Towards Predictable CORBA-based Web-Services

Andreas Polze, Jan Richling, Janek Schwarz, and Miroslaw Malek

Department of Computer Science
Humboldt University of Berlin
Rudower Chaussee 5, 12489 Berlin, Germany
{apolze,richling,schwarz,malek}@informatik.hu-berlin.de

Abstract

Over the past several years, the World Wide Web has emerged from a research project to an environment for open, commercial services, such as online-banking, travel reservation, and stock-trading. However, in contrast to the best-effort approach provided by the Web, many of those services demand higher predictability and quality-of-service properties such as security, end-to-end availability, dependability, and real time. The development of a systematic, architectural approach for connecting Web-interfaces to legacy systems, such as databases and transaction processing is an open research question.

The Common Object Request Broker Architecture (CORBA) is a widely-accepted, standardized open system integration framework based on distributed object technologies, which has been successfully used for implementation of open Web services. CORBA is focused on facilitating general computing environments and does not explicitly address quality-of-service parameters neither for its communication links nor its endsystems. However efforts like the Real-Time CORBA Special Interest Group (SIG) at OMG and the "pluggable protocols" proposal will ultimately lead to support of quality-of-service properties for CORBA communication links.

Within this paper we are concentrating on architectural approaches for fault-tolerant, highly available endsystems. We present the "Observer" approach for implementation of reliable CORBA clients. Consensus protocols based on the "Composite Objects" technique is our solution for constructing CORBA servers with high predictability regarding timely and reliable method execution. Our middleware uses commercial off-the-shelf (COTS) technology and aims at conversion of legacy applications into reliable Web-services.

We present Java-based Web-interfaces to the "Balancing Robots" soft real-time simulation. Also, we demonstrate a fault-tolerant version of the Netscape Navigator based on our "Observer" technique.

Key Words: Fault Tolerance, CORBA, Composite Objects, Observer, Scheduling Server, Web Services.

1. Introduction, Motivation

The World Wide Web has emerged from a research project to an environment for open, commercial services, such as online-banking, travel reservation, and stock-trading. However, in contrast to the best-effort approach provided by the Web, many of those services demand higher predictability and quality-of-service properties. Most of the highly-available, fault-tolerant, real-time systems have been implemented in embedded settings and there is an urgent need to open up this type of computing technology to a larger number of users utilizing heterogeneous, distributed computing environments [25]. The application of fault-tolerant, real-time computing techniques to endsystems for reliable, CORBA-based Web-services is the focus of this paper.

The Common Object Request Broker Architecture (CORBA) [15][26] is a widely-accepted, standardized system integration framework based on distributed object technologies. Through its Java language bindings, CORBA middleware offers a promising basis for construction of complex Web-services. In this context, Java applets can be used to access, monitor, and control responsive (fault-tolerant, real-time) applications via complex, stateful interactions. Within this paper we discuss how techniques like call admission, caching, and fallback procedures can be used to achieve predictable behavior of the responsive applications despite varying numbers of clients and interactions.

CORBA and the Object Management Architecture (OMA) specify three components as basic building blocks for distributed applications: the Object Request Broker (ORB), client interfaces (stubs), and server interfaces (skeletons). Communication protocols have been specified for interactions among ORBs (General Inter-Orb

appeared in Proceedings of 2st IEEE International Symposium on Object-oriented Real-time distributed Computing, pp.: 182-191, ISBN 0-7695-0207-5, St. Malo, France, May 1999.

Protocol: GIOP, Internet Inter-Orb Protocol: IIOP). The "pluggable protocols" proposal [14] to OMG and work in context of RT-CORBA deal with quality-of-service properties for CORBA's communication protocols.

Besides predictable communication protocols, techniques for reliable and timely execution in endsystems (client, server) are required to make predictable Web-services (see Fig. 1) feasible. Within this paper we are concentrating on architectural approaches for fault-tolerant, highly available endsystems. We present the "Observer" approach for implementation of reliable Web-clients[1]. The "Composite Objects" technique [18] is our solution for constructing CORBA servers with high predictability regarding timely and reliable method execution.

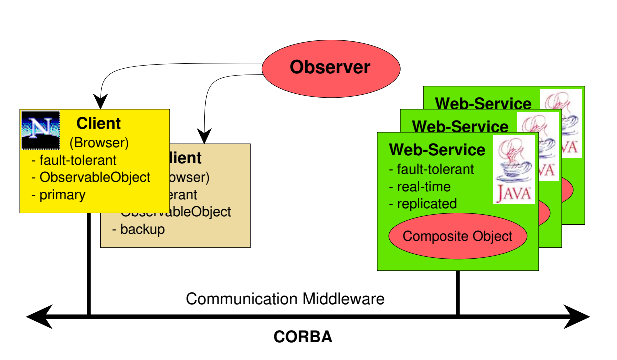

Fig. 1*: Components for Predictable Web-Services*

"Composite Objects" act as filtering bridge between standard middleware platforms and software frameworks providing services with certain quality-of-service (QoS) guarantees. Key concepts in our approach are noninterference, interoperability, and adaptive abstraction. Here, we discuss design and implementation of "Composite Objects" following these principles in the context of CORBA middleware and demonstrate our technique by constructing Java-based Web-interfaces to the "Balancing Robots" soft real-time simulation — our proof-of-concept vehicle. However, the concepts inherent to "Composite Objects" and presented throughout the paper are more general and applicable to other middleware platforms like DCE, or DCOM and other QoS-requirements as well. Fig. 2 shows "Composite Objects" mediating between middleware platforms and quality-of-service properties like fault tolerance, real time, and security.

Through definition of the "ObservableObject" interface our "Observer" provides generic means for implementation of fault-tolerant client applications following the primary/backup technique. We demonstrate a fault-tolerant version of the Netscape Navigator based on CORBA and our "Observer" technique. However, the underlying concepts are again applicable to other middleware platforms like DCE, or DCOM.

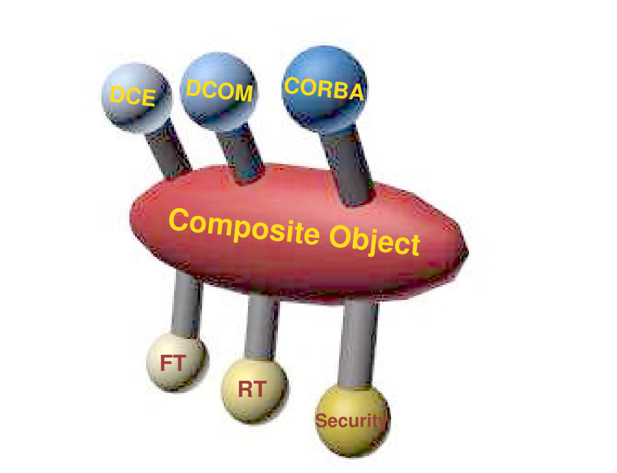

Fig. 2*: Bridging middleware and QoS-sensitive services with Composite Objects*

The remainder of the paper is organized as follows: Section 2 gives an overview over related work. While Section 3 presents our concept of reliable Web-services, Section 4 focuses on fault-tolerant clients. Implementation issues are discussed in Section 5, whereas Section 6 presents an example application. Section 7 concludes the paper.

2. Related Work

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

The "ObservableObject"-interface allows for implementation of robust CORBA clients following the primary/backup approach to fault tolerance. Related work describes two other approaches towards a fault-tolerant CORBA: a) Special ORB implementations may take

[1] Within context of this paper a Web-client is an application whose main purpose is access of Web-services via CORBA. In order to use the "Observer"-approach to fault tolerance, the Web-client (also CORBA client) has to support the "ObservableObject" interface. Therefore, in strict CORBA sense the application also acts as server.

advantage of underlying fault-tolerant hardware, and b) application of special CORBA-services for fault tolerance.

2.1. Realtime CORBA

The Object Management Group (OMG) has founded a "Realtime CORBA" special interest group (SIG) in 1996 and since then OMG has been soliciting technology for a Realtime Object Request Broker (ORB) comprising: fixed priority scheduling, control over ORB resources for end-to-end predictability, and flexible communications [9]. The Request for Proposal (RFP) for the fixed priority version of Realtime CORBA 1.0 has been announced in September 1997 [14]. A comparison of submissions responding to the RFP can be found in [3].

Work at the University of Rhode Island and the MITRE Corporations deal with syntactical extensions to CORBA IDL to express timing constraints [28][30]. "Timed distributed method invocations" are identified as one necessary feature in a real-time distributed computing environment as well as a "Global Time Service", "Real-Time Scheduling of Services", a "Global Priority Service", and "Bounded Message Latency". The "Affected Set Priority Ceiling Protocol" as a combination of semantic locking and priority ceiling techniques has been proposed for concurrency control in real-time object-oriented systems [27].

TAO is an innovative work on RT CORBA where fixed priority real-time scheduling is tightly integrated into the system [23][22]. Main goal of this work is to provide end-to-end quality-of-service for CORBA-based applications. A list of requirements for Object Request Broker implementations is presented, among them are resource reservation protocols, optimized real-time communication protocols and a real-time object adapter. However, TAO focuses rather on completely new, CORBA-based real-time systems, than on interfacing an existing real-time systems with CORBA.

2.2. Fault-tolerant CORBA

The idea of providing fault tolerance as additional feature to CORBA implementations has been the focus for several research activities within the last years. With the request for proposal for a *Fault tolerant CORBA Using Entity Redundancy* [13], issued in April 1998, the OMG is seeking to incorporate existing approaches for software fault tolerance into future versions of CORBA.

Somersault [11] is a platform for developing distributed fault-tolerant software components based on process replication. Somersault provides a fault-tolerant communication transport protocol, which can be plugged into an Object Request Broker, a combination which achieves replication transparency. The experimental ORBLite [10] developed at HP Labs. has been extended to use Somersault as fault-tolerant ORB transport.

Electra [7] is a CORBA ORB-implementation for reliable, distributed services. Electra extends the CORBA specification and provides group communication mechanisms, reliable multicasts, and object replication. The Electra-ORB uses services from the underlying ISIS [2] and HORUS [20] systems.

ORBIX+ISIS is an extension of the commercial ORBIX [6] ORB-implementation which introduces concepts like object groups and group communication based on the ISIS [2] toolkit. Again, the CORBA standard has been extended to allow for introduction of fault-tolerance measures. The programmer has to use special coding techniques to be able to use the fault-tolerance features of ORBIX+ISIS.

Phoinix [4] allows for implementation of server objects following the primary/backup approach for fault tolerance. In contrast to changing the ORB, Phoinix defines a new service as part of the Object Management Architecture (OMA). Using this service, a primary's object state can be periodically transferred into the corresponding backup object. Clients communicate solely with the primary object and have to detect failures themselves. Extended skeletons, stubs and libraries are provided to make those fault-tolerance services accessible.

All the approaches mentioned above are based on changes/additions to select CORBA-ORB implementations, which is acceptable for many (static) distributed environments. These proprietary solutions are currently the only way to achieve end-to-end reliability and have to be taken into account by the OMG standardization process. In contrast, our "Observer/Observable Objects"-technique and the concept of "Reliable Services" do not rely on one particular CORBA implementation, which makes them applicable to open environments like internet-based applications (Web-services), where heterogeneous scenarios are common and different vendors' ORBs have to interact.

3. Reliable Services: Composite Objects as filtering bridges

In many cases it is desirable to give predictions about behavior of remote services invoked via CORBA. Unfortunately, the specification of parameters for service execution like fault model, acceptable processor utilization, and timing behavior is completely beyond the scope of CORBA. We propose an extension of our previously developed CORE/SONiC [8] [17] framework for responsive computing by CORBA interfaces using "Composite Objects" as filtering bridges.

The CORE/SONiC (COnsensus for REsponsiveness/ Shared Objects Net-interconnected Computer) 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. COnsensus for REsponsiveness (CORE) provides protocols, services, and scheduling strategies at the microkernel level for real-time parallel computing even in presence of faults. Consensus protocols are used for fault detection and to establish a system-wide global view at certain points of program execution which serves as basis for decisions about re-execution and coarse-grained scheduling.

CORE provides a reliable communication subsystem to SONiC, using the services of the underlying Mach microkernel operating system. The CORE/SONiC "Scheduling Server" [19][21] provides predictable access to a workstation's resources and can be used to implement real-time scheduling policies like "Rate Monotonic Scheduling" or "Earliest Deadline First" on a standard operating system.

"Composite Objects" provide a method for creating objects which interface to CORBA and are simultaneously capable of real-time method execution. We use "Composite Objects" to rebuild the concept of CORE/SONiC in a CORBA-based environment. A group of composite objects distributed over the nodes of a workstation cluster forms the execution environment for reliable Web-services. Fault-detection among the composite objects is achieved by running periodic consensus protocols. This way open CORBA-based interfaces can be added to legacy applications based on the CORE/SONiC execution model, creating a reliable Web-service. CORBA requests are the basic units of replication and scheduling in our environment.

However, incoming CORBA calls may violate scheduling assumptions of the current execution. Implementation of call admission and caching schemes in a gateway object outside the real-time application can solve this problem. Depending on availability of resources, the execution environment may accept, reject or delay incoming CORBA calls. However, once a request has accepted it is processed — possibly replicated, depending on fault assumptions — in a predictable fashion. Fig. 3 illustrates our execution framework and highlights interactions among its components.

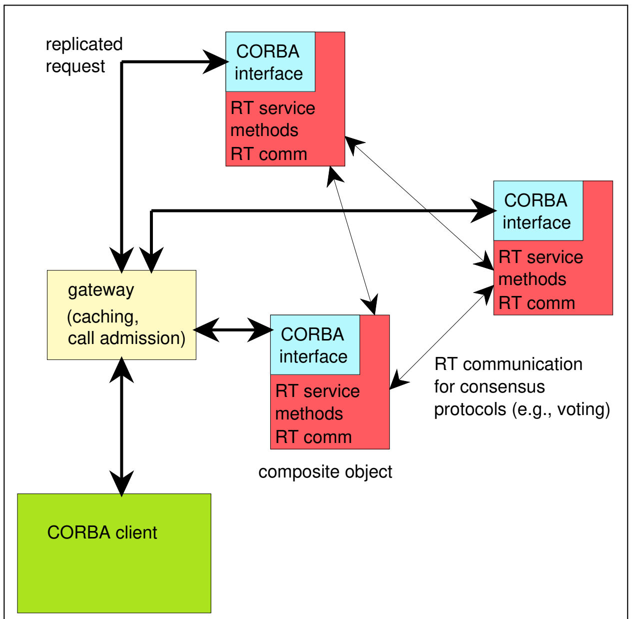

Fig. 3: *Execution framework for reliable Web-services*

"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. In our approach, we use composite objects to implement a filtering bridge between CORBA clients and a reliable service.

"Composite Objects" are based on three design rules:

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

The implementation of "Composite Objects" is discussed in some detail in Section 5.

4. Observable Objects: a technique for robust CORBA clients

Web-services such as online-banking, travel reservations, or brokering are demanding applications, which typically involve far more complex interactions than just serving static html-pages. Varying numbers of requests may create unpredictably high, bursty loads. Special measures like active replication and checkpointing have to be taken for fault-tolerance and performance. In contrast, the question of reliability for web browsers which are typically used to access these services receives much less attention. However, due to their fast evolution, web browsers are often unstable and exhibit problems like memory leaks which eventually may crash the browser. Browser crashes are disturbing for the user — they are prohibitive if predictability and end-to-end availability is the goal.

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

The ‘‘ObservableObject/Observer’’ architecture is based on CORBA. Non-CORBA, legacy applications may take advantage of our framework by using a wrapper approach. In order to encapsulate a non-CORBA application in a wrapper, the application has to offer a communication or programming interface, e.g. UNIX signals, stdin/stdout, IPC or different X11 mechanisms (e.g. properties), accessible to the wrapper.

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

In order to take advantage of our ‘‘Observer/ObservableObject’’ approach, a CORBA object has to implement the ‘‘ObservableObject’’ interface shown in Fig. 4.

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

Fig. 4*: Observable Object Interface - CORBA IDL*

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

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

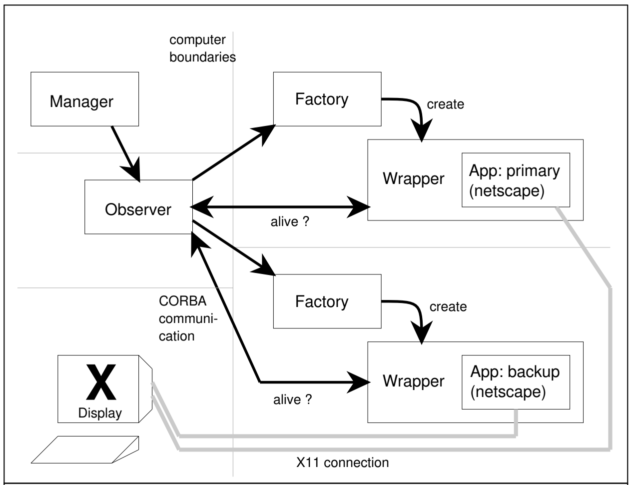

Fig. 5*: Primary/backup monitored by Observer*

We have used the ‘‘ObservableObject/Observer’’ architecture to increase the reliability of the Netscape Navigator web browser. In addition to a graphical user interface most browsers offer a programming interface for controlling the browser by external programs [12]. This functionality is used by a wrapper, which implements the

"ObservableObject" interface. A web browser is then monitored by the "Observer" by checking the availability of its wrapper.

In our simple test scenario, the state of the browser (current URL) is determined by the wrapper using the browser's programming interface. However, a more complex implementation of the "ObservableObject" interface, which is subject to future work, might use a proxy-approach and monitor the browser's TCP/IP connections to obtain a complete reflection of the browser's internal state (frames, cookies, html form fields, etc.). The contribution of our approach here is the genericity of the "ObservableObject" interface.

In case of a browser's crash, the wrapper terminates itself. From the "Observer's" viewpoint a browser has failed if its wrapper is no longer available. We assume that the wrapper will not fail, while the browser it encapsuletes continues to run. This assumption is sound, because a wrapper is a small program which can be tested thoroughly.

Fig. 5 depicts the communication structure in a scenario where two "ObservableObjects" acting as wrappers for Netscape Navigator processes are monitored by our system. Both, primary and backup are connected to the same X11 display. However, the backup's window is unmapped and becomes visible only in case of a crash fault of the primary.

The "ObservableObject/Observer" concept is not limited to CORBA. It can be adopted for other middleware platforms like DCE or DCOM.

5. Implementation Issues

Here, we discuss some implementation issues related to "Composite Objects". Also, we measure the overhead introduced by the concept.

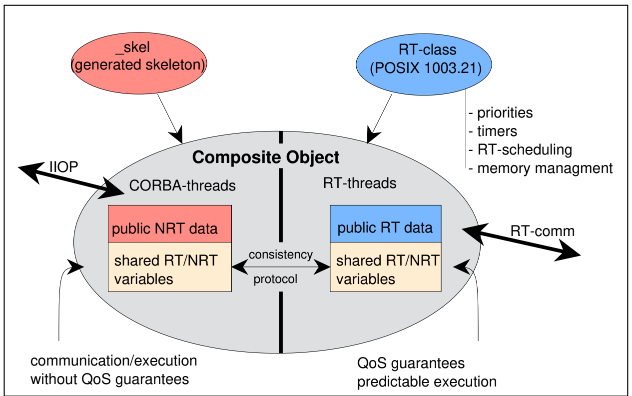

Fig. 6: *Structure of a Composite Object*

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 (`_skel` in many implementations). On the other hand, "Composite Objects" have the capability to create real-time programming abstractions like prioritized threads and real-time communication channels. As depicted in Fig. 6, they can be seen as descendants of a class which implements real-time servers.

"Composite Objects" consist of a real-time and a non-real-time part. Design time and runtime guarantees can be given for execution of real-time methods. In contrast, methods in the non-real-time part are executed following a best effort approach. "Composite Objects" establish timing firewalls [19] between real-time and non-real-time (CORBA) computing, so that the non-real-time part cannot violate the real-time scheduling rules [24] that are needed by the real-time part.

There are two different approaches to implementation of firewalls: General purpose operating systems may use a concept like the "Scheduling Server" to implement "vertical firewalls", whereas the different priority levels on a real-time operating system do implement "horizontal firewalls".

Our "Scheduling Server" [21] which has been developed as part of the "CORE/SONiC" project implements a time-slicing technique for predictable sharing of CPU cycles between real-time and non-real-time tasks. The "Scheduling Server" has been implemented as user-space server (no modifications to the OS' kernel) on the Mach and rtLinux operating systems. Similar work does exist for the Solaris operating system [5].

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.

The consistency protocol for shared RT/NRT variables as well as resource allocation (CPU cycles, memory) and call admission for CORBA clients are crucial for implementing the concept of "Composite Objects". Based on the Mach (NeXTSTEP 3.3) and rtLinux [1] operating systems we have used pipes, shared buffers and message queues (Mach IPC) for our implementation of "Composite Objects" as shown in Fig. 7.

Fig. 7 describes the data flow during `read/write` accesses to a "Composite Object's" variables. `read` requests in both parts of the "Composite Object" are always satisfied by accessing the local copy of a replicated, shared RT/NRT variable. In contrast, `write` requests result not only in updating the local copy, but the value is also stored in an interprocess communication (IPC) data structure (i.e., pipes, rt-fifos). Handler threads copy periodically values out of the IPC data structures into the local replica of the shared RT/NRT variable. The update rate for those handler threads is programmable.

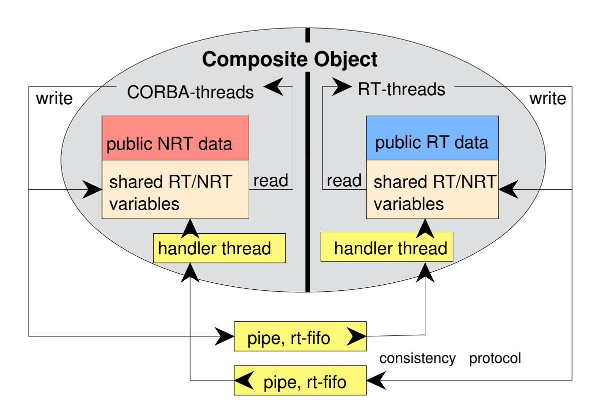

Fig. 7: *Consistency Protocol and Communication inside a Composite Object*

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 have implemented consistency protocols following both schemes.

To evaluate the overhead introduced by "Composite Objects", we have investigated a scenario where a real-time data source is accessed by a CORBA client through a "Composite Object" which implements an event-triggered consistency protocol.

Fig. 8 shows communication delays for the described scenario. The lower, dotted curve represents the time needed for real-time communication between "Composite Object" and data source (Mach Interprocess Communication). The middle curve describes the time needed for the consistency protocol inside the "Composite Object" plus real-time communication. The distance between lower and middle curve represents the overhead imposed by our implementation of a "Composite Object". It is roughly $1ms$ (10 % of the overall communication latency). One should notice, that both curves are pretty stable, indicating that introduction of a "Composite Object" as a mediator will not disturb an application's predictable timing behavior.

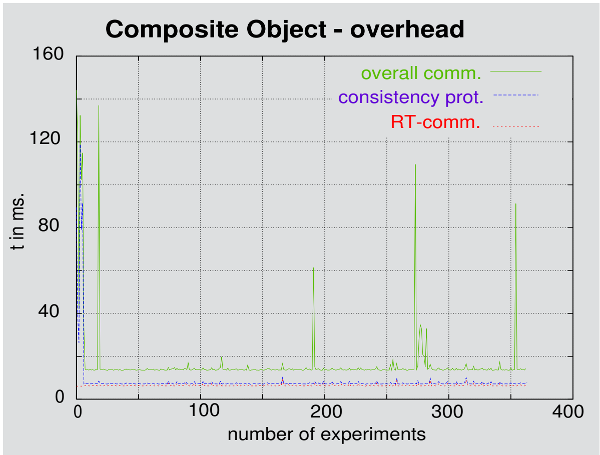

Fig. 8: *Overheads imposed by Composite Object*

In contrast, the curve at the top of the diagram, representing 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.

6. Example: Java interface to a soft real-time legacy application

As proof-of-concept vehicle we have created a web-service by extending a CORE/SONiC legacy soft real-time application with CORBA interfaces which are accessed by two Java applets.

The "Balancing Robots" [29] simulate the following scenario: Robots are moving on top of a plate which is kept in unstable balance. Robots are constantly using energy and therefore occasionally have to visit a fuel station which is located outside of the plate's center. Consensus among robots is achieved prior to every move with robots high on fuel acting as counter-balance to those moving towards the fuel station. Collision avoidance is implemented by consensus, too.

The application demonstrates toleration of robots' and controllers' crashes, omission, and computation faults. Control is subject to soft real-time contraints — calculations of controllers and robots' moves are executed with a frequency of 4Hz.

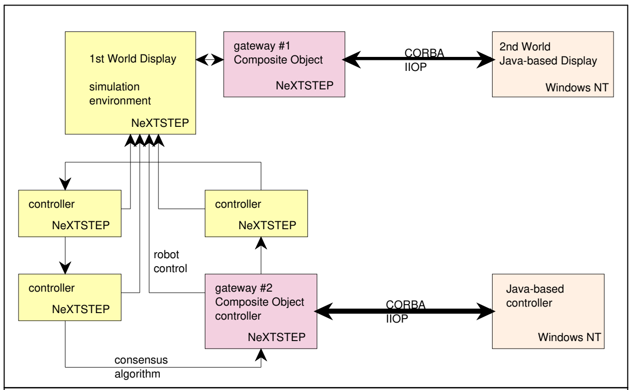

Fig. 9: *Communication structure of the Balancing Robots*

Using the "Composite Objects" technique we have extended the application by two CORBA-interface. Based on the Java language binding for CORBA, one applet has been implemented to monitor the soft real-time simulation and periodically obtain status information. Another applet implements the robots' control algorithm in Java and interacts with the real-time simulation's native controllers during consensus rounds. These interactions have fixed deadlines (inner control loop) which have to be met despite CORBA's and Java's variations in communication latency and execution times. Both Java applets are connected to the "Balancing Robots" via gateways implemented as "Composite Objects". Fig. 9 shows the communication structure of the complete application.

Problematic in our scenario is the potentially varying number of web-clients accessing the service. This may result in inacceptable high load on the real-time system and — in turn — in missed deadlines (indicated by stopped/crashed robots which ran out of fuel). To circumvent these problems we use three strategies inside the "Composite Objects" bridging between CORBA and the soft real-time application:

- caching:

  Due to the periodic operation of the soft real-time system it is sufficient to read status information once per period. Thus, the "Composite Object" serves as data cache, whose contents is valid for a limited time (the length of one period). That results in a bounded load for the soft real-time simulation.

- call admission:

  Even when using the caching approach mentioned above, incoming CORBA requests place some load on the real-time system (e.g. scheduling overheads, processing of network protocols). To bound these loads, CORBA requests can be handled by a "virtual processor" created by the Scheduling Server's [21] time-slicing scheme on top of the operating system's scheduler. We have used both ways to implement call admission within the "Composite Objects" used by our examples application.

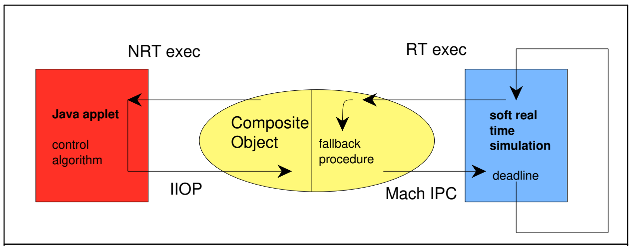

Fig. 10: *Java controller - execution scheme with fallback procedure*

- fallback procedures:

  In case of the Java-based controller, variations in communication and execution times due to CORBA and Java effect the soft real-time simulations inner control loop and may result in missed deadlines. We have used the technique of fallback procedures depicted in Fig. 10 to deal with those cases. In our approach, the fallback procedure is invoked simultanously with the CORBA invocation of the Java control algorithm. In case of a late reply, the fallback procedure's result is used instead of the Java algorithm's. However, in contrast to the Java algorithm, the fallback procedure's resource requirements and execution times are a priori known and, therefore, can be factored into the real-time application's scheduling analysis.

In Fig. 11 we compare the latency of CORBA requests sent to the Java-based controller with the "Balancing Robots" simulation's period. We see, that in case of an unloaded system running the Java controller, CORBA latency is well below the real-time simulation's period in most cases. Even on occasional delayed calls, the result is available before start of the simulations next period.

In contrast, in case of a loaded system running the Java controller, execution results are late in many cases. Without application of a fallback procedure, the "Balancing Robots" behavior would have been significantly disturbed. However, our experiments indicate stable behavior of the simulation and demonstrate the value of our technique to deal with variations in communication and execution times typical for Web-based environments.

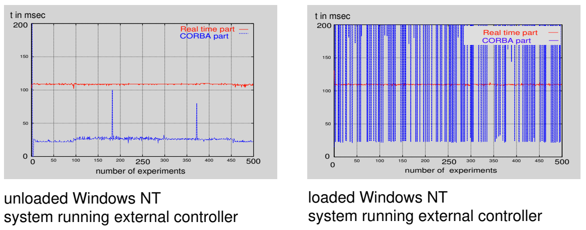

Fig. 11*: Execution of Java controller versus simulation's period*

The "Balancing Robots" Web-service has been implemented on top of the Mach-based NeXTSTEP 3.3 operating system. On NeXTSTEP we have used the ILU 2.0 alpha 12 CORBA implementation, whereas the Java-applet connecting to our Web-service has been implemented based on the Java ORB of ORBacus [16]. Fig. 12 shows screendumps of the Java components.

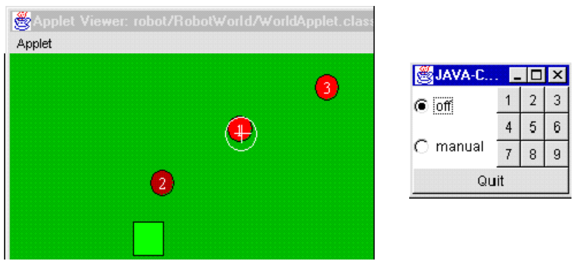

Fig. 12*: Screendump of Java components for the Balancing Robots*

7. Conclusions

The emerging World Wide Web has evolved to a point, where many users and applications demand for higher predictability and end-to-end quality-of-service properties which can not be achieved by just using faster network technologies. Within this paper, we have investigated architectural approaches for highly-available endsystems for Web-services based on standard middleware such as CORBA, or DCOM.

We have presented the CORBA-based "Observer" approach, a generic solution for implementation of reliable applications (FT Netscape web-client in our case). "Composite Objects" represent a viable technique to make CORBA-based Web-services predictable in its resource utilization and timing behavior. Bridging the gap between (legacy) real-time fault-tolerant computing systems and Web-based clients is a major precondition for remote operations in many application domains like banking, medicine, manufacturing, and booking systems. Our middleware uses commercial off-the-shelf (COTS) technology and aims at optimizing quality-of-service attributes such as timeliness and reliability.

Using principles of noninterference, interoperability and adaptive abstraction we have presented Java-based Web-interfaces to the "Balancing Robots" soft real-time simulation. Also, we have described a fault-tolerant version of the Netscape Navigator based on our "Observer" technique and the "ObservableObject" interface.

Acknowledgments

The authors would like to thank their colleagues Matthias Werner from Humboldt University of Berlin and Lui Sha and Kurt Wallnau from the Software Engineering Institute (SEI) at Carnegie Mellon University (CMU) for their insightful discussions leading to the work presented here.

References

[1] 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.
[2] K.P.Birman; *The Process Group Approach for Reliable Distributed Computing*; Communications of the ACM, pp.37-53, Vol. 36, No.12, December 1993.
[3] Z.Canela, E.Bertrand, J.Maisonneuve; *Realtime CORBA submissions comparison*; ftp.omg.org/pub/docs/realtime/98-03-01, Alcatel Alsthom Recherche, March, 1998.
[4] Y.-S.Chang, D.Liang, W.Lo, G.-W.Sheu, S.-M.Yuan; *A Fault-Tolerant Object Service on CORBA*; Proceedings of International Conference on Distributed Computing Systems (ICDCS'97), Baltimore, May 1997.
[5] H.Chu and K.Nahrstedt; *A Soft Real Time Scheduling Server in UNIX Operating System*; Technical Report, University of Illinois, UIUCDCS-R-97-1990, March 1997.
[6] IONA; *Orbix: Overview*; http://www.iona.ie/info/products/orbix/index.html IONA Technologies, Dublin, Ireland.
[7] S.Maffeis; *A Flexible System Design to Support Object Groups and Object-Oriented Distributed Programming*; in Proceedings of ECOOP'93, Lecture Notes in Computer Science 791, 1994.
[8] 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.
[9] J. McGoogan (ed.); *Realtime CORBA - A White Paper - Issue 1.0*; OMG Realtime Platform SIG, Initial Review Draft, November, 1996.

[10] K.Moore and E.Kirshenbaum; *Building Evolvable Systems: The ORBLite Project*; Hewlett-Packard Journal, February 1997.
[11] P.T.Murray, R.A.Fleming, P.D.Harry, and P.A.Vickers; *Somersault: Enabling Fault-Tolerant Distributed Software Systems*; Hewlett Packard Laboratories, Technical Report HPL-98-81, http://www.hpl.hp.com/ techreports/98/HPL-98-81.html.
[12] Jamie Zawinski; *Remote Control of UNIX Netscape*; Netscape Communications Corporation, Mountain View, California, http://home.netscape.com/newsref/ std/x-remote.html, December 1994.
[13] OMG; *Fault tolerant CORBA Using Entity Redundancy RfP*; OMG Document orbos/98-04-01, at http://www.omg.org
[14] OMG; responses to *Realtime CORBA 1.0 RfP*; OMG Document orbos/97-06-01, articles at http://www.omg.org
[15] OMG; *The Common Object Request Broker: Architecture and Specification*; Object Management Group, Inc., Framingham, MA, USA, 1995.
[16] OOC; *ORBacus C++/Java: An open architecture for distributed solutions*; Object Oriented Concepts, Inc., http://www.ooc.com/ob/, 1999.
[17] 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.
[18] A.Polze, L.Sha; *Composite Objects: Real-Time Programming with CORBA*; Proceedings of 24th Euromicro Conference, Network Computing Workshop, Vol.II, pp.: 997-1004, Vaesteras, Sweden, August 25-27, 1998.
[19] A.Polze, G.Fohler, M.Werner; *Predictable Network Computing*; Proceedings of International Conference on Distributed Computing Systems (ICDCS'97), pp.: 423-431(9), Baltimore, May 1997.
[20] R. van Renesse, K.P.Birman; *Fault-Tolerant Programming using Process Groups*; in F.Brazier, D.Jones (Eds.) *Distributed Open Systems*, Computer Society Press, 1994.
[21] 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.
[22] D.C.Schmidt, D.Levine, and S.Mungee; *The Design and Performance of Real-Time Object Request Brokers*; Computer Communications, Volume 21, No. 4, April, 1998.
[23] 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.
[24] 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.
[25] R.M.Soley; *Are Real-Time Objects Ready for Prime Time*; Keynote speech at 1st IEEE International Symposium on Object-oriented Real-time distributed Computing, Kyoto, Japan, April 1998.
[26] R.M.Soley (ed.); *Object Management Architecture Guide*; Third Edition, John Wiley & Sons, New York, 1995.
[27] M.Squadrito, L.Esibov, L.C.DiPippo, V.F.Wolfe, G.Cooper, B.Thuraisingham, P.Krupp, M.Milligan, R.Johnston; *Concurrency Control in Real-Time Object-Oriented Systems: The Affected Set Priority Ceiling Protocols*; Proceedings of First IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC), pp. 96-105, April 1998, Kyoto, Japan, IEEE Comp. Soc. Press, ISBN 0-8186-8430-5.
[28] 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.
[29] M.Werner, M.Malek; *The Unstoppables - Responsiveness by Consensus*; HUB Informatik-Berichte No.: 90/97, 19 pages, ISSN 0863-095 90, Berlin, December 1996.
[30] 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.