Data Replication and Weak Memory Consistency: Predictable CORBA Interactions with Composite Objects[1]

Andreas Polze and Jan Richling
Humboldt-University of Berlin
Department of Computer Science
10099 Berlin, Germany
email: {apolze,richling}@informatik.hu-berlin.de

Abstract

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 requirements — such as manufacturing, process control, and transport systems. Furthermore, those attributes make CORBA an interesting candidate for responsive (fault-tolerant, real-time) cluster computing.

However, the specification of timing behavior and quality-of-service parameters like communication latency and acceptable processor utilization is beyond the scope of today's CORBA. 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 implementation of ''Composite Objects''. We evaluate the timing behavior of ''Composite Objects'' and demonstrate the value of our technique to predictable CORBA-based cluster computing.

Key words: Predictable Cluster Computing, CORBA, Composite Objects, Resource Reservation.

1. Introduction, Motivation

The Common Object Request Broker Architecture (CORBA) [1] is a widely-accepted, standardized system integration framework based on distributed object technologies. CORBA solves a number of problems regarding programming in distributed environments and workstation clusters. However, the specification of an application's timing behavior and quality-of-service parameters like communication latency and acceptable processor utilization is completely beyond the scope of today's CORBA.

Within this paper we evaluate the ''Composite Objects'' approach [2], a means for predictable integration of CORBA with real-time requirements. ''Composite Objects'' rely on the principle of ''separation of concerns''. Replicated data and weakly consistent memory protocols are used to decouple CORBA communication latency and the application's timing behavior. By making the tradeoff between an application's predictability and CORBA's communication latency explicit, ''Composite Objects'' allow the programmer to use CORBA for responsive (fault-tolerant, real-time) cluster computing. Our measurements show stable and predictable timing behavior of a distributed demo application using ''Composite Objects'' under varying numbers of communication partners and changing loads.

The remainder of the paper is organized as follows: Section 2 gives an overview over related work. In Section 3 we describe the ''Composite Objects'' approach in some detail. Section 4 discusses implementation issues, whereas Section 5 presents measurements based on our example application. Section 6 concludes the paper.

2. Related Work

The Object Management Group (OMG) has founded a ''Realtime CORBA'' special interest group (SIG). A forthcoming white paper on real-time CORBA is on the status of an ''Initial Review Draft'', as of November, 15, 1996 [3]. The Request for Proposal (RFP) for the fixed priority version of Realtime CORBA 1.0 has been announced in September 1997. The RFP solicits technology for a Realtime Object Request Broker (ORB) comprising: fixed priority scheduling, control over ORB resources for end-to-end predictability, and flexible communications. However, it is not clear to what extend RT-CORBA will be compatible with the standard CORBA specification.

Work at the University of Rhode Island and the MITRE Corporations deals with syntactical extensions to CORBA IDL to express timing constraints [4]. A proposed implementation uses CORBA context declarations to specify deadlines for transmission of data between client and server and for execution of the server's method. ''Timed distributed method invocations'' are identified as one necessary feature in a real-time distributed computing environment. A ''Global Time Service'', ''Real-Time Scheduling of Services'', a ''Global Priority Service'' and ''Bounded Message Latency'' are named as prerequisites for the proposed approach.

[1] appeared in Proc. of 10th IASTED/ISMM Intl. Conf. on P arallel and Distributed Comp. and Syst., Las Vegas, USA, Oct. 1998.

TAO is a pioneering work on RT-CORBA where fixed priority real-time priority scheduling is tightly integrated into the system [5]. Main goal of this work is to provide end-to-end Quality-of-Service qualities 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. However, TAO rather focuses on completely new, CORBA-based real-time systems, than on interfacing an existing real-time system with CORBA.

3. Composite Objects

Our concept of ‘‘Composite Objects’’ is an approach to integrate responsive (fault-tolerant, real-time) and CORBA computing into a single object-based framework. It allows the programmer to make an explicit trade-off between an application’s predictable resource utilization and its communication latency and make implementation details visible and explicit which are usually hidden and abstracted away by CORBA. There are three design rules which form the basis for our implementation of ‘‘Composite Objects’’.

1) NON-INTERFERENCE: general purpose computing and time-aware responsive computing should not burden each other.
2) INTER-OPERABILITY: services exported by general purpose computing objects and by responsive computing objects can be utilized by each other.
3) ADAPTIVE ABSTRACTION: lower level information and scheduling details are available for real-time objects but transparent to non-real-time objects.

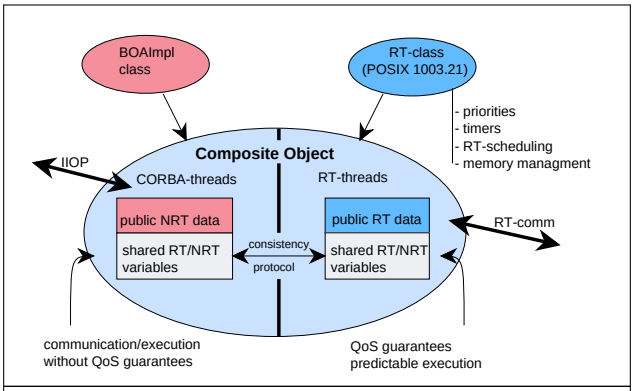

Fig. 1*: Structure of a Composite Object*

Since ‘‘Composite Objects’’ provide functionality to react on CORBA method invocations, they can be seen as descendants of a class which implements Basic Object Adaptor functionality (`BOAImpl` in CORBA jargon). On the other hand, ‘‘Composite Objects’’ have the capability to create real-time programming abstractions like prioritized threads and real-time communication channels. As depicted in Fig. 1, they can be seen as descendants of a class which implements real-time servers.

‘‘Composite Objects’’ consist of a real-time 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 [6] between real-time and non-real-time (CORBA) computing, so that the non-real-time part cannot violate the real-time scheduling rules [7] that are needed by the real-time part.

Data replication is the key to independent data accesses from real-time and non-real-time threads inside a ‘‘Composite Object’’. We split an object’s data into a part, which is statically locked in memory to fulfill real time scheduling assumptions (RT data), and a part which can be paged out and is treated like a standard non-real time user’s process’ data (NRT data). A pair of RT/NRT variables can be viewed as replicated variable. ‘‘Composite Objects’’ implement a consistency protocol to update both replicas and create the impression of a shared variable.

4. Implementation Issues

The consistency protocol for shared RT/NRT variables as well as resource allocation (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 [8] operating systems we have used pipes, shared buffers and message queues (Mach IPC) for our implementationof ‘‘Composite Objects’’ as shown in Fig. 2.

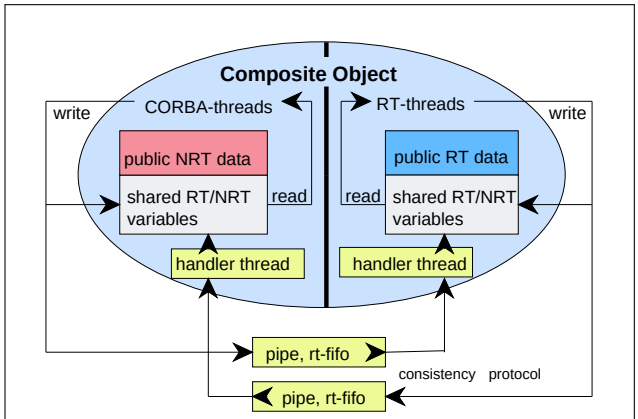

Fig. 2*: Consistency Protocol inside a Composite Object*

Fig. 2 describes the data flow during 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 result not only 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 copy periodically 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.

Alternatives to the implementation outlined here include usage of *shared memory* or *message passing* (Mach IPC) for the data transfer between CORBA part and real-time part of the ‘‘Composite Object’’. In our experiments, we have evaluated ‘‘Composite Objects’’ implementations based on pipes and shared memory. Another design alternative is an event-triggered consistency protocol in contrast to the time-triggered, periodical protocol described above. In that case a `write` operation would include signaling arrival of a new data value to the handler thread on the opposite side of the ‘‘Composite Object’’. We do have implemented consistency protocols following both schemes.

5. Example Application

Our producer/consumer example studies the timing behavior of two threads, communicating via shared memory. Both threads are scheduled according to the *fixed priority* scheduling policy available on the Mach operating system. The example can be seen as a generic ‘‘real-time’’ application, where processor and resource requirements are known a priori. Producer and consumer run periodically as depicted in Fig. 3 (60`ms` period) and go through several phases: produce/consume, communicate, sleep.

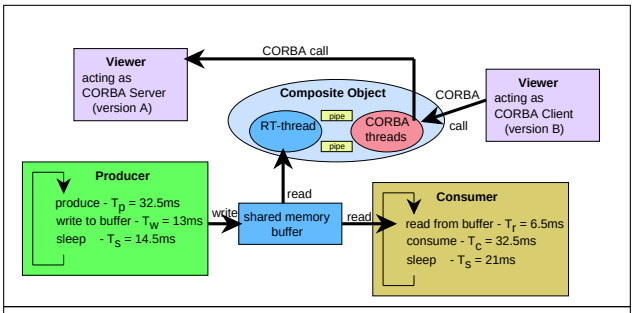

Fig. 3*: Producer/Consumer/Viewer - overall scenario*

In addition to producer and consumer, Fig. 3 depicts two variants of a third component - the viewer. We assume, that the data which is communicated by producer and consumer is of interest to some kind of graphical display - connected via a ‘‘Composite Object’’ and CORBA. In all experiments, the viewer component has been run remotely, connected via CORBA.

In general, there exist two approaches to attach new CORBA components to a legacy application: the application may either actively send data to the CORBA components (i.e., act as a CORBA client, version A in Fig. 3) or it may hand out data on request (i.e., act as a CORBA server, version B in Fig. 3). 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 producer’s and consumer’s periodicity.

For our experiments we have used the Mach-based NeXTSTEP 3.3 operating system run on a 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) CORBA implementation. We have used OOC’s OmniBroker 2.02 CORBA implementation on the Solaris system.

Version A: Viewer as CORBA Server

In this scenario the viewer component acts as CORBA server, our ‘‘legacy’’ real-time application sends data via CORBA. Fig. 4 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’’).

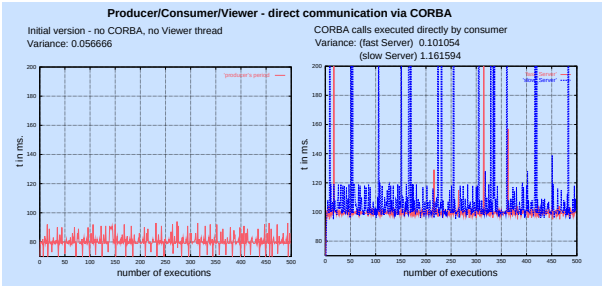

Fig. 4*: How CORBA affects a client application’s timing behavior*

The right hand diagram in Fig. 4 impressively 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 and compared against a fast server on an unloaded system. However, even in the case of a fast, unloaded CORBA server one may notice substantial variations in the producer/consumer’s periodicity - caused by CORBA. The variance is given in all diagrams as a measure for our test-application’s stable timing behavior.

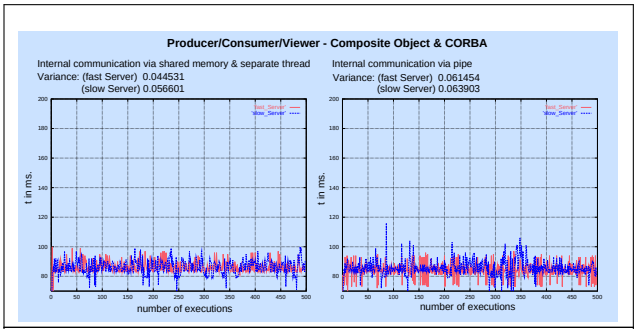

Fig. 5*: Decoupling CORBA and the application via a Composite Object*

Fig. 5 compares two implementations of ‘‘Composite Objects’’ acting as mediator between Producer/Consumer and CORBA+Viewer which use shared memory and pipes for internal communication. Both diagrams in Fig. 5 demonstrate the decoupling of CORBA latency from the application’s timing behvior.

Version B: Viewer as CORBA Client

Now, we want to study the effects of CORBA clients requesting data from the producer/consumer application. Since varying numbers of CORBA clients sending requests to our producer/consumer application may impose in-acceptable high loads, we have to develop a scheme for call admission - which controls the behavior of the Object Request Broker's dispatcher.

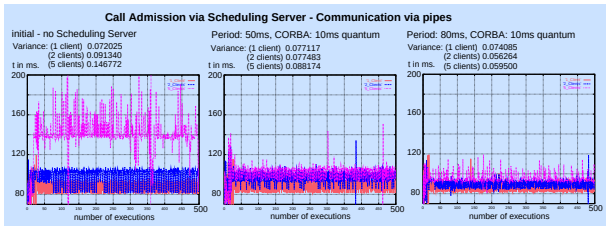

Fig. 6*: Restricting the ORB: Call admission via scheduling server*

The ''Scheduling Server'' [9] as developed earlier in context of the SONiC project [10] can be used to implement call admission for a ''Composite Object'' acting as CORBA server. Using fixed-priority scheduling and dynamic changes to a client's task's priority, the ''Scheduling Server'' implements sharing of CPU resources on a a priori known basis. The ''Scheduling Server's'' quantum is adjustable.

In the diagrams shown in Fig. 6, we have used the ''Scheduling Server'' to restrict the CPU cycles available to the CORBA Object Request Broker's (ORB) main loop. In contrast to the left diagram, where the ORB is not restricted in its CPU usage, the middle and right diagram show cases where a 10ms quantum is allocated to CORBA, once every 50ms (middle diagram) or once every 80ms (right diagram). Thus, the remainder of the ''Scheduling Server's'' period of 40ms (middle diagram) and 70ms (right diagram) are left as undisturbed phases for the producer/consumer application on the otherwise unloaded computer.

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 do hardly affect the producer/consumer's timing behavior. We should mention, that the application of both, the ''Composite Objects'' and ''Scheduling Server'' concepts did not require any changes to the implementation of the CORBA Object Request Broker nor the operating system's kernel.

6. Conclusions

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

The ''producer/consumer/viewer'' example application uses a minimalistic, legacy, real-time application as basis to study the effects of bridging 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 technique for bounding the ORB's resource utilization can be implemented without changes to the CORBA implementation and the operating system's kernel.

''Composite Objects'' represent 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

[1] OMG; *The Common Object Request Broker: Architecture and Specification*; Object Management Group, Inc., Framingham, MA, USA, 1995.

[2] A.Polze, L.Sha; *Composite Objects: Real-Time Programming with CORBA*; Proc. of 24th Euromicro Conference, Network Computing WS, Vaesteras, Sweden, August 25-27, 1998.

[3] J.McGoogan (ed.); *Realtime CORBA - A White Paper - Issue 1.0*; OMG Realtime Platform SIG, Initial Review Draft, November, 1996.

[4] B.Thuraisingham, P.Krupp, and V.Wolfe; *Position Paper: On Real-Time Extensions to Object Request Brokers*; in Proc. of 2nd WS on Object-Oriented Real-Time Dependable Systems (WORDS), Laguna Beach, CA, USA, February, 1996.

[5] D.C.Schmidt, A.Gokhale, T.H.Harrison, D.Levine, and C.Cleeland; *TAO: a High-performance Endsystem Architecture for Real-time CORBA*; RFI response to the OMG SIG on RT-CORBA, 1997.

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

[7] L.Sha, R.Rajkumar, S.S.Sathaye; *Generalized Rate-Monotonic Scheduling Theory: A Framework for Developing Real-Time Systems*; Proc. of IEEE, Vol. 82, No. 1, January 1994.

[8] M.Barabanov, V.Yodaiken; *Introducing Real-Time Linux*; Linux Journal, issue 34, an SSC publication, February 1997, see also: http://www.ssc.com/ lj/issue34/0232.html.

[9] J.Richling, A.Polze; *Scheduling Server for Predictable Computing: an Experimental Evaluation*; in Proc. of IEEE WS on Middleware for Dist. RT Systems and Services, held in conjunction with RTSS, San Francisco, USA, December 2-5, 1997.

[10] A.Polze, M.Malek *Network Computing with SONiC*; in Journal on System Architecture 44 (1998) (the Euromicro Journal), special issue on Cluster Computing, pp.: 169-187, Elsevier Science B.V., Netherlands, 1998.