Data Replication and Weak Memory Consistency — Predictable CORBA Interactions with Composite Objects

9 April 1998

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 behaviour and quality-of-service parameters like communication latency and acceptable processor utilization is beyond the scope of today's CORBA.

Within this paper, 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’’. Based on an example application, we evaluate the timing behaviour of different implementations of ‘‘Composite Objects’’ and demonstrate the value of our technique to predictable CORBA-based cluster computing.

1. Introduction, Motivation

Clusters of networked off-the-shelf workstations allow users to benefit from high performance parallel computing, increased accessible resources, and scalability. A number of practical problems, however, jeopardize the benefits. Cumbersome methods to program parallel applications limit the effective parallelism. Uncoordinated resource management and unpredictable loads impede the performance gains: Workstations will be loaded above tolerable levels. No predictions or guarantees about a program's timing behaviour can be given.

The Common Object Request Broker Architecture (CORBA) [OMG95] [Soley95] 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 behaviour and quality-of-service parameters like communication latency and acceptable processor utilization is completely beyond the scope of today's CORBA. An ongoing effort within the OMG concerns real time CORBA (RT-CORBA) [McGoogan96], which is a future extension to the OMG Object Management Architecture to support a wide spectrum of distributed, real time, fault-tolerant systems.

In an earlier project we have developed the ‘‘Shared Objects Net-interconnected Computer (SONiC)’’ platform for execution of parallel programs in networked environments [Polze98a]. Besides an object-based distributed shared memory system and a remote execution service, SONiC provides a scheduling service which allows for execution of an application's sub-tasks in a predetermined fashion [Polze97]. SONiC is based on the Mach operating system.

Within this paper we discuss the ‘‘Composite Objects’’ approach, a means for predictable integration of CORBA with real-time requirements. ‘‘Composite Objects’’ rely on the principle of ‘‘separation of concerns’’. We use a technique similar to SONiC's scheduling server to restrict CORBA in its processor utilization — thus leaving the responsive application a predetermined share of CPU cycles. Replicated data and weakly consistent memory protocols are used to decouple CORBA communication latency and the application's timing behaviour. 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.

The ‘‘Composite Objects’’ concept has been introduced elsewhere [Polze98b]. We have studied several implementation alternatives for ‘‘Composite Objects’’. Here, we describe and evaluate those implementations based on a ‘‘Producer/Consumer/Viewer’’ example application. We present measurements showing stable and predictable timing behaviour of our distributed application under changing loads. Also, we study the effect of varying numbers of communication partners (clients) in a CORBA-based setting. We demonstrate how ‘‘Composite Objects’’ can be used to implement call admission and to restrict load on the side of an server object.

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

Our ‘‘Composite Objects’’-approach provides a means for achieving predictable behaviour in CORBA-based cluster computing applications. Here, we summarize related work with a slightly different objective: We discuss the idea of providing real time as an additional feature in a CORBA-compliant Object Request Broker implementation. This approach has been proposed by the OMG Real Time Special Interest Group.

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 [McGoogan96]. 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.

ANSA (Advanced Networked Systems Architecture) [ANSA] is an open, collaborative research program managed by the british company APM Limited. The initial programming interface supported by ANSA follows ‘‘Open Distributed Processing’’ (ODP) standards. However, the current project ‘‘Jet’’ was started to provide a CORBA API for ANSA. The project’s objective is to extend CORBA with real-time multi-media functionality, such as streams, signals, explicit binding and QoS.

Work at the University of Rhode Island and the MITRE Corporations deal with syntactical extensions to CORBA IDL to express timing constraints [Thuraisingham96] [Wolfe95]. A proposed implementation uses four CORBA context declarations (_after, _before, _by, _execute), 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.

TAO is a pioneering work on RT-CORBA where fixed priority real time priority scheduling is tightly integrated into the system [Schmidt97a] [Schmidt97b]. 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 - Data Replication and Predictable Execution

Our concept of ‘‘Composite Objects’’ is an approach to integrate responsive (fault-tolerant, real-time) and CORBA computing into a single object-based framework. ‘‘Composite Objects’’ allow the programmer to make an explicit tradeoff between an application’s predictable resource utilization and its communication latency. Therefore, ‘‘Composite Objects’’ make implementation details visible and explicit which are usually hidden and abstracted away by CORBA. We advocate a "co-existence and the-need-to-know" approach to reach this goal. There are three design rules.

1) NON-INTERFERENCE: we should create an environment in which general purpose computing and time-aware responsive computing will not burden each other.
2) INTER-OPERABILITY: the services exported by general purpose computing objects and by responsive (real-time, fault-tolerant) computing objects can be utilized by each other.
3) ADAPTIVE ABSTRACTION: lower level information and scheduling actions needed by responsive computing is available for real-time objects but transparent to non-real-time objects.

The implementation of "Composite Objects" follows these design rules.

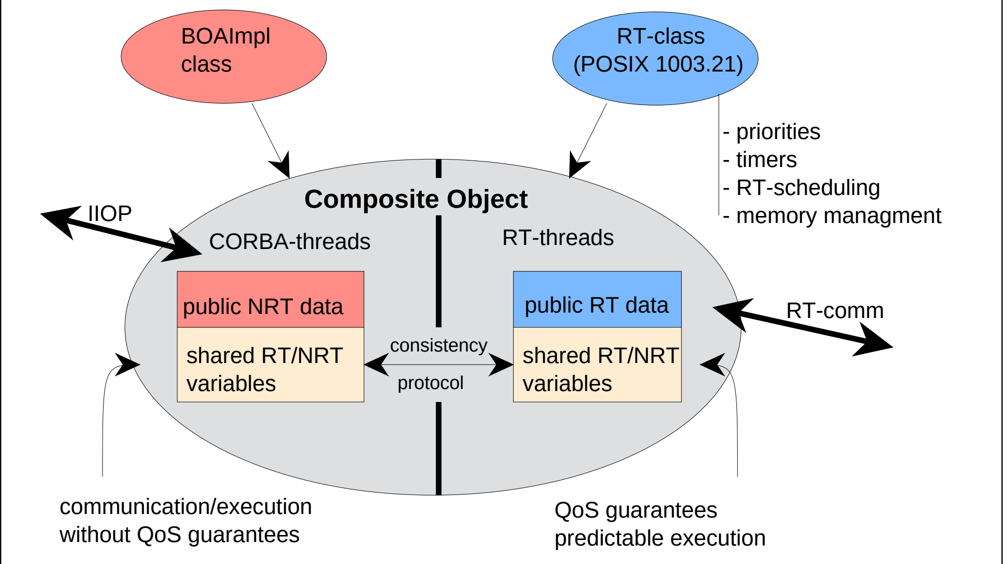

Fig. 1: *Structure of a Composite Object*

Since "Composite Objects" provide functionality to react on CORBA method invocations, they can be seen as descendants of a class which implements Object Adaptor functionality (Basic Object Adaptor Implementation — `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. Thus, they can be seen as descendants of a class which implements real-time servers. Fig. 1 shows the overall structure of a composite object.

Composite Objects consist of a real-time and a non-real-time part. Design time and run-time guarantees can be given for execution of real-time methods. In contrast, methods in the non-real-time part of the composite object are executed following a best effort approach. Those non-real-time methods correspond to the well-known standard CORBA method invocations. It is the task of the "Composite Object" to ensure that the service to non-real-time clients will be carried out in best effort without adversely affecting the real-time computation. The principle of ADAPTIVE ABSTRACTION as realized here allows to hide all implementation details from the CORBA user, whereas scheduling and timing information is accessible for the real-time part of a composite object.

‘‘Composite Objects’’ establish timing firewalls [Polze97] between real-time and non-real-time (CORBA) computing, so that the non-real-time part cannot violate the real-time scheduling rules [Sha94] that are needed by the real-time part. This approach allows us to implement the idea of NON-INTERFERENCE. Additionally, ‘‘Composite Objects’’ allow for INTER-OPERABILITY between real-time and non-real-time computing by providing data transfer mechanisms in such a way that real-time accesses to the data cannot be blocked by non-real-time functions.

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. Those data structures are accessible by both, real time and non-real time threads of a ‘‘Composite Object’’ and allow for predictable interactions. Therefore, a ‘‘Composite Object’’ is rather a design concept than a contiguous data structure at runtime of a program.

4. Implementation Issues and Measurements

The implementation of the consistency protocol for shared RT/NRT variables as well as implementation of resource allocation (CPU cycles, memory) and call admission for CORBA clients are crucial for bringing the concept of ‘‘Composite Objects’’ to life. We have used the Mach (NeXTSTEP 3.3) and rtLinux [Barabanov96] operating systems for our implementations of ‘‘Composite Objects’’. To realize the consistency protocols, we have used the operating system’s concepts of pipes, shared buffers and message queues (Mach IPC). Fig. 2 shows the communication structure required by the consistency protocol inside a ‘‘Composite Object’’.

Let us discuss the data flow during `read/write` accesses to a ‘‘Composite Object’s’’ shared RT/NRT variables in some detail. As depicted in Fig. 2, `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 (pipes or rt-fifos in our case). On both sides of the ‘‘Composite Object’’ do exist handler threads, which 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.

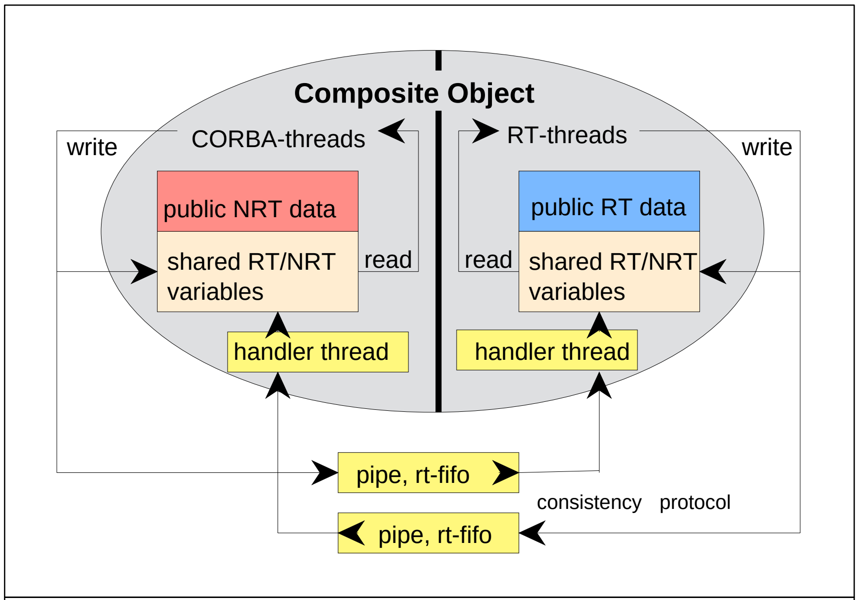

Fig. 2*: Consistency Protocol and Communication inside a Composite Object*

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.

‘‘Composite Objects’’ act as bridge between threads on one node in a workstation cluster and clients sending service requests from elsewhere in the cluster. In the isolated case, offline analysis and real-time scheduling techniques can be applied to local threads to give guarantees about the local threads’ behaviour. However, in contrast to the local case remote CORBA clients sending requests to the ‘‘Composite Object’’ can create inacceptable resource and load requirements. The number of clients may vary and is - in general - not a priori known.

There are two locations inside a ‘‘Composite Object’’ which can be used to decouple CORBA communication and predictable, real-time method execution. The concept of replicated, shared RT/NRT variables ensures, that blocking accesses on CORBA side will not disturb the real-time computing. Since the handler thread’s period is adjustable, the programmer can make an explicit tradeoff between data consistency and load requirements (CPU cycles).

Restricting the CORBA Object Request Broker (ORB) in its ability of servicing client requests is another technique which is needed if we assume highly varying numbers of CORBA clients and a high burstiness of service requests. We have succesnfully used our concept of a ‘‘Scheduling Server’’ as described in [Richling] to implement call admission for CORBA clients of a ‘‘Composite Object’’. The idea here is, to allocate a fixed percentage of CPU cycles to the dispatcher thread which de-multiplexes CORBA requests (the ORB's mainloop). This way, we can make sure that CORBA's resource requirements are bounded - independently of the number of clients,

Both, the implementation of data consistency scheme for shared RT/NRT variables and our call admission approach impose some overhead on the performance of "Composite Objects". We have used the scenario shown in Fig. 3 to evaluate a "Composite Object's" behaviour.

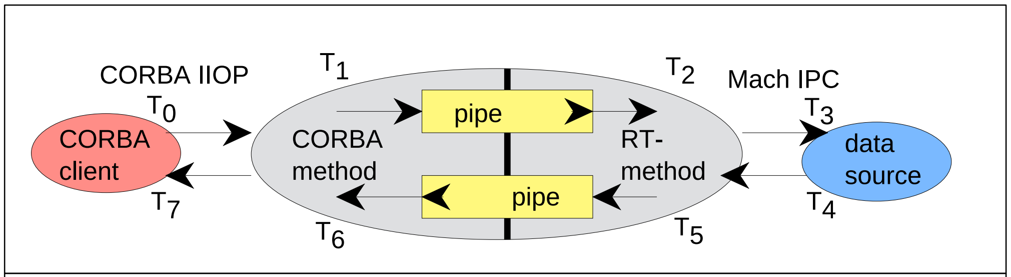

Fig. 3*: Measuring Overhead inside a Composite Object*

Fig. 3 shows a CORBA client which is accessing a real-time data source via a "Composite Object". With points in time T_0 to T_3 we measure CORBA communication latency, the "Composite Object's" internal latency and real-time communication latency for the request sent out by the CORBA client, respectively. Time T_4 to T_7, respectively, measure the same effects for the transmission of a return value from the real-time data source back to the CORBA client.

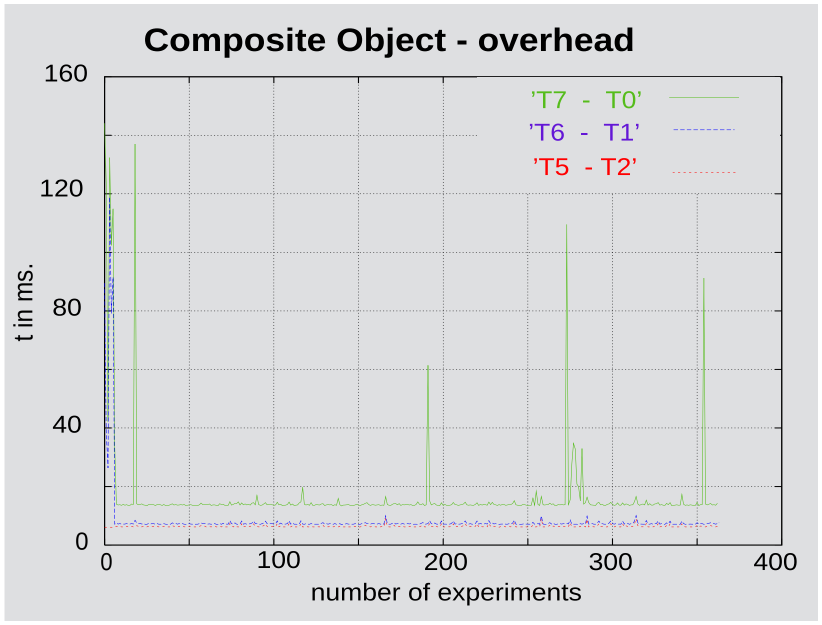

Fig. 4*: Overheads imposed by Composite Object*

Fig. 4 shows communication delays for the scenario mentioned above. The lower, red curve represents the difference $T_5 - T_2$, basically the time needed for Mach Interprocess Communication between "Composite Object" and data source. The middle, blue curve shows the difference $T_6 - T_1$. The distance between blue and read curve reprents the overhead imposed by our implementation of a “Composite Object”. It is roughly 1ms. One should notice, that both, blue and red curve are pretty stable, indicating that introduction of a “Composite Object” as mediator will not disturb an application’s predictable timing behaviour.

In contrast, the green curve, representing the difference $T_7 - T_0$ - the overall communication delay including CORBA - shows clearly visible peaks. Although the average CORBA communication latency is around 12ms (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.

In Fig. 4 we did not try to represent the difference $T_4 - T_3$ which basically denotes the time for accessing a single memory location.

5. Producer/Consumer/Viewer Example

Within this section we want to study the effects and applicability of the “Composite Objects” technology in a minimalistic application setting. Our producer/consumer example studies the timing behaviour 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 [1], where processor and resource requirements are a priori known.

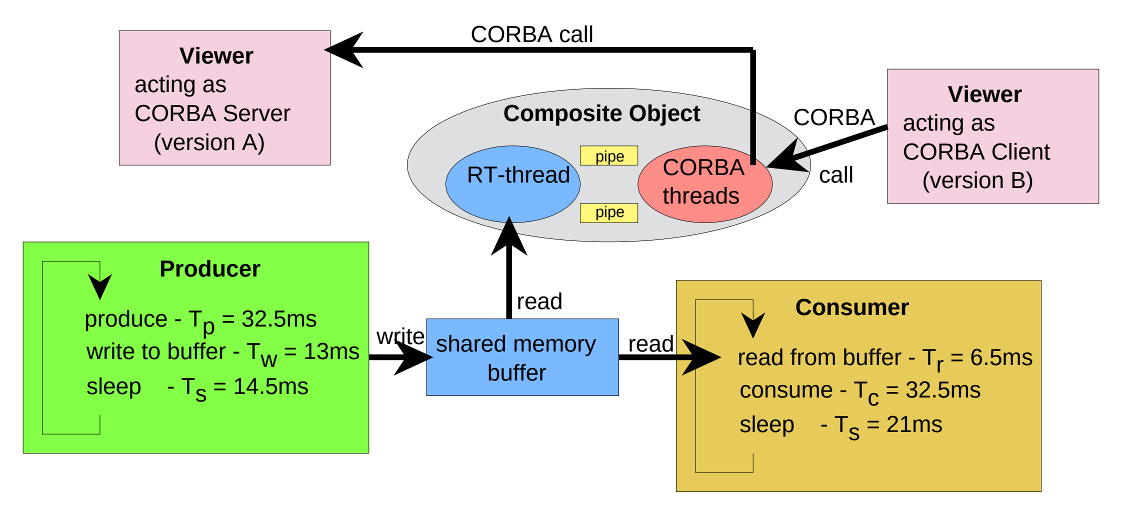

Fig. 5: *Producer/Consumer/Viewer - overall scenario*

Both, 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 60ms per invocation.

[1] Although we are not using a “pure” real-time operating system in the experiments described here, the producer/consumer application sheduled with fixed priority is a good example to study CORBA’s impact on an application’s predictable behaviour.

Accesses to the shared memory buffer are guarded by a mutex. Since the buffer is bounded, two condition variables are used to synchronize producer and consumer. Resource requirements and timing behaviour of both, producer and consumer are simulated by computation-intensive `for`-loops. Waiting for the next invocation has been implemented using the `select()` system call. Fig. 5 shows the overall scenario and gives the theoretical parameters for the threads' execution times.

In addition to producer and consumer, Fig. 5 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. 5) or it may hand out data on request (i.e., act as a CORBA server, version B in Fig. 5). In the latter case resource problems resulting from varying numbers of clients requesting data may show up. We want to study both cases and evaluate the timing behaviour of our original application. We measure predictability of our application in terms of changes in producer's and consumer's periodicity. Ultimately, we are going to demonstrate, how the ‘‘Composite Objects’’ technology may help to save the producer/consumer's predictable behaviour - despite of and without changes to CORBA.

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. In contrast, we have used OOC's OmniBroker 2.02 CORBA implementation on the Solaris system.

Version A: Composite Object as CORBA client - viewer as CORBA server

In this scenario the viewer component acts as CORBA server, our ‘‘legacy’’ real-time application sends data via CORBA. Fig. 6 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 Fig. 6 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. 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. Therefore, we have introduced a ‘‘Composite Object’’ as mediator between producer/consumer and viewer+CORBA.

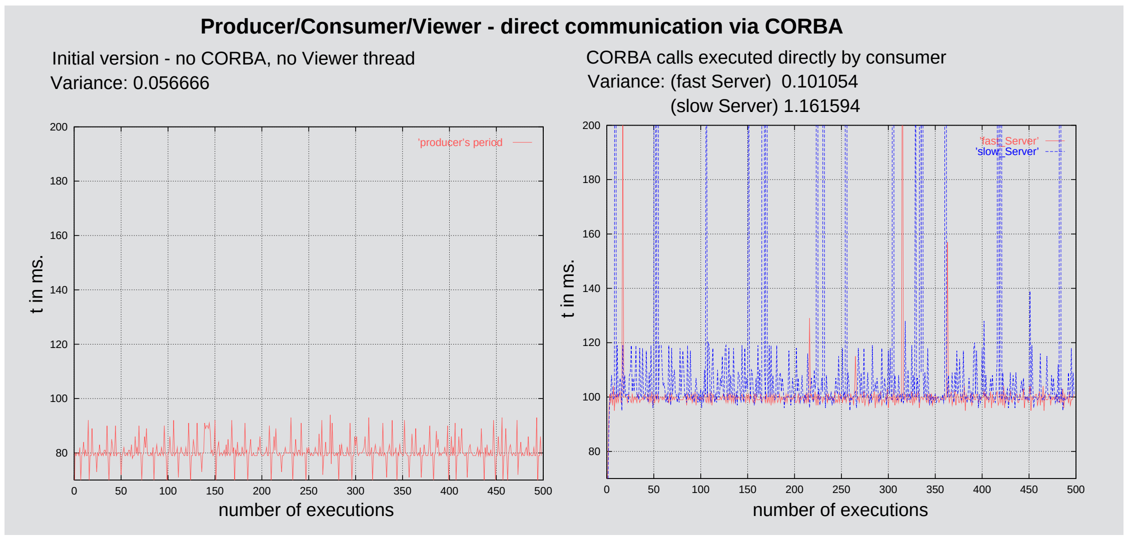

Fig. 6: *How CORBA affects a client application's timing behaviour*

The variation coefficient is given in all diagrams as a measure for our test-application's stable timing behaviour. One should also notice, that all diagrams are normalized to allow easy comparison with the initial, unloaded test run presented in Fig. 6. 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).

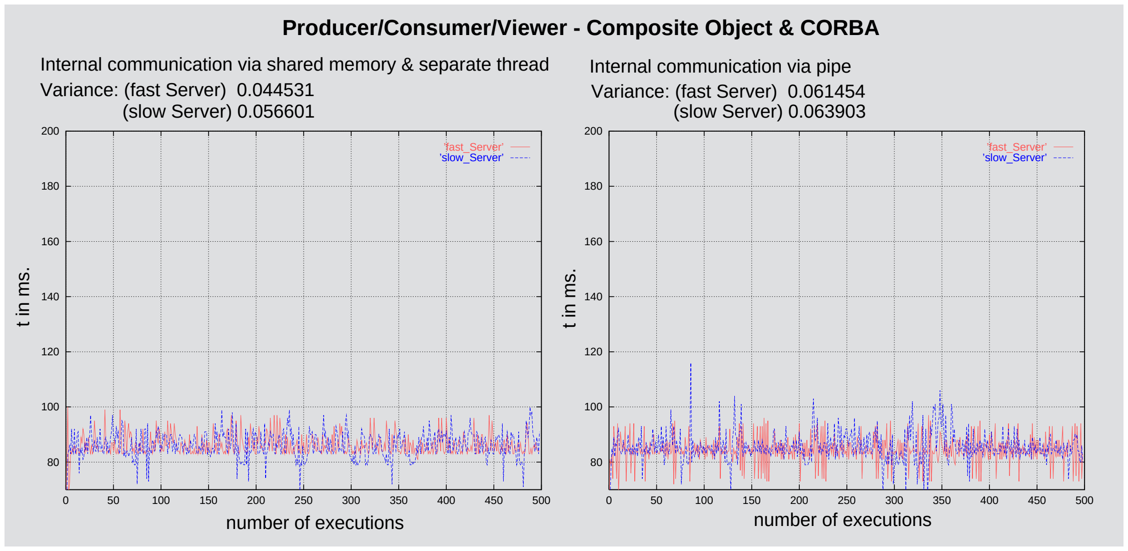

Fig. 7: *Decoupling CORBA and the application via a Composite Object*

Fig. 7 compares two implementations of the "Composite Objects" approach. First, we have used shared memory for internal communication inside the "Composite Object".

In contrast, the right diagram in Fig. 7 shows the application's timing behaviour in the case where 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 Fig. 7 represent a timing behaviour 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 behaviour. Further studies will compare results achieved on Mach (NeXTSTEP 3.3) with a similar setting on the rtLinux operating system.

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 behaviour. 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 behaviour of the Object Request Broker’s dispatcher.

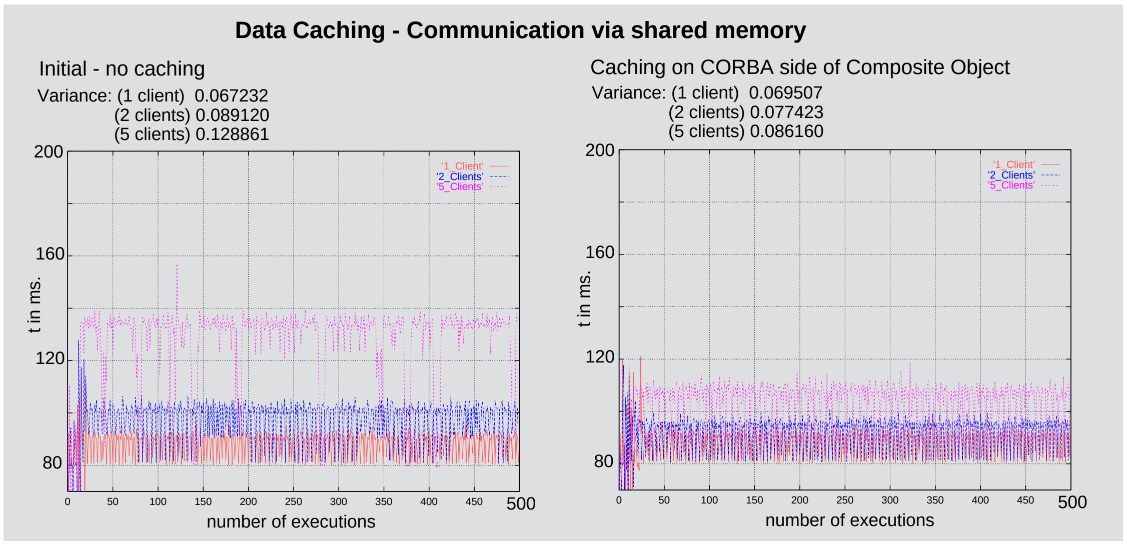

Fig. 8: *Minimizing data accesses - effects of caching*

A first solution to the potential burstiness problem of CORBA requests is the introduction of a data cache between CORBA server and the producer/consumer application. In our case, the ‘‘Composite Object’s’’ shared RT/NRT variables with its weak memory consistency and the periodic handler thread implement exactly 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 in-acceptable high loads on the server’s system.

In Fig. 8 we compare the initial case - where the CORBA server directly accesses the shared memory buffer used by producer/consumer with a variant where the ‘‘Composite Object’’ performs caching. We have run our experiments for different numbers of CORBA clients, Fig. 8 depicts the cases of 1, 2, and 5 CORBA clients sending requests with a frequency of 10Hz.

One may notice, that in the left diagram in Fig. 8 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 producer/consumer's periods. This demonstrates how ‘‘Composite Objects’’ can be used to make an explicit tradeoff between an application's predictability and its communication performance and overall execution time.

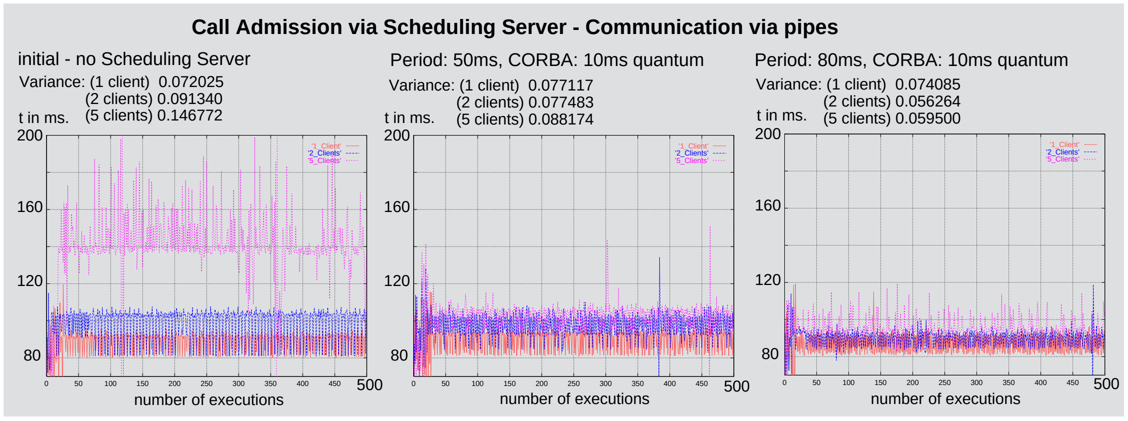

Fig. 9*: Restricting the ORB: Call admission via scheduling server*

The ‘‘Scheduling Server’’ [Polze96] as developed earlier in context of the SONiC project [Polze98a] represents another way 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.

For the diagrams shown in Fig. 9, 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.

The ‘‘Scheduling Server's’’ effect is quite obvious: in contrast to the first case, where producer/consumer's periodicity varies largely with changing CORBA loads, the latter two cases show much more stable behaviour. Again, we see a tradeoff between predictable behaviour and communication latency - as producer/consumer's period lengths increase in cases where 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 do hardly affect the producer/consumer's timing behaviour. 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 behaviour - 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 behaviour. Future work will include detailed studies of alternative implementations for ‘‘Composite Objects’’ on the operating systems rtLinux, Solaris, and Windows NT.

Acknowledgements

The authors want to thank their colleague Matthias Werner for valuable comments and helpful hints on earlier versions of this paper.

References

[ANSA] see multiple articles at http://www.ansa.co.uk/ANSA/index.html

[Barabanov96] 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.

[McGoogan96] J.McGoogan (ed.); *Realtime CORBA - A White Paper - Issue 1.0*; OMG Realtime Platform SIG, Initial Review Draft, November, 1996.

[OMG95] OMG; *The Common Object Request Broker: Architecture and Specification*; Object Management Group, Inc., Framingham, MA, USA, 1995.

[Polze98a] 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.

[Polze98b] A.Polze, K.Wallnau, and D.Plakosh; *CORBA in Real-Time Settings: A Problem from the Manufacturing Domain*; to appear in Proceedings of 1st IEEE International Symposium on Object-oriented Real-time distributed Computing, Kyoto, Japan, April 1998.

[Polze97] A.Polze, G.Fohler, M.Werner; *Predictable Network Computing*; Proceedings of International Conference on Distributed Computing Systems (ICDCS'97), Baltimore, May 1997.

[Polze96] A.Polze; *How to Partition a Workstation*; in Proceedings of Eight IASTED/ISMM Intl. Conf. on Parallel and Distributed Computing and Systems, Chicago, USA, October 16-19, 1996.

[Richling97] J.Richling, A.Polze; *Scheduling Server for Predictable Computing: an Experimental Evaluation*; in Proceedings of IEEE Workshop on Middleware for Distributed Real-Time Systems and Services, held in conjunction with Real-Time Systems Symposium, pp.: 130-137, San Francisco, USA, December 2-5, 1997.

[Schmidt97a] 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 Special Interest Group on Real-time CORBA, 1997.

[Schmidt97b] D.C.Schmidt, A.Gokhale, T.H.Harrison, G.Parulkar; *A High-performance Endsystem Architecture for Real-time CORBA*; IEEE Communications Magazine, Vol. 14, No. 2, February 1997.

[Sha94] L.Sha, R.Rajkumar, S.S.Sathaye; *Generalized Rate-Monotonic Scheduling Theory: A Framework for Developing Real-Time Systems*; Proceedings of the IEEE, Vol. 82, No. 1, January 1994.

[Soley95] R.M.Soley (ed.); *Object Management Architecture Guide*; Third Edition, John Wiley & Sons, New York, 1995.

[Thuraisingham96] B.Thuraisingham, P.Krupp, and V.Wolfe; *Position Paper: On Real-Time Extensions to Object Request Brokers*; in Proceedings of Second Workshop on Object-Oriented Real-Time Dependable Systems (WORDS), February 1996, Laguna Beach, CA, USA, IEEE Comp. Soc. Press, ISBN 0-8186-7570-5.

[Wolfe95] V.F.Wolfe, J.K.Black, B.Thuraisingham, P.Krupp; *Real-time Method Invocations in Distributed Environments*; Proceedings of the International High Performance Computing Conference, December, 1995.