Composite Objects: Real-Time Programming with CORBA[1]

Andreas Polze[2] and Lui Sha

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

lrs@sei.cmu.edu
Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA, 15213, USA

Abstract

The Common Object Request Broker Architecture is a successful, standardized system integration framework based on distributed object technologies. An ongoing effort concerns extensions to CORBA to incorporate real-time computing needs.

Although the use of objects in real-time computing is straightforward, the technical challenge lies in the replacement of static real-time computing infrastructures with a flexible real-time computing infrastructure, in which distributed real-time client and server objects can be created and connected as needed during runtime.

We propose the concept of "Composite Objects" for the integration of real-time and non-real-time computing into a single object-based framework. Within the paper, we present a methodology for creating objects which interface to CORBA without violating real-time assumptions. We present a example scenario which integrates a real-time computing architecture and CORBA components via "Composite Objects". Finally, we discuss implementation-related issues based on our experiences with "Composite Objects".

Key words: Object-based Real-Time Computing, CORBA, Composite Objects, Non-interference, Interoperability.

1. Introduction, Motivation

The Common Object Request Broker Architecture (CORBA) [1][2] is a successful, standardized system integration framework based on distributed object technologies. An ongoing effort within the OMG concerns real-time CORBA (RT-CORBA) [3], as an extension to the OMG Object Management Architecture to support a wide spectrum of distributed, real-time, fault-tolerant systems.

The use of objects in real-time computing is straightforward. The natural model for many real-time computing applications has been the data flow model. For example, distributed sensors in a plant are first sampled. The sampled data streams are filtered and sent to various display servers for operators and to distributed plant controllers. Based on the sensor readings and operator inputs, the controller send commands to the actuators of the plant. Each of these activities: sampling, display and control can be easily written as objects. These objects can then be mapped to runtime scheduling units such as threads or processors on a real-time OS such as POSIX.1b, with priorities according to some real-time scheduling algorithm, e.g., GRMS [4]. When objects are distributed, the data flows between them can be supported by real-time communication network standards such as DeviceNet designed for noisy factory environments, running a real-time communication protocols on top of IP, or using the emerging standard POSIX.21 [5] over a (nearly) collision free network.

The technical challenge lies in the replacement of static real-time computing infrastructures with a flexible real-time computing infrastructure, in which distributed real-time client and server objects can be created and connected as needed during runtime. The Simplex architecture developed at the SEI has made considerable progress in the improvement of flexibility [6]. Without stopping the system operation, software or hardware modules encapsulated by the ‘‘replacement units’’ [7] of Simplex architecture can be added, deleted or replaced easily and reliably. What has been missing, however, is the system integration facilities provided by CORBA such as name service, interface management, dynamic invocations, etc. The authors have been cooperating on the integration of CORBA and Simplex architecture.

However, there are challenges. The key feature of CORBA is that users can create application systems by

[1] Appeared in Proc.of 24th Euromicro Conf., Vol.II, pp.: 997-1004, Vaesteras, Sweden, August 25-27, 1998.

[2] Work described here has been done in part while the author was visiting scientist at the Software Engineering Institute, CMU.

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 concern 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 they do 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, the correctness of real-time computing depends not only on the values output by computation but also the times at which outputs are generated. Examples of distributed real-time systems include refineries, manufacturing plants, tele-communication networks, command and control systems, power generation and distributions. Locations, hardware speeds, workloads, and the schedulers used by operating systems and applications can affect timing and therefore may affect the correctness of real-time computing. Indeed, real-time scheduling such as GRMS demands such information.

Thus, the real-time extension of CORBA is not a simple refinement of the existing CORBA model. Rather, it is at the heart of CORBA specifications: what should and should not be abstracted away and how to do it.

One way to address the real-time issues is represented by the excellent work done by the OMG Real-Time CORBA working group. The Request for Proposal for the fixed priority version has been announced recently. The success of RT CORBA will allow software engineers to directly address the real-time computation issues within RT CORBA. However, in an enterprise system, having a framework to properly integrate the real-time portion (RT CORBA or traditional real-time programming environment) with non-real-time environment is still an important issue. In this paper, we present the ''Composite Objects'' approach which can be used to systematically integrate real-time and non-real-time computation.

We advocate a ''co-existence and the-need-to-know'' approach to address this problem. There are three design rules.

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

Rule 1 is implemented by partitioning the system's computation and communication resources. Such partitions can be done physically or logically as discussed later in this paper. Rule 2 and 3 will be implemented by creating ''Composite Objects''.

The rest of the paper is organized as follows: In Section 2 we briefly discuss some related work. Section 3 describes the concept of ''Composite Objects''. In Section 4 we discuss the idea of timing firewalls and inter-firewall communication. An example application employing the concept of ''Composite Objects'' is shown in Section 5. Section 6 concludes the paper.

2. Related Work

We summarize related work with two different objectives: 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. Other related work deals with object-based techniques for structuring and specification of real-time systems.

Realtime CORBA

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.

ANSA (Advanced Networked Systems Architecture) [8] 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 [9][10]. 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.

TAO is a pioneering work on RT CORBA where fixed priority real-time scheduling is tightly integrated into the system [11][12]. 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.

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.

[13] 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 [14]. 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.

[15] 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 [16]. The 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 system's runtime behavior, the RTO.k model [13] 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 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.

3. The Concept of Composite Objects

Our concept of ''Composite Objects'' is an approach to integrate real-time and non-real-time computing into a single object-based framework. We present a methodology for creating objects which interface to CORBA without violating real-time assumptions. Composite objects may provide a unified interface for inter-acting between real-time and non-real-time objects. This interface is identical to the existing CORBA definitions from the view point of non-real-time objects. Non-real-time clients can invoke the service of real-time objects via the standard CORBA interface. 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. This way, non-real-time components like graphical user interfaces and databases can be connected to an existing real-time systems.

Real-time clients can obtain the service from standard CORBA facilities. In that case the composite object will act as the surrogate of the real-time object in getting the service and forwarding it to the real-time object, so that real-time clients will not wait for the un-predictable timing of non-real-time services. An important application of this service is to identify potential real-time servers registered with CORBA. Once real-time clients and servers are identified and their resources are reserved, execution of their methods and creation of data flow paths can be managed by sending commands to Simplex architecture servers.

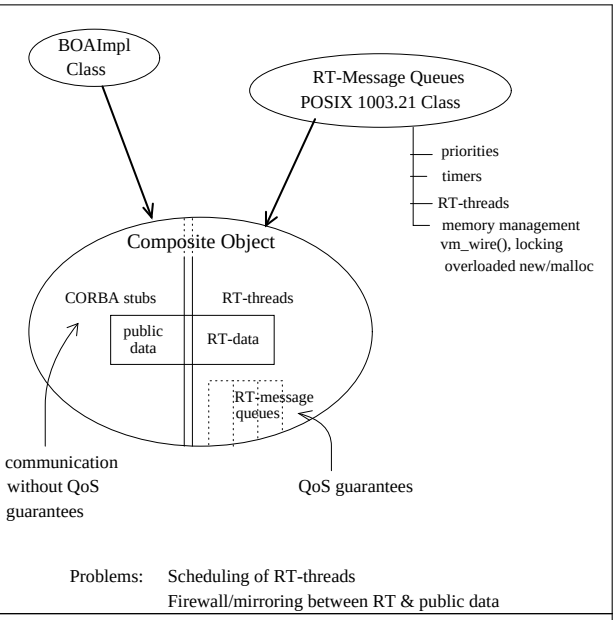

Fig. 1*: Structure of a composite object*

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 such as Simplex architecture 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 runtime guarantees can be given for execution of ‘‘time triggered methods’’ and ‘‘real-time service methods’’, respectively. 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. 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.

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 [17].

The invocation of these methods can be either accepted or rejected, subject to the decision of the schedulability analysis provided by Simplex replacement transaction servers. If all the requested client objects, server objects and their data flow paths are schedulable, they will be created and a new real-time service will be created at runtime. Otherwise, the request will be rejected.

Having introduced the basic concept of ‘‘Composite Objects’’ for integration of CORBA and the Simplex architecture, we move on to addressing the detailed design and application issues.

4. Timing Firewalls and Inter-Firewall Communication

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 that are needed by the real-time part. Second, the implementation of data transfer between it’s 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.

Our ‘‘Composite Objects’’ approach uses the idea of separation of concerns. A flexible real-time subsystem can be developed within the framework of Simplex architecture. This real-time subsystem is then encapsulated by the real-time part of the composite object: RTCO. RTCO can, of course, run on a processor dedicated to real-time computing. It can also run on a logical real-time virtual computer. There are many ways to partition a computer into a real-time virtual computer and a non-real-time virtual computer.

The easiest way is to use a real-time operating systems and to divide the priorities into two classes. The higher priorities for the real-time processing and the low priorities for the non-real-time processing. As long as the real-time computing’s utilization of the processor is suitably bounded, the non-real-time virtual computer, with a guarantee percentage of the CPU, can provide acceptable performance.

The performance of non-real-time service can be further improved using the sporadic server [4]. In this approach, each sporadic server provides an aperiodic service of up to k msec in a period of n msec. The sporadic servers are treated as if it is a real-time periodic task. The

recently developed two level scheduling approach is an advanced method to support multiple real-time and non-real-time computing paradigms on a single processor [18].

The Scheduling Server [19] 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.

The techniques mentioned above allow us to implement the idea of NON-INTERFERENCE within the composite object framework. The users of Composite Objects are free to use any simple or sophisticated scheduling method to create virtual real-time and non-real-time computers.

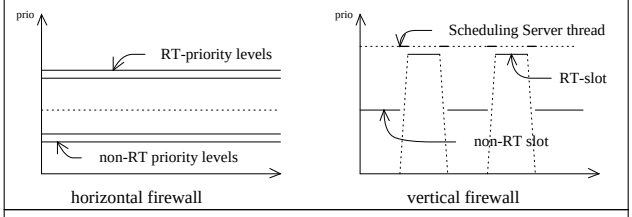

Fig. 2*: Firewalls: separation of RT- and non-RT-threads*

Data replication is the key to allow 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 discuss protocols to provide either strong or weak consistent view on an object’s data from both, the real-time and the non-real-time threads. A ‘‘Composite Object’’ is therefore rather a design concept than a contiguous data structure at runtime of a program.

We distinguish three different kinds 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. A composite object’s base class provides overloaded versions of operator `new()` and `malloc()` to create real-time variables.

• 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 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.

There are two approaches to handle shared RT/NRT variables. First, if strong consistency is needed in data sharing, shared RT/NRT variables can be treated as real-time variables using real-time synchronization methods for access control, for example, the priority inheritance protocols supported by POSIX.1b and most commercial RT OS [4]. Under this approach, a multi-threaded object is written to encapsulate the shared data. Each method represents a critical section that operates on the shared RT/NRT data. The worst case duration of priority inversion caused by non-real-time access is bounded by the duration of critical sections and can be easily computed and its effect can be factored into schedulability analysis.

If the shared RT/NRT data are sampled data, older versions, up to some maximal permissible delay, are often still usable for non-hard real-time applications. For example, the very latest quotes of stock prices are essential for real-time program trading but 1 day old quotes on newspaper are good for most long term investors. Multi-version data with weak consistency allows for a higher degree of flexibility when transferring distributed real-time data to non-real-time users.

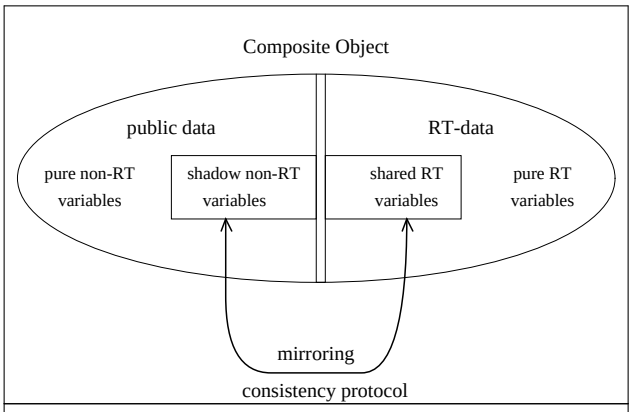

Fig. 3*: Data representation inside a composite object*

Composite Objects support the use of shadowed real-time variables for non-real-time applications using weak consistency. These shadowed RT variables reside in pageable memory and are copies of RT/NRT shared variables. The rate of updates is programmable. The data transfer mechanisms inside the composite object allow for INTER-OPERABILITY between real-time and non-real-time parts of the object. Fig. 3 shows a closer view onto an composite object’s data.

5. Unstoppable Robots - an Example Application

The ‘‘Unstoppable Robots’’ [20] is a distributed demo application based on our framework for High-Performance Responsive Computing [21]. The application demonstrates soft real-time and fault tolerance aspects. Our initial implementation of the ‘‘Unstoppable Robots’’ is based on the Mach operating system and the Scheduling Server [19][22] concept. Here, we describe an extended version of the ‘‘Unstoppable Robots’’ application which employs ‘‘Composite Objects’’ to interconnect CORBA with a legacy soft real-time application and illustrates the principles of NON-INTERFERENCE, INTER-OPERABILITY, and ADAPTIVE ABSTRACTION.

The original application’s idea looks like follows: A number of simulated robots (3..6) are moving on a simulated plate (the world). The plate has to be kept in instable balance. Robots are using up energy and, therefore, have to move to a designated fuel station regularly. The fuel station is located outside of the gravity center. Thus, if one robot feels that it has to move towards the fuel station, it has to convince other robots to move in the opposite direction to keep the plate in balance. Agreement among robots is achieved by running a consensus algorithm (Actually, since robots are attached to controllers, the consensus is achieved among the robots’ controllers). Various levels of component replication allow to demonstrate fault-tolerant behavior of the robots.

The initial version ot the application is a pure simulation. Robots are simulated on the screen. Changes in robots’ colors indicate varying fuel levels. The simulation executes the consensus algorithm with a frequency of 4 Hz. Within each consensus round, a decision about each robot’s next step is being made. The coordination between their actions is soft real time where weak consistency of data can be used. To guard against occasional violation of the weak temporal constraints, local timing fault tolerance is taken. The initial version of ‘‘Unstoppable Robots’’ is based on network-transparent Mach interprocess communication.

Now, we want to demonstrate how ‘‘Composite Objects’’ can be employed to interconnect a Java-based display (a second world) and a X Window-based interactive controller (‘‘virtual joystick’’) to the ‘‘Unstoppable Robots’’ simulation via CORBA. The Java-based display represents non-real-time computation that must be interfaced with the soft real-time application.

Furthermore, we did acquire a ‘‘real’’ tin can-sized robot (khepera-robot) [23] which we wanted to connect to the simulation. The real robot is controlled by a computer running rtLinux [24]. The control of the robot represents hard real-time computations. Again, we want to exploit the ‘‘Composite Object’’ concept and use CORBA to interconnect the khepera robot to the fault-tolerant, soft real-time simulation.

Usage of ‘‘Composite Objects’’ is critical since we have to ensure that CORBA’s unpredictable, varying communication latency does not block/disturb neither the simulation nor the rtLinux task driving the actual khepera robot. The khepera robot receives its commands via the computer’s serial interface. The corresponding rtLinux driver task runs with a frequency of 800 Hz and uses `outb` assembly instructions to control the robot.

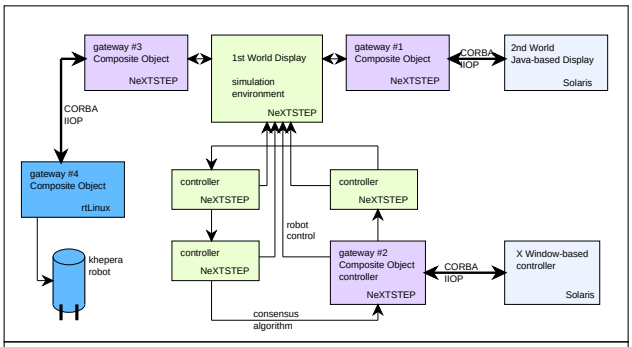

Fig. 4*: Communication structure of the Unstoppable Robots demo*

Components and communication structure of the extended ‘‘Unstoppable Robots’’ are depicted in Fig. 4 for the case of 4 controllers interacting with the simulation environment (world).

Fig. 4 shows 4 gateways - implemented as ‘‘Composite Objects’’ - which interface CORBA with components of the demo application. Let us study in some detail the requirements for the different ‘‘Composite Objects’’.

Gateway #1 interconnects the Java-based 2nd world (display) with the simulation environment. The Java-based display periodically invokes a method of the gateway object to obtain status information from the simulation (1st world). Since this gateway object is not part of the simulation’s control loop, a best effort approach can be used to forward CORBA requests from the 2nd world to the simulation. However, the gateway has to implement some means of call admission. Otherwise, the Java-based 2nd world might overflow the simulation environment with requests, putting an in-acceptable high load onto the system (Actually, this could be achieved by just running enough copies of the 2nd world). We demonstrate the ‘‘Composite Object’s’’ principle of INTER-OPERABILITY by using gateway #1 for implementation of call admission and caching of status information.

Gateway #2 is located inside the simulation's control loop. Therefore, it has to react in a timely manner. High CORBA communication latency must not disturb the timing behavior of the simulation. Our solution here is to use multiversion programming: Although the complex, consensus-based control algorithm is executed in the remote, X Window-based controller, the gateway itself implements a ''fallback'' algorithm. It decides to stop the robot if a routing decision from the CORBA side comes late. This may result in a slight, temporary inbalance of the simulated plate - however, the remainder of the system is not affected by CORBA's misbehavior. Furthermore, we assume that CORBA communication latency is well below the simulation's cycle time in most cases.

Gateway #2 demonstrates the ''Composite Object's'' principle of ADAPTIVE ABSTRACTION - as the CORBA programmer - who has no chance of expressing timing behavior - just uses the standard CORBA interface to implement his control algorithm. In contrast, the programmer of the gateway object knows about timing constraints and employs multi-version programming to deal with the requirements.

Gateway #3 is similar to gateway #1 with respect to timing criticality - it is located outside of the demo application's control loop. However, in contrast to the Java-based 2nd world, which is polling information from the demo application, gateway #3 is pushing data to the rtLinux driver for the khepera-robot. Gateways #1-#3 are implemented using the XEROX PARC Inter-Language Unification (ILU 2.0 alpha11) CORBA implementation on the NeXTSTEP 3.3 operating system.

Gateway #4 is a ''Composite Objects'' implemented on the rtLinux system using OOC's OmniBroker CORBA implementation. It forwards CORBA position information to the real-time control task. The ''Composite Object's'' principle of NON-INTERFERENCE between both parts of gateway #4 is achieved by employing the `rtfifo`-communication mechanisms available under rtLinux. This allows us to generate a trajectory and send control commands to the khepera robot with a frequency of 800 Hz - and to move the actual robot in sync with one of the simulated counterparts.

The whole scenario demonstrates a remakable degree of complexity. We are using 3 different CORBA implementations (ILU on NeXTSTEP, OmniBroker on rtLinux and Solaris, Orbix on Solaris). Components are written in 4 different programming languages (Objective-C, C, C++, Java). And - although Java is basically platform independent - we have demonstrated interactions between 3 different operating systems (NeXTSTEP, Solaris, rtLinux).

The extended ''Unstoppable Robots'' demo application shows the advantages of using CORBA technology in heterogeneous environments and simultaneously demonstrates how the ''Composite Objects'' approach can be used to integrate time-critical and time-aware components with standard CORBA. Also, we have shown how a ''legacy'' real-time application can be extended with new components using ''Composite Objects'' and CORBA.

6. Conclusions

The question: ''what should and should not be abstracted away?'' is critical for the connection of real-time programming and CORBA. Based on the ''co-existence and the-need-to-know'' approach we have developed the ''Composite Objects'' approach for integration of real-time programming and CORBA.

''Composite Objects'' allow for predictable integration of real-time and non-real-time computing into a single object-based framework. We have presented a methodology for creating objects which interface to CORBA without violating real-time assumptions. Therefore, we support the construction of layered distributed real-time systems: composite objects support the construction of hard real-time subsystems with design time guarantees. The emerging real-time CORBA can be used to interconnect those subsystems, forming an upper soft real-time layer.

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. With the extended ''Unstoppable Robots'' demo we have described, how the composite object idea has been employed in a distributed real-time robotics application. Finally, we have discussed issues of prototypical implementations of the ''Composite Objects'' concept.

Acknowledgements

The original ''Unstoppable Robots'' simulation was written by Matthias Werner. We want to thank our student Jan Richling for his implementation work in context of the extended simulation and the driver for the khepera robot.

German Research Foundation (DFG) has supported part of this work through funding for a research stay of the first author at SEI, CMU, under contract Po552/4-1.

References

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

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

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

[4] 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.

[5] B.Pollak (ed.); *Portable Operating System Interface (POSIX) - Part 1x: Realtime Distributed Systems Communication Application Program Interface (API)*; IEEE Standard, P1003.21 LIS/V1.0, September 1996.

[6] L.Sha, R.Rajkumer, M.Gagliardi; *Evolving dependable real-time systems*; Proceedings of 1996 IEEE Aerospace Applications Conference, Vol.1, pp.335-46, Aspen, CO, USA, February 1996.

[7] M.Gagliardi, R.Rajkumar, L.Sha; *Designing for Evolvability: Building blocks for Evolvable Real-Time Systems*; Real-Time Applications Symposium, 1996.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] Z.Deng, L.Zhang, J.W.-S.Liu; *An Open Environment for Real-Time Applications*; submitted to Real-Time Systems Journal, 1997.

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

[20] M.Werner, M.Malek; *The Unstoppables - Responsiveness by Consensus*; HUB Informatik-Berichte No.: 90/97, 19 pages, ISSN 0863-095 90, Berlin, December 1996.

[21] M.Malek, A.Polze, 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.

[22] 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.

[23] K-Team SA; *Khepera User Manual, Version 4.06*; K-Team SA, Ch.du Vuasset, CP 111, 1028 Preverenges, Lausanne, Switzerland, Novemver 1995; see also: http://diwww.epfl.ch/Khepera/.

[24] 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.