Responsive Computing with CORBA

25 November 1997

Andreas Polze and Miroslaw Malek

{apolze,malek}@informatik.hu-berlin.de
Department of Computer Science
Humboldt University of Berlin
Axel-Springer Str. 54a
10099 Berlin, Germany

ABSTRACT

The Common Object Request Broker Architecture (CORBA) is a widely-accepted, standardized system integration framework based on distributed object technologies. CORBA is focused on facilitating general computing environments and does not explicitly address the needs of responsive (fault-tolerant, real-time) computing. Therefore, the question remains how to extend today's CORBA implementations for support of responsive computing.

In this paper, we propose a CORBA-based distributed framework for responsive execution of CORBA method invocations. The framework exploits consensus for synchronization, reliable communication, and fault diagnosis among replicated server objects.

Within the replicated server objects, we use the "Composite Objects" technique for predictable integration of CORBA and responsive computing. We present a methodology for creating objects which interface to CORBA while simultaneously supporting fault-tolerance and real-time requirements.

1. Introduction, Motivation

The Common Object Request Broker Architecture (CORBA) [OMG95] [Soley95] is a widely-accepted, standardized system integration framework based on distributed object technologies. An ongoing effort within the Object Management Group (OMG) concerns real-time CORBA (RT-CORBA) [McGoogan96], which is future extension to the OMG Object Management Architecture to support distributed, real-time systems. Besides real time there are other quality-of-service attributes whose support by CORBA would be highly desirable. Quality of service has many faces but certain properties such as security, dependability and real time seem to be of greater importance to the majority of users than others.

Combining the two requirements for dependability and real-time leads us to responsive computing whose primary objective is to maximize responsiveness. Responsiveness - as a quantitative measurement - is the probability of correct execution on time under a given fault and load hypothesis. The quest for responsiveness has been stimulated by transportation, manufacturing and communication systems; and with ever-increasing reliance on computers by the entire society, the need for responsiveness will continue to grow.

The implementation of a CORBA-based framework for responsive computing seems to be rather straightforward. However, there are some challenges. The key feature of CORBA is that users can create application systems by integrating existing heterogeneous hardware and software components. As long as each object adheres to the interface standards provided by CORBA, clients and servers do not need to be concerned with others' locations, hardware, operating system and other implementation details. The fundamental assumption made by CORBA is that such implementation information can be and should be abstracted away since it does not affect the correctness of computation, modeled as values produced by computations. This assumption is indeed sound for general purpose data processing.

On the other hand, responsiveness, i.e., probability of correct execution on time even in the presence of faults, of an application depends not only on the values output by computation but also the times at which outputs are generated. These in turn are dependent on load and failure hypotheses as well as the hardware/software system. Techniques such as replication, consensus (e.g., by voting), and checkpointing require detailed knowledge about the internals of an application's objects. Physical locations, hardware speeds, workloads, and the schedulers used by operating systems and applications can affect timing and therefore may affect the responsiveness level.

Thus, a possible extension of CORBA for responsive computing is not a simple refinement of the existing CORBA model. Rather, it is about the whole nature of CORBA specifications: what should and should not be abstracted away and how to do it. Also, we have to ask ourself what kind of hooks should be incorporated into current CORBA implementations to support responsive computing.

The rest of the paper is organized as follows: Section 2 discusses desirable extensions to today's CORBA. In Section 3 we present a CORBA-based object architecture for responsive computing. Section 4 discusses implementation issues, whereas Section 5 describes related work. Finally, Section 6 concludes the paper.

2. What is missing in today's CORBA?

We briefly describe the Mach-based CORE/SONiC (COnsensus for REsponsiveness/Shared Objects Net-interconnected Computer) [Malek95][Polze97a] framework for Responsive Cluster Computing to derive a list of requirements for a special CORBA implementation for responsive computing. You will notice that our list of requirements goes far beyond the scope of today's CORBA. Since the ''roll your own''-approach (implementation of an extended CORBA ORB) clearly violates the idea of COTS-based computing, we will end up with an object architecture which interfaces with CORBA on one hand and provides qualities of responsive computing on the other. We use ''Composite Objects'' [Polze97c] as basis for implementation.

The CORE/SONiC framework integrates the objectives of responsive systems — fault tolerance and timeliness — with issues associated with parallel computing in distributed environments — idle-time computing, ease-of-use, and communication avoidance. Based on a number of interconnected PC's and workstations CORE/SONiC provides a layered software architecture and uses a hierarchical model for each processing element in the system.

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. Replicated C++ objects (''Data Objects'') provide the programmer's interface to SONiC. SONiC supports a fork/join-style of parallel programming.

COnsensus for REsponsiveness (CORE) provides protocols, services, and scheduling strategies at the microkernel level for real-time parallel computing even in presence of faults. The programmer's interface to CORE consists of so-called ''Consensus Objects''. Consensus protocols are used to establish at certain points of program execution a system-wide global view in consensus objects distributed over the nodes of the CORE/SONiC system.

CORE provides a reliable communication subsystem to SONiC, using the services of the underlying Mach microkernel operating system. The CORE/SONiC ''Scheduling Server'' [Polze97b] provides predictable access to workstations resources and can be used to implement real-time scheduling policies like ''Rate Monotonic Scheduling'' or ''Earliest Deadline First'' on a standard operating system.

Because of its object-oriented nature, SONiC's replicated data objects and CORE's consensus objects are a natural place to interface CORE/SONiC and CORBA. Those objects are implemented in a replicated fashion, distributed over the nodes of a cluster. However, if we just replace CORE's communication protocols by CORBA's synchronous method invocation capabilities, we experience a number of difficulties. Critical issues are:

- assignment of priorities: a client's CORBA method invocation should be executed with a priority lower than those used for implementation of consensus and consistency protocols — but CORBA has no notion of priorities attached to method invocations.
- management of shared resources (memory, I/O channels): CORE/SONiC assumes that memory for all components of an application is allocated up front and is

locked against paging during runtime. It has to be investigated, how CORBA server objects (depending on ORB/OA implementation) can be forced to follow this rule — at least there is no solution to this problem specified by CORBA.

- priority inversion problems: at least the CORE consensus protocols have to be implemented in a fashion that they can work completely independent from any client-initiated CORBA interactions. Otherwise, a blocking CORBA interaction may violate the system assumptions — CORBA does not deal with synchronization between simultaneous method invocations.
- predictable performance: to implement the concept of a responsive service, performance of the CORBA gateway to the service should be predictable with respect to method execution time and communication latency. This is highly dependent on the ORB (preferable multithreaded) implementation — in fact, CORBA does not deal with quality-of-service parameters of the underlying transport media.

3. An Object Architecture for Responsive Computing

Our concept of "Composite Objects" provides a methodology for creating objects which interface to CORBA and are simultaneously capable of in-time fault-tolerant method execution using replication in space and time. Composite objects provide a unified interface which will be identical to the existing CORBA definitions from the view point of CORBA clients.

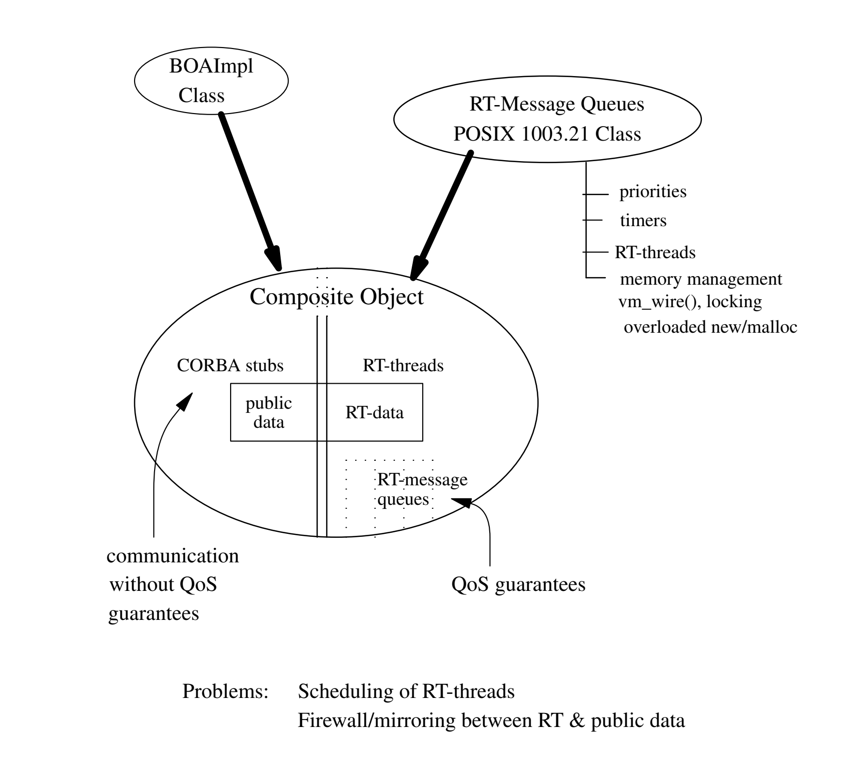

Fig. 1*: Structure of a Composite Object*

Therefore, "Composite Objects" provide a path for integration of responsive computing with CORBA — without requiring changes to the CORBA specification and current implementations.

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 "time triggered methods" and "real-time service methods", respectively. Composite Objects as realized here allow 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.

A composite object's real-time services include:

- Time-triggered methods: those methods encapsulate time-triggered actions such as periodic sampling of a sensor device.
- Real time service methods: those methods encapsulate data driven service at a given level of priority and with a worst case execution time, e.g., a particular filtering method in a filter object.
- Data flow service: those methods encapsulate real-time communication service such as the real-time publisher and subscriber[Rajkumar95].

We use "Composite Objects" to re-build the concept of CORE's consensus objects and SONiC's data objects. A group of composite objects distributed over the nodes of a workstation cluster forms the execution environment for "Responsive Services". Within consensus objects, "time-triggered" methods are used to implement protocols for synchronization, fault-detection, and reconfiguration. Data objects employ "real-time service" methods to implement weak consistency protocols for replicated data. Both, consensus and data objects use the composite object's "data flow services" for predictable communication.

Since data and consensus objects are also accessible through the CORBA interface, a special "broker" object can be introduced to transparently replicate a client's CORBA request. Then, according to a previously chosen fault model, the request is sent to a group of consensus objects. In turn, those objects execute the request in a parallel fashion as "real-time service" methods. All method instances have a common view onto their shared data — implemented within data objects. Fig. 2 shows a view onto our execution framework and highlights interactions between consensus objects.

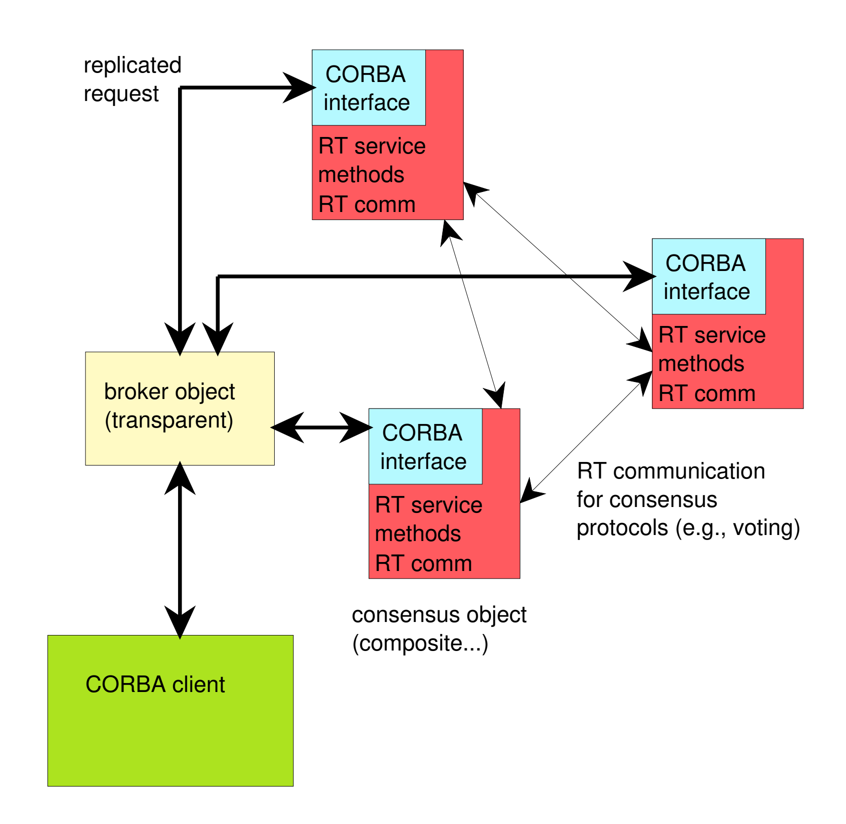

Fig. 2: *Execution framework for Responsive Services*

In our system and computation model, we consider a system to be a number of independent processing elements (PEs) connected by communication lines which allow any pair of PEs to send messages between themselves in a known, bounded time. The consensus objects' "data flow service" is used for this purpose. A processing element consists of the hardware, including a linearly monotonic clock that can give an impression of real-time, and the software — represented as composite objects — required to execute a schedule of method invocations (threads) as well as to interface with the communication subsystem. A scheduler, which accepts threads along with their priorities and deadlines, is included in the composite objects concept. It attempts to complete threads before their deadlines, preempting lower priority threads if needed.

The components, i.e., the PEs of such a system tend to run apart, even if no fault occurs. This is true especially for the PEs' clocks. For that reason, we consider a cyclic computation model, which is an extension of the bulk-synchronous model (BSP) [Valiant90]. For purposes of fault tolerance, the same thread may run redundantly on different nodes. An execution cycle consists of two phases: one phase of computation and one phase of reaching consensus among the nodes on the calculated results (see [Malek94]). Fig. 3 shows an execution cycle in our system employing "consensus objects" and "shared data objects".

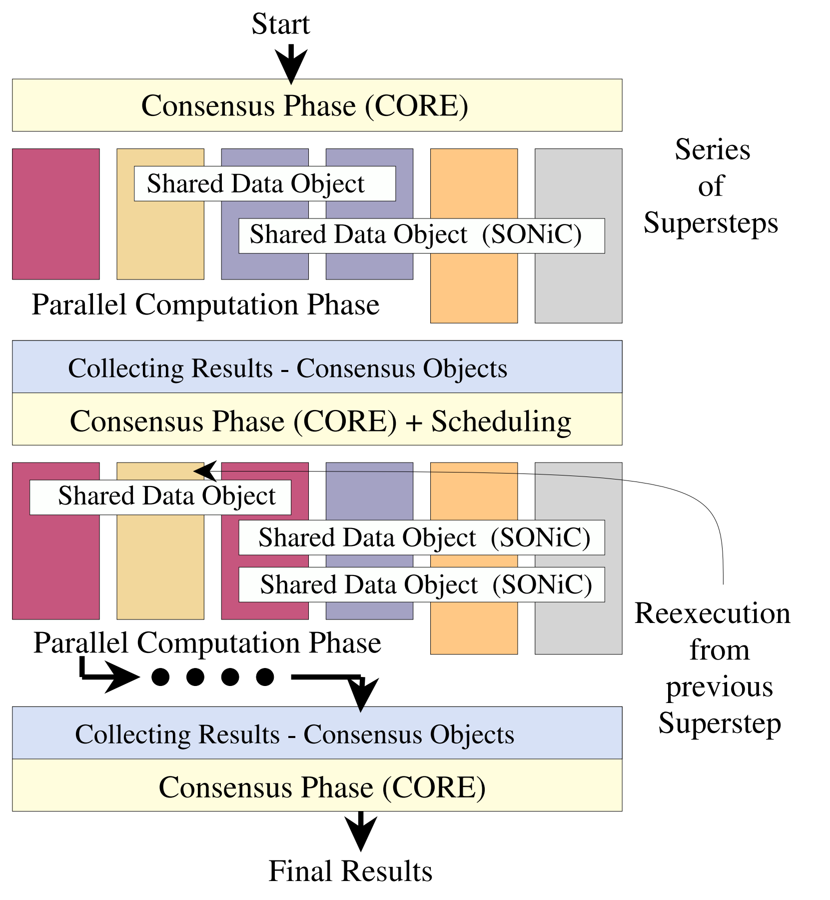

Fig. 3: *Execution framework for Responsive Services*

4. Implementation Issues

In this section, we focus on two issues. First, the establishment of a timing firewall between real-time and non-real-time (CORBA) parts of the composite objects, so that the non-real-time part cannot violate the real-time scheduling rules. Second, the implementation of data transfer between its real-time and non-real-time part in such a way that real-time accesses to the data cannot be blocked by non-real-time functions.

We assume that CORBA method invocations result in execution of non-real-time threads in a composite object. However, a CORBA method invocation may trigger execution of a "real-time service" method. Those "real-time service" methods are executed following the Consensus/Execution model discussed earlier and implement "Responsive Services".

We use the Scheduling Server technology [Polze97b] to establish a firewall between real-time and CORBA parts of a composite object. The scheduling server has been implemented and experimentally evaluated on the Mach operating system. Its principles are also applicable to COTS operating systems like Windows NT.

The Scheduling Server uses a time-slicing technique to divide CPU cycles between real-time and non-real-time tasks. In this approach "guarantee slots" are scheduled to run at fixed periods and provide a guaranteed amount of time and resources. Real-time scheduling methods can be applied to manage the tasks inside the guarantee slots. Therefore, we can view the series of guarantee slots as CPU of reduced availability.

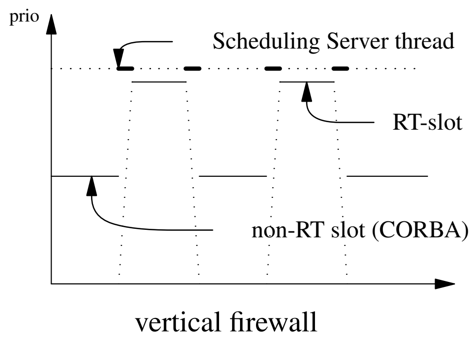

Fig. 4*: separation of RT- and non-RT-threads*

Data replication is the key to allowing independent 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, and a part which can be paged out and is treated like a standard non-real-time user’s process’ data. We employ the scheduling server to trigger periodic execution of a consistency protocol for a composite object’s replicated variables. Depending on the protocol’s period, either a strong or weak consistent view on an object’s data is provided from both, the real-time and the non-real-time threads.

We distinguish three types of instance variables inside a “Composite Object”:

• real-time variables:
  These variables are accessible only from the real-time threads. They are locked in memory using `mpin()`, `vm_wire()` or similar system calls.

• non-real-time variables:
  These variables are accessible only from a composite object’s non-real-time threads. They may be paged out of memory.

• shared RT/NRT variables:
  Those data structures are replicated and accessible by both, real-time and non-real-time threads of an composite object. They allow for predictable interactions between both parts of the composite object. Fig. 5 shows a closer view onto an composite object’s data.

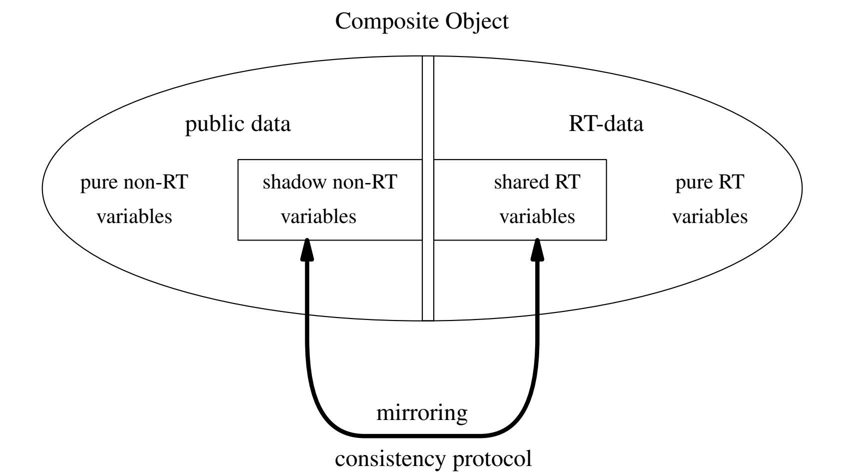

Fig. 5*: Data representation inside a composite object*

5. Related Work

Our ‘‘Composite Objects’’-approach provides a means for integration of the CORBA distributed computing platform with an object-based programming approach for responsive (fault-tolerant real-time) programming. We have outlined, how an object framework for responsive services can be constructed on top of the composite objects idea.

Here, we summarize related work with two different objectives: We discuss the idea of providing time guarantees 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. Other related work deals with techniques for structuring and specification of object-based systems which provide timing guarantees. Although a substantial amount of work exists in this area, none of the approaches focuses on interconnecting CORBA with object-based real-time programming.

Realtime CORBA

The Object Management Group (OMG) has founded a ‘‘Realtime CORBA’’ special interest group (SIG) [McGoogan96].

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 deals 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 identified as prerequisites for the proposed approach.

A new endsystem architecture for CORBA systems called TAO is discussed in [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, in contrast to the ‘‘Composite Objects’’ idea of interfacing an existing real-time system with CORBA, the TAO-approach focuses on completely new, CORBA-based real-time systems.

Object-based real-time systems

The idea of using object-oriented structuring methods for real-time programming has been the focus for several research activities within the last years.

[Kim95] examines an object-based approach for encapsulation of temporal characteristics of adaptable, real-time software. The work proposes the association of deadlines and criticalities with method invocations on real-time objects and compares the object-oriented design with the more traditional function-oriented approach.

The concept of active objects with associated timing constraints for real-time programming is discussed in [Attoui91]. The behavior of an active object is described by the runtime for methods/method blocks and by the periodicity of the object’s activities. A priority-based client/server model for the execution of distributed real-time programs is proposed.

[Shrivastava91] describe an object-oriented model for structuring distributed real-time applications. The active object model, extended with atomic method invocations (transactions) and exception handling is used to implement active replication of objects. This way, the availability of real-time objects is increased.

The application of object technology fo real-time operating systems has been studied in [Mercer90]. A Real-Time System (ARTS) kernel and its object model are presented. Classification schemes for objects and priority inheritance properties are presented. Furthermore, methods for the implementation of critical regions are described.

Besides description of an object’ systems runtime behavior, the RTO.k model [Kim95] focuses on design-time guarantees for timely service capabilities of encapsulated object-subsystems. Based on an active object approach, time-triggered (spontaneous) methods with design-time guarantees and service methods without those guarantees are distinguished. A basic concurrency constraint-rule restricts the execution of service methods in a way, that no time-triggered methods are disturbed. The Distributed Real-time Ever Available Microcomputing (DREAM) kernel implements an execution engine for RTO.k objects. A prototypical implementation is based on PCs and ethernet. However, in contrast to the ‘‘Composite Object’’ approach, the RTO.k model makes no attempt to interface with a standard distributed computing platform.

6. Conclusions

There is an urgent need to enhance today's distributed computing environments with a notion of quality-of-service as provided by the responsiveness measure. However, the question: ‘‘what should and should not be abstracted away?’’ is critical for those enhancements to today's CORBA. Based on the ‘‘Composite Objects’’ idea — our answer to the question above — we have described an object framework for integration of responsive computing and CORBA.

‘‘Composite Objects’’ allow for predictable interactions between real-time and non-real-time (CORBA) methods. We have presented a methodology for creating objects which interface to CORBA without violating real-time requirements. Also, we have presented our approach to timing firewalls and inter-firewall communication. We propose to use weakly consistent memory management schemes for data transfer between real-time and non-real-time part of a composite object.

We have described, how responsive CORBA services can be provided based on the composite object idea. Redundant execution within special composite objects (consensus objects) is used to implement fault tolerance. Consensus is employed at various stages during execution of a responsive service to achieve synchronization, reliable communication, and fault diagnosis.

References

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

[Attoui91] A.Attoui, M.Schneider; *An Object-Oriented Model for Parallel and Reactive Systems*; Proceedings. Twelfth Real-Time Systems Symposium, San Antonio, TX, USA, pp. 84-93, ISBN 0-8186-2450-7, 1991.

[Kim95] K.H.Kim; *Toward new-generation real-time object-oriented computing*; Proceedings of the Fifth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems, Cheju Island, South Korea, pp. 520-529, ISBN 0-8186-7125-4, 1995.

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

[Malek94] M. Malek; *A Consensus-Based Framework and Model for the Design of Responsive Computing Systems*; in Ultradependable Parallel and Distributed Computing, G. Koob and C. Lau (eds.), Kluwer, 1994.

[Malek95] M. Malek, A. Polze, and M. Werner; *A Framework for Responsive Parallel Computing in Network-based Systems*; in Proceedings of International Workshop on Advanced Parallel Processing Technologies, Bejing, China, pp. 335-343, September 1995.

[Mercer90] C.W.Mercer, H.Tokuda; *The ARTS real-time object model*; Proceedings. 11th Real-Time Systems Symposium, Lake Buena Vista, FL, USA, pp. 2-10, ISBN 0-8186-2112-5, 1990.

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

[Polze97a] A.Polze, M.Malek; *Network Computing with SONiC*; in Journal on System Architecture (the Euromicro Journal), Special Issue on Cluster Computing, Elsevier Science B.V., Netherlands, 1997.

[Polze97b] A.Polze, G.Fohler, M.Werner; *Predictable Network Computing*; Proceedings of International Conference on Distributed Computing Systems (ICDCS'97), Baltimore, May 1997.

[Polze97c] A.Polze, K.Wallnau, D.Plakosh; *A Study in the Use of CORBA in Real-Time Settings: Model Problems for the Manufacturing Domain*; to appear as technical report CMU/SEI-97-TR-011, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, USA, December 1997.

[Rajkumar95] R.Rajkumar, M.Gagliardi, and L.Sha; *The Real-Time Publisher/Subscriber Inter-Process Communication Model for Distributed Real-Time Systems: Design and Implementation*; Proceedings of The First IEEE Real-Time Technology and Applications Symposium, May 1995.

[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.

[Shrivastava91] S.K.Shrivastava, A.Waterworth; *Using objects and actions to provide fault tolerance in distributed, real-time applications*; Proceedings. Twelfth Real-Time Systems Symposium, San Antonio, TX, USA, pp. 276-285, ISBN 0-8186-2450-7, 1991.

[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.

[Valiant90] L.G.Valiant; *A Bridging Model for Parallel Computation*; in Communications of the ACM, 33(8):103-11, Aug. 1990

[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.