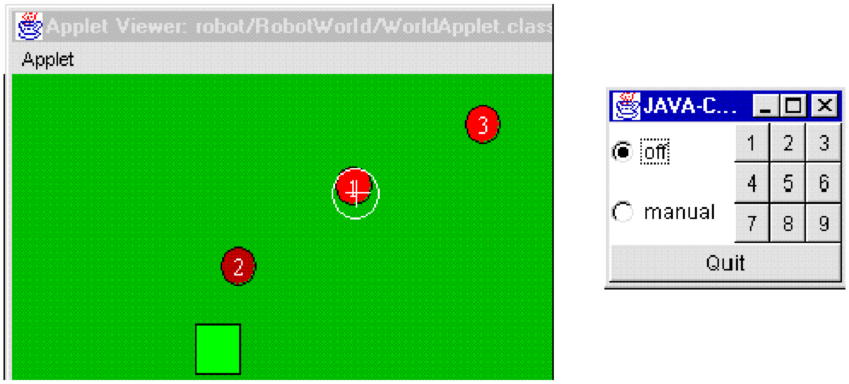


Figure 3-5: Screen Dump of Java Components for the Balancing Robots

3.2 Producer/Consumer/Viewer Example Application

In this section, we describe the effects and applicability of the composite objects approach in a minimal application setting. Our producer/consumer example studies the timing behavior of two threads that communicate via shared memory. Both threads are scheduled according to the fixed-priority scheduling policy available on the NeXTSTEP operating system. The example can be seen as a generic real-time application, where processor and resource requirements are known.

The producer/consumer application scheduled with fixed priority is a good example to study CORBA's impact on the predictability of an application's behavior. Both the producer and consumer run periodically and go through several phases. Both threads do some real work (produce/consume data for periods T_p/T_c, respectively), then both threads try to access the shared memory buffer (read/write for periods T_r/T_w, respectively), and finally they sleep (for period T_s) until their next invocation. In our example, the theoretical execution times for both producer and consumer are 60 ms per invocation.

Access to the shared memory buffer is guarded by a mutex. Since the buffer is bounded, two condition variables are used to synchronize the producer and consumer. The resource requirements and timing behavior of both the producer and consumer are simulated by computation-intensive *for*-loops, while waiting for the next invocation to be implemented using the *select()* system call. Figure 3-6 shows the overall scenario and gives the theoretical parameters for the threads' execution times.

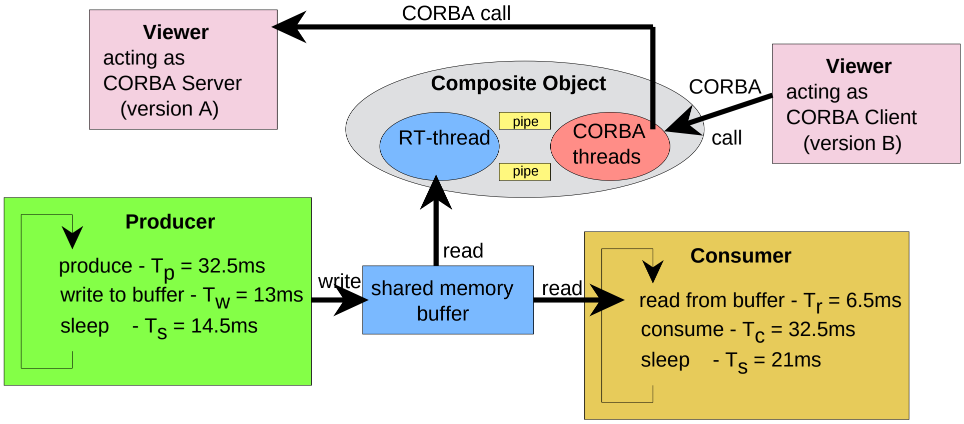


Figure 3-6: Producer/Consumer/Viewer: Overall Scenario

In addition to producer and consumer, Figure 3-6 depicts two variants of a third component—the viewer. We assume that the data which are communicated by the producer and consumer are of interest to some kind of graphical display, which is connected via a composite object and CORBA. In all experiments, the viewer component has been run remotely.

In general, there are two approaches to attach new CORBA components to a legacy application: (1) The application may either hand out data on request (i.e., act as a CORBA server, version A in Figure 3-6), or (2) it may actively send data to the CORBA components (i.e., act as a CORBA client, version B in Figure 3-6). In the first case, resource problems resulting from varying numbers of clients requesting data may show up. We want to study both cases and evaluate the timing behavior of our original application. We measure predictability of our application in terms of changes in the producer's and consumer's periodicity. Ultimately, we are going to demonstrate how the composite objects technology may keep the producer/consumer's behavior predictable—despite of and without changes to CORBA.

For our experiments, we have used the Mach-based NeXTSTEP 3.3 operating system run on a Hewlett-Packard (HP) 715/50 computer as host for producer and consumer. We have run the viewer process remotely on a SparcStation~2 computer with the Solaris 2.6 operating system. On the NeXTSTEP side, we have used the XEROX PARC Inter-Language Unification (ILU 2.0 alpha12) implementation, which provides most features specified in the CORBA standard and is interoperable with CORBA-compliant object request brokers. We have used Object Oriented Concept's (OOC's) OmniBroker 2.02 CORBA implementation on the Solaris system.

3.2.1 Version A: Composite Object as CORBA Client; Viewer as CORBA Server

In this scenario, the viewer component acts as CORBA server, and our "legacy" real-time application sends data via CORBA. Figure 3-7 compares variation in the producer/consumer's periodicity in the initial case and for a naive solution, where the consumer directly calls the viewer (without the intermediate composite object).

The right-hand diagram in Figure 3-7 demonstrates CORBA's varying communication latencies under changing loads. In our experiments we have run multiple computation-intensive processes on the viewer's host computer to simulate a slow CORBA server on a loaded system. However, even in the case of a fast, unloaded CORBA server, one may notice substantial variations in the producer/consumer's periodicity, which are caused by CORBA. Therefore, we have introduced a composite object as a mediator between the producer/consumer and viewer+CORBA.

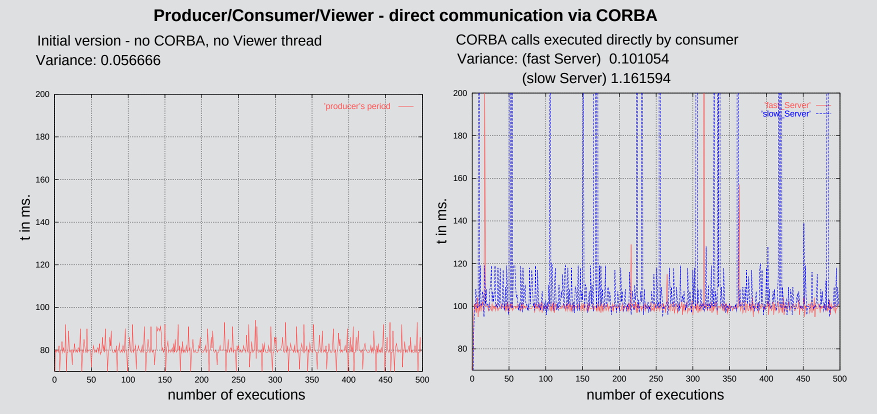


Figure 3-7: How CORBA Affects a Client Application's Timing Behavior

The variation coefficient is given in all diagrams as a measure for our test application's stable timing behavior. All diagrams have been normalized to allow easy comparison with the initial unloaded test run presented in Figure 3-7. In all CORBA-related diagrams, we compare the case of an unloaded remote computer running the CORBA server (fast server) against a loaded remote computer (slow server).

Figure 3-8 compares two implementations of the composite objects approach. First, we have used shared memory for internal communication inside the composite object (left diagram in Figure 3-8). In contrast, the right diagram in Figure 3-8 shows the application's timing behavior when pipes are used to implement the composite object's shared RT/NRT variables. In this case, the producer thread directly writes data onto a shared RT/NRT variable of the composite object and, simultaneously, into the pipe.

Both diagrams in Figure 3-8 represent a timing behavior for the producer/consumer application, which is quite similar to the original unloaded case shown in the previous figure. Thus, using the composite objects technique, we were able to decouple CORBA communication with the viewer component from the original real-time application and to save the application's timing behavior. Further studies will compare results achieved on Mach (NeXTSTEP3.3) with a similar setting on the rtLinux operating system.

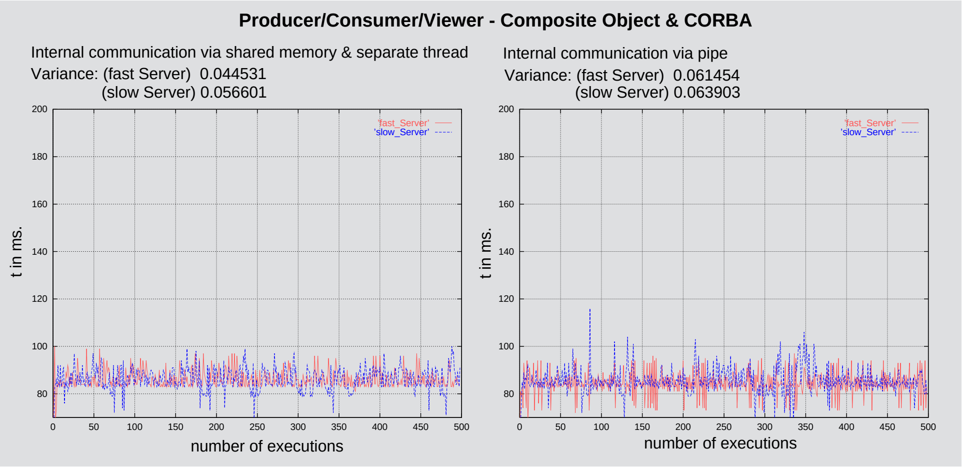


Figure 3-8: Decoupling CORBA and the Application via a Composite Object

3.2.2 Version B: Composite Object as CORBA Server; Viewer as CORBA Client

Now we want to study the effects of CORBA clients requesting data from the producer/consumer application. Again, we compare different techniques for decoupling CORBA and the application's timing behavior. Since varying numbers of CORBA clients sending requests to our producer/consumer application may impose unacceptably high loads, we must develop a scheme for call admission, which controls the behavior of the object request broker's dispatcher.

A first solution to the potential burstiness problem of CORBA requests is the introduction of a data cache between the CORBA server and the producer/consumer application.[1] In our case, the composite object's shared RT/NRT variables, with its weak memory consistency, and the periodic handler thread implement exactly as such a data cache. We should mention that caching does not completely solve the problem of bursty CORBA requests: sufficiently high numbers of clients still can create unacceptably high loads on the server's system.

In Figure 3-9, we compare the initial case where the CORBA server directly accesses the shared memory buffer used by the producer/consumer with a variant where the composite object performs caching. We have run our experiments for different numbers of CORBA clients. Figure 3-9 depicts the cases of one, two, and five CORBA clients sending requests with a frequency of 10 Hz.

[1] In this report, the term "burstiness" is used to refer to the pattern of incoming CORBA requests.

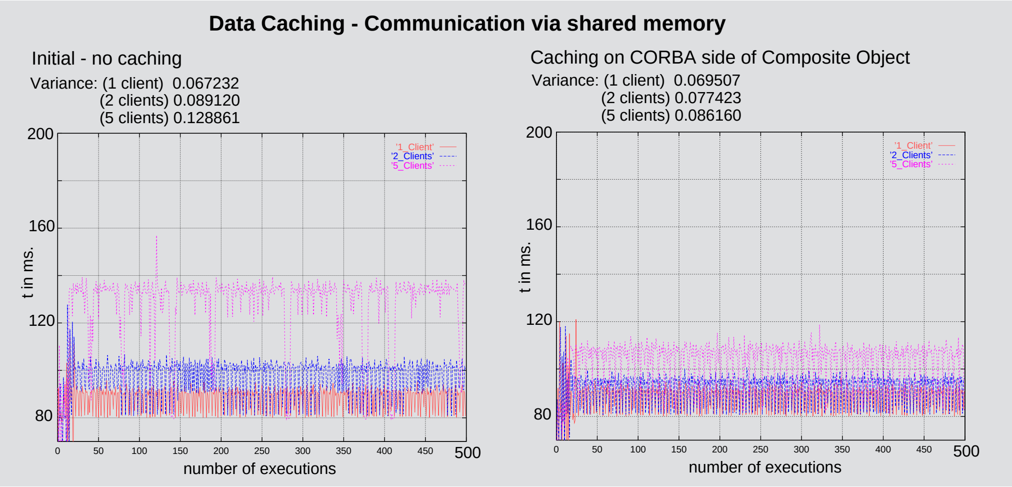


Figure 3-9: Minimizing Data Accesses: Effects of Caching

In the left diagram of Figure 3-9, the producer/consumer's periods change with the number of active CORBA clients. In contrast, as shown in the right diagram, caching in the composite object lowers the variations, but introduces some overhead and a slight, constant increase to the producer/consumer's periods. This demonstrates how the composite objects approach can be used to make an explicit tradeoff between an application's predictability and its communication performance and overall execution time.

The scheduling server [Polze 96], as developed earlier in context of the SONiC project [Polze 98], represents another way to implement call admission for a composite object acting as a CORBA server. Using fixed-priority scheduling and dynamic changes to the priority of a client's tasks, the scheduling server implements sharing of CPU resources on an a priori known basis. The scheduling server's quantum is adjustable.

For the diagrams shown in Figure 3-10, we have used the scheduling server to restrict the CPU cycles available to the main loop of the CORBA object request broker. In contrast to the left diagram, where the ORB is not restricted in its CPU usage, the middle and right diagrams show cases where a 10 ms quantum is allocated to CORBA, once every 50 ms (middle diagram), or once every 80 ms (right diagram). Thus, the remainder of the scheduling server's period of 40 ms (middle diagram) and 70 ms (right diagram) are left as undisturbed phases for the producer/consumer application on the otherwise unloaded computer.

.

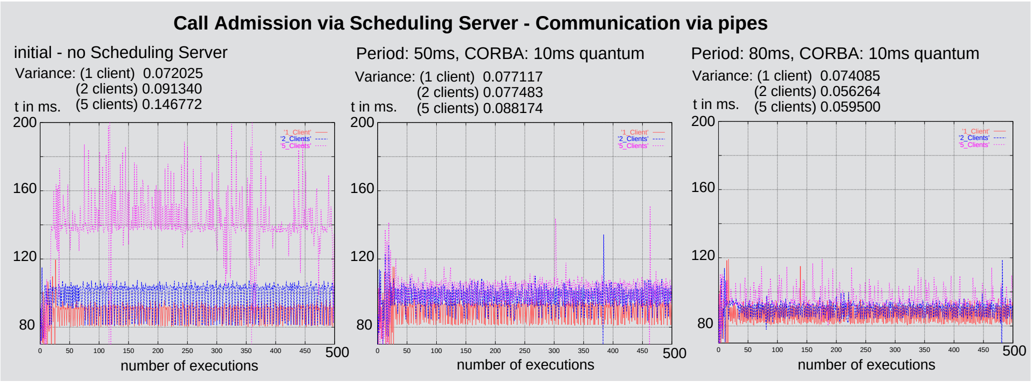


Figure 3-10: Restricting the ORB: Call Admission via Scheduling Server

The scheduling server's effect is quite obvious. In contrast to the first case, where the producer/consumer's periodicity varies largely with changing CORBA loads, the latter two cases show much more stable behavior. Again, we see a tradeoff between predictable behavior and communication latency; the length of the producer/consumer's period increases when we do not restrict the ORB in its CPU usage.

Since the scheduling server suspends the object request broker's main loop, our approach implements true call admission. Even high numbers of CORBA clients and high burstiness of traffic hardly affect the producer/consumer's timing behavior. The application of both the composite objects approach and scheduling server concept did not require any changes to the implementation of the CORBA object request broker nor the operating system's kernel.

4 Fault-Tolerance Case Studies

4.1 Observable Objects: A Technique for Robust CORBA Clients

We have developed the *Observable Object/Observer* technique for fault tolerance. Following a primary/backup approach our technique allows observation, saving of state, and automatic restart of (distributed) applications. The *Observable Object* base class can be seen as a generic interface for supervision of critical applications. Derived classes may implement observation of a program using special, application-dependent programming interfaces. We have applied our *Observable Object/Observer* technique to the Netscape Navigator Web browser to create a fault-tolerant Web client.

The *Observable Object/Observer* architecture is based on CORBA. Non-CORBA legacy applications may take advantage of our framework by using a wrapper approach. To encapsulate a non-CORBA application in a wrapper, the application must offer a communication or programming interface [e.g., UNIX signals, stdin/stdout, IPC, or different X11 mechanisms (e.g., properties)] accessible to the wrapper.

Our architecture implements a generic mechanism for monitoring CORBA objects. This mechanism is used for increasing the reliability of the monitored objects. The *Observer* can tolerate two classes of faults: crash faults of the computer nodes and crashes of the monitored objects. The *Observer* starts monitored objects on different computation nodes in a replicated manner according to the primary/backup principle. The *Observer* checks the replies of primary and backup instances to periodic *ALIVE* messages. The primary instance regularly saves its state on stable storage. When the primary fails, the *Observer* selects a backup for reconfiguration as a new primary. It does so by reading the last saved state.

To take advantage of our *Observable Object/Observer* approach, a CORBA object has to implement the observable object interface shown in Figure 4-1.

The *Observer* checks availability of all registered observable objects by calling their *state()* methods in user-specified intervals. The object reference of a crashed object becomes invalid. In this case, the CORBA run-time system raises an exception. Thus, the *Observer* can detect an *Observable Object's* failure by catching this exception.

.

```
interface ObservableObject {
        short state();
        short id();
        void become_primary(in short o_id);
        oneway void shutdown(in short o_id);
};
```

Figure 4-1: Observable Object Interface: CORBA IDL

Factory objects are used to create new backup instances of our service in case of crashes. A factory's interface, *Observable Factory,* is derived from *Observable Object*. Thus, the O*bserver* can monitor factory objects. Factories are assumed to be well tested and reliable; therefore an exception resulting from an invalid reference to a factory object is interpreted as an indication of a node (computer) failure.

We have used the *Observable Object/Observer* architecture to increase the reliability of the Netscape Navigator Web browser. In addition to a graphical user interface, most browsers offer a programming interface for controlling the browser by external programs [Zawinski 94]. This functionality is used by a wrapper, which implements the observable object interface. A Web browser is then monitored by the *Observer* by checking the availability of its wrapper. Additionally, the state of the browser (current URL) is determined by the wrapper. If a browser crashes, the wrapper terminates itself. From the *Observer's* viewpoint, a browser has failed if its wrapper is no longer available. We assume that the wrapper will not fail while the browser it encapsulates continues to run. This assumption is sound, because a wrapper is a small program that can be tested thoroughly.

Figure 4-2 depicts the communication structure in a scenario where two *Observable Objects* acting as wrappers for Netscape Navigator processes are monitored by our system. Both primary and backup are connected to the same X11 display. However, the backup's window is unmapped and becomes visible only when there is a crash fault of the primary.

The *Observable Object/Observer* concept is not limited to CORBA. It can be adopted for other middleware platforms such as distributed computing environments (DCEs) or the Distributed Component Model (DCOM).

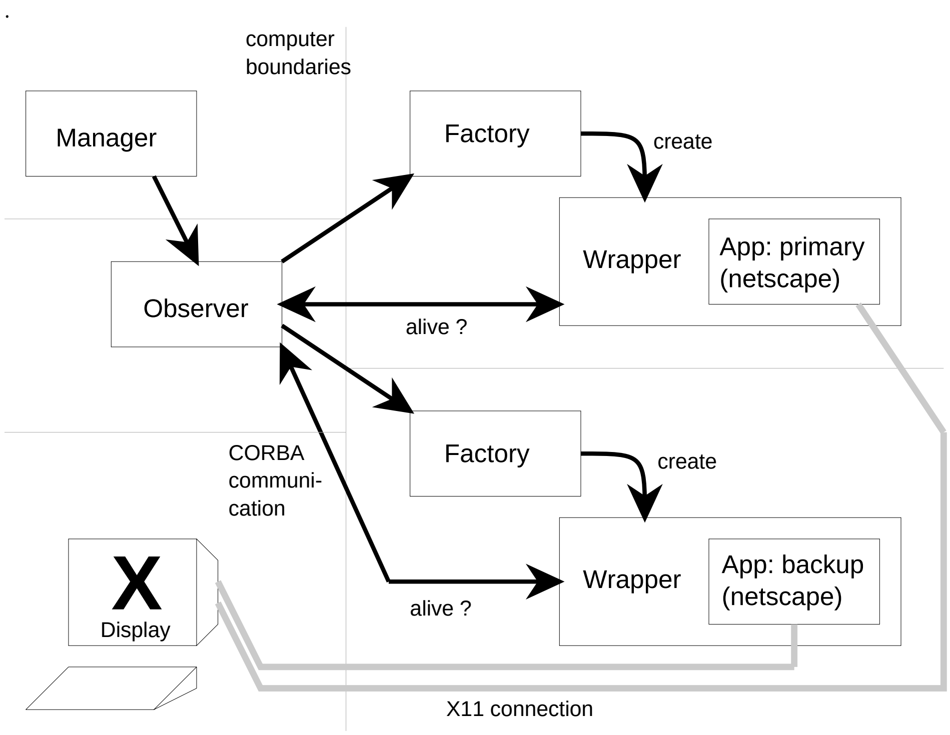


Figure 4-2: Primary/Backup Monitored by Observer

4.2 Consensus-Based Responsive Services

In many cases, it is desirable to predict the behavior of remote services invoked via CORBA. Unfortunately, the specification of parameters for service execution such as fault models, acceptable processor utilization, and timing behavior is beyond the scope of CORBA. We propose an extension of our previously developed CORE/SONiC framework [Malek 95], [Polze 98] for responsive computing by CORBA interfaces using composite objects as filtering bridges.

In the CORE/SONiC model, several parallel tasks may run on a number of interconnected computing units. SONiC implements an object-based distributed shared memory system to allow for communication between tasks. COnsensus for REsponsiveness (CORE) provides protocols, services, and scheduling strategies at the microkernel level for real-time parallel computing, even in the presence of faults. Consensus protocols are used to detect faults and to establish a system-wide global view at certain points of program execution. This serves as the basis for decisions about re-execution and coarse-grained scheduling.

CORE provides a reliable communication subsystem to SONiC, using the services of the underlying Mach microkernel operating system. The CORE/SONiC scheduling server [Polze 97], [Richling 97] provides predictable access to workstation resources and can be used to implement real-time scheduling policies such as rate monotonic scheduling or “earliest deadline first” on a standard operating system. Composite objects provide a method for creating objects that interface to CORBA and are simultaneously capable of real-time method execution. We use the composite objects approach to rebuild the concept of CORE/SONiC in a CORBA-based environment. A group of composite objects distributed over the nodes of a workstation cluster forms the execution environment for responsive services. CORBA requests are the basic units of replication and scheduling in our environment.

Search algorithms benefit highly from parallelization and load partitioning in distributed environments. Here, we present a distributed labyrinth search as an example application. Our application demonstrates structuring of a responsive service and employs consensus (group membership) for fault tolerance. Composite objects are used to implement timely behavior of the consensus protocol.

The scenario depicted in Figure 4-3 shows a client (Java based) that generates tasks—mazes with one entry and one exit point in our case—with different structures. These tasks are replicated and sent to four worker nodes. Each worker searches a subportion of the maze. Load partitioning is done statically. Each search step is reported to the client and displayed on screen, using different colors for different nodes.

A group membership protocol is run among the worker nodes. Each node communicates with its two left neighbors and sends periodic *alive* messages. The timing behavior of the protocol is ensured by executing it within real-time threads of composite objects. The scheduling server is used to allocate a fixed percentage of CPU cycles (guarantee slots) to the search processes. Our responsive service tolerates crash faults of search processes and processing nodes. The crash fault of a search process or node is detected by missing *alive* messages. In this case, the communication structure is reconfigured. Furthermore, a crashed node’s left neighbor re-executes the crashed node’s task. This scheme implements graceful degradation; a solution to the labyrinth search is found if a single node survives.

To demonstrate the behavior described above, the client application provides a user interface to send kill-messages to a particular worker node. The labyrinth search service demonstrates responsiveness (i.e., a high probability to deliver a timely solution under a given load and fault hypothesis) even in the presence of faults.

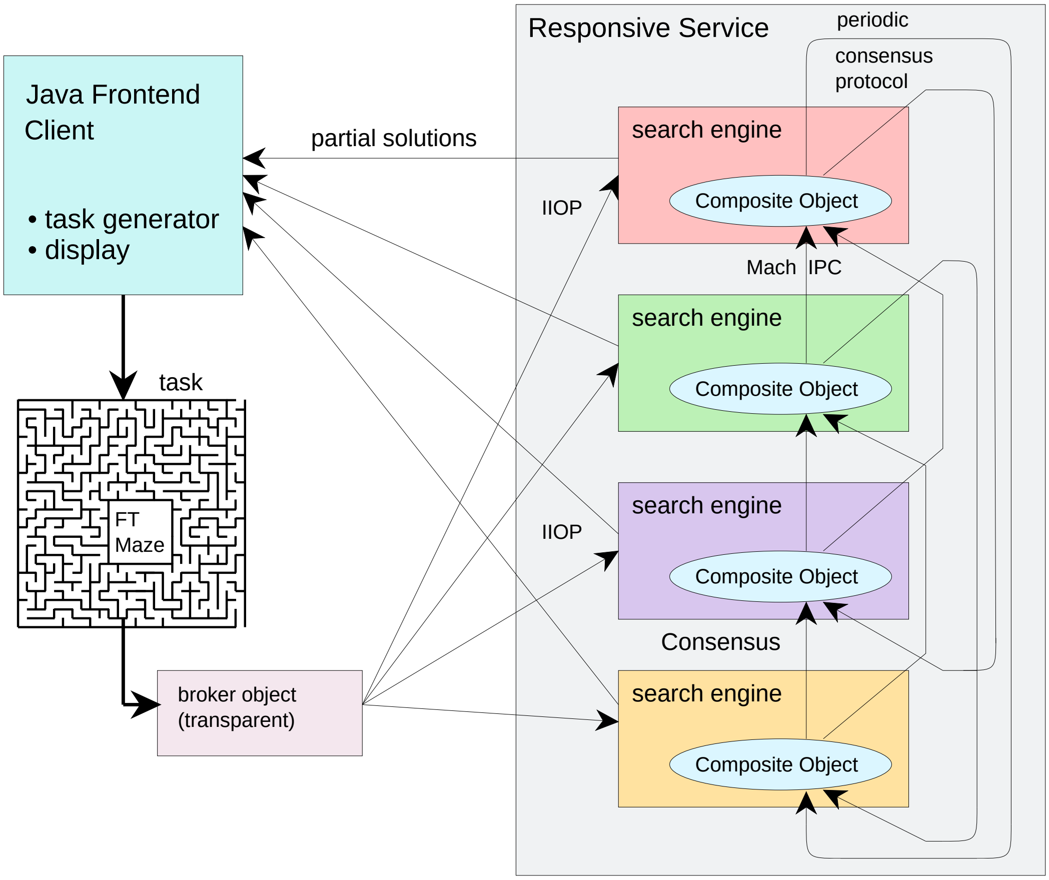


Figure 4-3: Labyrinth Search: A Responsive Service

5 Related Work

The composite objects technique can be used as a bridge between CORBA’s client/server model and execution models for replicated, fault-tolerant, real-time services. Related work describes the idea of providing real time as an additional feature in a CORBA-compliant object request broker (ORB) implementation. This approach has been proposed by the OMG Real-Time Special Interest Group.

The observable object interface allows for implementation of robust CORBA clients following the primary/backup approach to fault tolerance. Related work describes two other approaches toward a fault-tolerant CORBA:

1. special ORB implementations, which may take advantage of underlying fault-tolerant hardware
2. application of special CORBA services for fault tolerance

5.1 Real-Time CORBA

The Object Management Group (OMG) founded a Real-Time CORBA Special Interest Group (SIG) in 1996. Since then OMG has solicited technology for a real-time object request broker (ORB) that consists of fixed-priority scheduling, control over ORB resources for end-to-end predictability, and flexible communications. The request for proposal (RFP) for the fixed priority version of Real-Time CORBA 1.0 was announced in September 1997 [OMG 97].

Work at the University of Rhode Island and the MITRE Corporation deals with syntactical extensions to the CORBA interface description language (IDL) to express timing constraints [Thuraisingham 96], [Wolfe 95]. “Timed distributed method invocations” are identified as one necessary feature in a real-time distributed computing environment as well as a “global time service,” “real-time scheduling of services,” a “global priority service,” and “bounded message latency.” The “affected set priority ceiling protocol” as a combination of semantic locking and priority ceiling techniques has been proposed for concurrency control in real-time object-oriented systems [Squadrito 98].

TAO is an innovative work on Real-Time CORBA, where fixed-priority real-time scheduling is tightly integrated into the system [Schmidt 97], [Schmidt 98]. The main goal of this work is to provide end-to-end quality of service for CORBA-based applications. A list of requirements for object request broker implementations is presented; among them are resource reservation protocols, optimized real-time communication protocols, and a real-time object adapter. TAO, however, focuses on completely new CORBA-based real-time systems, rather than interfacing existing real-time systems with CORBA.

5.2 Fault-Tolerant CORBA

The idea of providing fault tolerance as an additional feature to CORBA implementations has been the focus for several research activities within the last few years. With the request for proposal issued in April 1998 [OMG 98], OMG is seeking to incorporate existing approaches for software fault tolerance into future versions of CORBA.

Electra is a CORBA ORB implementation for reliable, distributed services [Maffeis 94]. Electra extends the CORBA specification and provides group communication mechanisms, reliable multicasts, and object replication. The Electra-ORB uses services from the underlying ISIS systems [Birman 93], [Renesse 94].

ORBIX+ISIS is an extension of the commercial ORBIX [IONA] ORB implementation that introduces concepts such as object groups and group communication based on the ISIS toolkit [Birman 93]. Again, the CORBA standard has been extended to allow for introduction of fault-tolerance measures. The programmer must use special coding techniques to use the fault-tolerance features of ORBIX+ISIS.

Phoinix [Chang 97] allows for implementation of server objects following the primary/backup approach for fault tolerance. In contrast to changing the ORB, Phoinix defines a new service as part of the Object Management Architecture (OMA). Using this service, a primary object's state can be periodically transferred into the corresponding backup object. Clients communicate solely with the primary object and have to detect failures themselves. Extended skeletons, stubs, and libraries are provided to make those fault-tolerance services accessible.

6 Conclusions and Future Work

There is an urgent need to support heterogeneity and openness in today's COTS-based distributed computing environments and to enhance the computing environments by quality-of-service attributes such as responsiveness. Although CORBA is a good candidate for support of heterogeneity and openness, it does not allow specification of resource, scheduling, and timing requirements; fault models; or measures. Based on the composite objects approach, we have developed building blocks that integrate proven techniques for predictable computing with CORBA.

We have discussed the composite objects approach for the integration of CORBA with real-time computing. Data replication and weak memory consistency have been discussed as key concepts for implementing this approach and for decoupling CORBA and real-time computing. We have described implementation alternatives for the composite objects approach and have presented measurements for the overhead imposed by our implementation of this approach.

The producer/consumer/viewer and "balancing robots" example scenarios study the effects of bridging legacy applications to CORBA via composite objects. Our measurements indicate that the composite objects approach provides a promising way to retain an application's predictable timing behavior, even when communicating via CORBA. Furthermore, using the scheduling server developed earlier, we have discussed and demonstrated how call admission, as a technique for bounding the ORB's resource utilization, can be implemented without changes to the CORBA implementation and the operating system's kernel.

We have presented the CORBA-based observer approach, a generic solution for implementing reliable applications (fault-tolerant Netscape Web client in our case). Bridging the gap between (legacy) real-time fault-tolerant computing systems and CORBA-based clients is a major precondition for remote operations in many application domains such as banking, medicine, manufacturing, and booking systems.

The composite objects approach represents a viable technique to make CORBA-based cluster computing predictable in its resource utilization and timing behavior. Future work will include detailed studies of alternative implementations for composite objects on the operating systems rtLinux, Solaris, and Windows NT.

References

[Barabanov 97] Barabanov, M. & Yodaiken, V. “Introducing Real-Time Linux.” *Linux Journal 34*, an SSC Publication (February 1997). Available WWW <URL: http://www.ssc.com/ lj/issue34/0232.html>.

[Birman 93] Birman, K. P. “The Process Group Approach for Reliable Distributed Computing.” *Communications of the ACM 36*, 12 (Dec.1993): 37-53.

[Chang 97] Chang, Y.S.; Liang, D.; Lo, W.; Sheu, G.-W.; & Yuan, S.-M. “Fault-Tolerant Object Service on CORBA.” *Proceedings of International Conference on Distributed Computing Systems (ICDCS’97)*. Baltimore, MD, May 1997.

[IONA] IONA. *Orbix: Overview.* Dublin, Ireland: IONA Technologies. Available WWW <URL: http://www.iona.ie/info/products/orbix/ index.html>.

[Maffeis 94] Maffeis, S. “Flexible System Design to Support Object Groups and Object-Oriented Distributed Programming.” *Proceedings of ECOOP’93, Lecture Notes in Computer Science 791,* 1994.

[Malek 95] Malek, M.; Polze, A.; & Werner, M. “Framework for Responsive Parallel Computing in Network-Based Systems,” 335-343. *Proceedings of International Workshop on Advanced Parallel Processing Technologies*. Bejing, China, September 1995.

[OMG 97] OMG. *Responses to Real-Time CORBA RFP* (OMG Document orbos/97-06-01). Available WWW <URL:http://www.omg.org>.

[OMG 98] OMG. *Fault Tolerant CORBA Using Entity Redundancy RFP* (OMG Document orbos/98-04-01). Available WWW <URL: http://www.omg.org>.

[OOC 99] Object Oriented Concepts, Inc. *Orbacus C++/Java: An Open Architecture for Distributed Solutions*. Available WWW <URL: http://www.ooc.com/ob/> (1999).

[Polze 96] Polze, A. "How to Partition a Workstation." *Proceedings of Eight IASTED/ISMM International Conference on Parallel and Distributed Computing and Systems.* Chicago, IL, Oct. 16-19, 1996.

[Polze 97] Polze, A.; Fohler, G.; & Werner, M. "Predictable Network Computing," 423-431. *Proceedings of International Conference on Distributed Computing Systems (ICDCS'97)*. Baltimore, MD, May 1997.

[Polze 98] Polze, A. & Malek, M. "Network Computing with SONiC." *Journal on System Architecture (the Euromicro Journal) 44* (1998): 169-187. Elsevier Science B. V., Netherlands.

[Renesse 94] van Renesse, R. & Birman, K.P. "Fault-Tolerant Programming Using Process Groups," in *Distributed Open Systems*, Brazier, F. & Jones, D. (eds.). Computer Society Press, 1994.

[Richling 97] Richling, J. & Polze, A. "Scheduling Server for Predictable Computing: An Experimental Evaluation," 130-137. *Proceedings of IEEE Workshop on Middleware for Distributed Real-Time Systems and Services* (held in conjunction with Real-Time Systems Symposium). San Francisco, CA, Dec. 2-5 1997.

[Schmidt 97] Schmidt, D. C.; Gokhale, H.; Harrison, T. H.; & Parulkar, G. "High-Performance End System Architecture for Real-Time CORBA." *IEEE Communications Magazine 14,* 2 (February 1997).

[Schmidt 98] Schmidt, D. C.; Levine, D.; & Mungee, S. "The Design and Performance of Real-Time Object Request Brokers." *Computing Communications 21,* 4 (April 1988).

[Sha 94] Sha, L.; Rajkumar, R.; & Sathaye, S. S. "Generalized Rate-Monotonic Scheduling Theory: Framework for Developing Real-Time Systems." *Proceedings of the IEEE 82,* 1 (January 1994).

[Soley 98] Soley, R. M. “Are Real-Time Objects Ready for Prime Time” (keynote speech). *Proceedings of First IEEE International Symposium on Object-Oriented Real-Time Distributed Computing.* Kyoto, Japan, April 1998.

[Squadrito 98] Squadrito, M.; Esibov, L.; DiPippo, L. C.; Wolfe, V. F.; Cooper, G.; Thuraisingham, B.; Krupp, P.; Milligan, M.; & Johnston, R. “Concurrency Control in Real-Time Object-Oriented Systems: The Affected Set Priority Ceiling Protocols,” 96-105. *Proceedings of First IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*. Kyoto, Japan, April 1998. IEEE Comp. Soc. Press, ISBN 0-8186-8430-5.

[Thuraisingham 96] Thuraisingham, B.; Krupp, P.; & Wolfe, V. “On Real-Time Extensions to Object Request Brokers” (position paper). *Proceedings of Second Workshop on Object-Oriented Real-Time Dependable Systems (WORDS).* Laguna Beach, CA, February 1996. IEEE Comp. Soc. Press, ISBN 0-8186-7570-5.

[Werner 96] Werner, M. & Malek, M. *The Unstoppables: Responsiveness by Consensus* (HUB Informatik-Berichte No.: 90/97, ISSN 0863-095 90). Berlin, December 1996.

[Wolfe 95] Wolfe, V. F.; Black, J. K.; Thuraisingham, B.; & Krupp, P. “Real-Time Method Invocations in Distributed Environments.” *Proceedings of the International High Performance Computing Conference.* December 1995.

[Zawinski 94] Zawinski, Jamie. *Remote Control of UNIX Netscape*. Mountain View, CA: Netscape Communications Corporation. Available WWW <URL: http://www.home.netscape.com/newsref/std/x-remote.html> (Dec. 1994).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (leave blank)	2. REPORT DATE May 1999	3. REPORT TYPE AND DATES COVERED Final

4. TITLE AND SUBTITLE

Building Blocks for Achieving Quality of Service with Commercial Off-the-Shelf (COTS) Middleware

5. FUNDING NUMBERS

C — F19628-95-C-0003

6. AUTHOR(S)

Andreas Polze

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

8. PERFORMING ORGANIZATION REPORT NUMBER

CMU/SEI-99-TR-001

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

HQ ESC/DIB
5 Eglin Street
Hanscom AFB, MA 01731-2116

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

ESC-TR-99-001

11. SUPPLEMENTARY NOTES

12.a DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified/Unlimited, DTIC, NTIS

12.b DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

To date, most of the fault-tolerant, real-time systems have been implemented in embedded settings, and there is an urgent need to open up this type of computing technology to a larger number of people who use heterogeneous distributed computing environments. Today's transportation, manufacturing, and communication systems require the integration of multiple embedded real-time control systems with standard distributed computing environments in a predictable fashion. Humboldt University has developed the concept of *composite objects* as a filtering bridge between standard middleware platforms and software frameworks providing services with certain quality-of-service (QoS) guarantees. Current research focuses on the Common Object Request Broker Architecture (CORBA) middleware platform; however, composite objects are also applicable to platforms like the Distributed Component Model (DCOM) and distributed computing environments (DCEs). Key concepts in Humboldt's approach are analytic redundancy, noninterference, interoperability, and adaptive abstraction. These concepts originated in SEI work on the Simplex architecture and have been reapplied to extend the reach of commercial off-the-shelf (COTS) software technologies into demanding application settings (such as those found in military and industrial applications). Here, we discuss building blocks and techniques for fault-tolerant, real-time applications based on CORBA.

14. SUBJECT TERMS

commercial off-the-shelf (COTS) software, Common Object Request Broker Architecture (CORBA), composite objects, Distributed Component Model (DCOM), distributed computing environment, fault-tolerant systems, real-time systems

15. NUMBER OF PAGES

35

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL