Proceedings of the IEEE Workshop on Middleware for Distributed
Real-Time Systems and Services, December 1997.
(Based on research performed at UIUC.)

Providing for an Open, Real-Time CORBA

W. Feng, U. Syyid, and J. W.-S. Liu
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
{*feng, syyid, janeliu*}@cs.uiuc.edu

Abstract

While CORBA provides an infrastructure which allows objects to communicate, independent of the specific techniques, languages, and platforms used to implement the objects, it is not yet suited for real-time applications since CORBA lacks essential quality-of-service (QoS) features. Current work on real-time CORBA includes an off-line scheduled, hard, real-time system based on rate-monotonic scheduling and an on-line scheduled, best-effort, real-time system based on the earliest-deadline-first algorithm. The former provides QoS guarantees at the expense of run-time scheduling flexibility while the latter provides the complement. In this paper, we propose an approach which provides the advantages of both, that is, QoS guarantees and run-time scheduling flexibility.

1 Introduction

The Common Object Request Broker Architecture (CORBA) [6], an evolving standard for distributed object-computing middleware, is a specification of an architecture and interface that allows an application to operate on objects (servers) in a transparent, independent manner, regardless of platform, operating system or locale considerations. This middleware, which resides between clients and servers, simplifies and reduces the cost of application software development by providing a uniform view of a heterogeneous environment. The capability of simplifying and reducing the cost of software development is critical as software cycle times are getting shorter and shorter for increasingly complex software. This trend is mainly driven by competitive forces of the market and a need for more functionality in software. A recent IEEE survey found that 30% of all software development projects are cancelled, 50% are more than 150% over budget, and only 60% of desired functionality on average is achieved [15].

Many researchers have recognized that while CORBA is well-suited for conventional non-real-time applications, the same cannot be said for real-time applications. Thus, over the past few years, there has been a movement to standardize middleware for real-time technologies. In particular, the Object Management Group (OMG) recently put out a Request for Information (RFI) to solicit proposals for the need of real-time capabilities in CORBA [7]. The results of which are available in [8].

2 Related Work

In order to shorten the software development time of real-time applications, programmers need to automate as much of the development process as possible with maximum code re-use. CORBA provides for such an environment; however, to provide real-time services, CORBA must be enhanced to provide specifications for quality-of-service (QoS), facilities to enforce QoS and real-time execution, and better performance in general.

The work in real-time CORBA has taken two major directions: the design of ORBs which can support hard and soft real-time applications using static scheduling and the design of ORBs which support dynamic scheduling but with best-effort enforcement of timing constraints in applications. The former is implemented in a real-time middleware framework called TAO (The ACE ORB) [16], and the latter is part of the NRaD/URI real-time CORBA system [5, 18].

Other related work in QoS distributed computing, but not directly related to CORBA, includes the EPIQ project at the University of Illinois [13, 3] and the ARMADA project at the University of Michigan [1].

2.1 TAO

The ACE ORB (TAO) by Schmidt et al. [16] addresses many of the issues involved in providing support for hard and soft real-time applications in a static environment. Schmidt identifies three major weaknesses in CORBA which prevent its use in real-time applications: QoS specification and enforcement, real-time scheduling, and performance. The performance

problems, which caused priority inversion and non-determinism in several areas of the ORB, have been identified and rectified [16]. Currently, TAO relies on an off-line, rate-monotonic (RM) scheduler to guarantee that the deadlines of real-time tasks will be met on a particular object, and hence, guarantee the QoS for that object. While this approach works exquisitely for real-time applications like avionics [16], it is not well-suited for real-time applications which require dynamic admission of clients at run-time, efficient handling of changing resource requirements, or availability of other scheduling policies for applications which may require them. In addition, the scheduling in TAO focuses on a single CPU rather than the distributed scheduling problem.

2.2 NRaD/URI Real-Time CORBA

Wolfe's NRaD/URI RT-CORBA system [19], on the other hand, uses the earliest-deadline-first (EDF) scheduling algorithm and allows clients to be admitted at run-time; hence, many of its benefits and drawbacks are complementary to those of TAO. Due to the dynamic nature of NRaD/URI's RT-CORBA, their on-line EDF scheduler does not offer the guarantees of an off-line RM scheduler and unfortunately can behave non-deterministically under heavy loads, thus providing only best-effort guarantees [18]. Another potential drawback is the need for global knowledge of the real-time constraints of each application to perform schedulability analysis for the system.

2.3 EPIQ

The EPIQ [13, 3] framework consists of an end-to-end quality-of-service (QoS) management architecture for complex, distributed real-time systems. The system handles all types of scheduling policies and guarantees QoS to all real-time applications. Its open-system approach allows real-time applications which use different scheduling policies to be validated *independently* of each other, thus making it possible to easily add new applications to the system during run-time by examining the real-time requirements of the application independent of all other applications.

Specifically, EPIQ provides a two-level scheduling scheme where each application is assigned a constant utilization server or total bandwidth server [3] at the upper level. The applications themselves may use any scheduling policy which is suitable for the application, e.g., RM, EDF, cyclic executive. At the lower level, the OS scheduler maintains and schedules each of the servers by an EDF policy. This approach allows a gamut of real-time applications (from soft to hard real-time with static and dynamic priorities) to be developed and validated independently, particularly at run-time. For example, a user may request the start of a real-time application whose schedulability has yet to be analyzed with the real-time applications currently running on the system. Based on a schedulability test, the system admits the request only if the new application and all existing real-time applications are schedulable.

On the other hand, the above scheduling scheme is currently designed to run on a uniprocessor machine and thus is not directly suitable to run the kinds of distributed applications that CORBA systems are used for. In addition, it is a non-CORBA compliant system.

3 Approach

Based on the current state-of-the-art, there exists no single ORB system that guarantees real-time performance *and* supports all types of real-time applications and scheduling policies dynamically without having to reconfigure the system. Our approach addresses this void. First, we extend the TAO system from being a closed system with off-line scheduling to an open system with on-line scheduling. Such a system would allow clients to be added (and removed) dynamically via an admissions test at run-time and would ensure scheduling feasibility. Second, we show how some of the scheduling principles from EPIQ can be adapted and applied to the extended TAO system to provide a more open system which can deal with an even wider range of real-time applications with potentially different scheduling policies.

3.1 Dynamic Clients

To allow new client requests to be handled on-line, we propose a modification to the real-time scheduling scheme provided in TAO. In addition to using TAO's Scheduling Service (SS) in a deterministic, static, and off-line manner for hard real-time applications, we also instantiate an SS object at run-time for each server object. This run-time SS object can be viewed as a *scheduling broker* [14] for the server object because it performs schedulability analysis on behalf of the server object. That is, as with the off-line SS object, the run-time SS object, or scheduling broker, performs schedulability analysis on all IDL operations that register with it in order to produce a schedule for the run-time scheduler. If a new request for an IDL operation on a server object fails the schedulability test, the client's request for that IDL operation is rejected. In short, this extension of TAO's SS allows clients' real-time tasks to be added to a server object's run-time schedule while maintaining real-time performance, i.e., meeting deadlines and guaranteeing QoS.

For example, Figure 1 shows a scenario with three clients, a server, and the server's scheduling broker.

Before run-time, the off-line SS creates a real-time schedule for the server based on the QoS parameters of Clients 1 and 2, as in [16]. At run-time, this schedule is then used by the server's run-time scheduler to schedule Client 1 and Client 2 IDL operations (or tasks). When Client 3 makes a new request for an IDL operation on the server object, the request must first go through the scheduling broker for schedulability analysis in order to see if all the existing IDL operations from Clients 1 and 2 as well as the new IDL operations from Client 3 can be scheduled by their deadlines. Next, the scheduling broker indicates to Client 3 whether or not it has been admitted to the system with its current set of QoS parameters. If not, then Client 3 must wait for some of the server's resources to free up or modify its QoS parameters and try again. If, on the other hand, Client 3 is admitted, then the scheduling broker forwards a new schedule (which accounts for Client 3's IDL operations) to the server object for the run-time scheduler to execute. Once this is done, the server indicates readiness to Client 3, and at this point, Client 3 communicates its IDL operations directly to the server just as Clients 1 and 2 do.

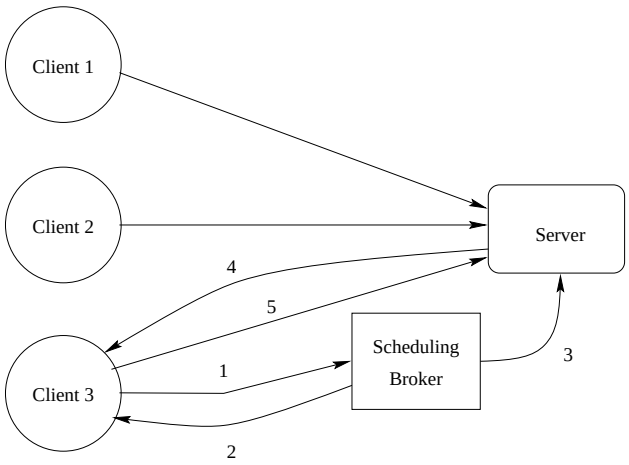

Figure 1: A Run-Time View of the Scheduling Broker

To further ensure that real-time tasks meet their deadlines, the scheduling broker's server object can employ the imprecise-computation technique to trade off service quality for information timeliness [12, 4]. Since many large, complex, and distributed real-time application systems are built based on an estimate of the worst-case load level, transient overloads can occur whenever the actual load level of the system temporarily exceeds its estimated worst-case load level. During these transient overloads, the imprecise-computation technique prevents missed deadlines and provides graceful degradation by ensuring that an approximate result of acceptable quality is available whenever the exact result cannot be obtained in time. The details of creating such an imprecise-computation server can be found in [9].

3.2 An Open System

Until recently, many classical scheduling algorithms required that schedulability analysis be done globally by analyzing all the real-time applications in the system [11, 10, 17]. We call such a system a *closed* system because all the real-time applications on each processor are known *a priori*. This is in contrast to an *open* system where applications may be dynamically added at run-time and validated independently of the other applications in the system.

We propose to further extend TAO to not only handle the dynamic admission and removal of clients via a scheduling broker but also to provide for an open system by enabling the dynamic handling of different types of real-time applications without *a priori* knowledge of global application requirements. This would allow a server object to be instantiated and added to the system regardless of what its scheduling policy is. To provide for such an open system, we modify the uniprocessor scheduling scheme found in EPIQ to handle server objects (with their respective scheduling brokers) rather than server applications. This modification would then allow server objects to be scheduled and validated independently of one another in a dynamic manner.

3.2.1 Admissions Testing and Scheduling

At the lower level, when the ORB receives a request from a client for service, it first checks to see if an instance of the required server object type exists. If such a server object has not been instantiated, then a new server object of the required type is instantiated, and it undergoes an admissions test. The new server object is only accepted into the system if it passes this test.

To perform the admissions test, each client's request must include the speed of a slower virtual processor whose speed is a fraction of the speed of the physical processor. The idea here is to set the fractional processor bandwidth that the corresponding server object will use to satisfy the client's request. The system, using this speed and the current utilization of the processor, can then determine if the new request can be scheduled by the system. Furthermore, this decision can be made independently of the other server objects in the system. Other factors that may affect the schedulability include the presence of aperiodic tasks

and the maximum length of all non-preemptable sections guarding global resources. In any case, the aforementioned admissions test is simple and is of the same variety that the two-level scheduler in EPIQ performs for real-time applications which enter the system, independent of all other applications currently running on the system [13]. So, rather than having to examine the schedulability of every combination of server objects together globally and *a priori*, each server object can be validated independently at run-time.

At the upper level, each server object will have associated with it a server scheduler and a scheduling broker object, as described in Section 3.1. The server scheduler will perform scheduling for the server object based on the algorithm which is most suitable for the application. (This algorithm is chosen by the code developer.) At the lower level, where the admissions test is performed, the system scheduler uses an EDF policy to schedule between all the server schedulers in the system. Essentially, the system provides a slower virtual processor to each server object running on the system which can satisfy the real-time constraints of the application.

In addition, as introduced in Section 3.1, if new client requests come into the system for the same server object, they are handled by the scheduling broker for the particular server object. Again, these requests will need to specify the speed for the slower virtual processor on which this new task can be run. The scheduling broker will then renegotiate with the system to determine if this now increased speed can be offered to the currently running server object. The system performs the same test using the current utilization of the processor and the required speed by the new client application and then determines if the task is schedulable. If so, it informs the server object of the new client request, and if the client is accepted informs the client of its admission into the system. After acceptance, the client application may then directly communicate with the server object. If the server scheduler is implementing a static scheduling policy in which it has created a schedule a priori, the scheduling broker object must create a new schedule for the server object which incorporates the new incoming client request and forward this schedule to the server scheduler. In short, this provides an extension to Schmidt's static real-time SS and allows real-time tasks to be added to a server object's run-time schedule while ensuring the same QoS at run-time.

3.2.2 Run-Time Sequencing

When a server object is instantiated by the object adapter, a queue for incoming requests for each server object is created in the I/O subsystem. Incoming requests are queued in the I/O system based on the connection socket which has been established between the client and server objects. When a request arrives, it is first de-multiplexed by the I/O subsystem into its respective queue. When the EDF scheduler schedules the respective server object, it must first retrieve jobs (i.e., requests) from its corresponding I/O queue and move them into its own privately maintained queue. To accomplish this movement, a queued request must pass through the Object Adapter which de-multiplexes the request to the appropriate server queue. Hence, when a server object is to be scheduled, the EDF scheduler ensures that the first pending request in its I/O queue is sent up to the server object's own queue to be scheduled in accordance with the server object's scheduling policy.

For example, Figure 2 illustrates the run-time sequencing of the ORB system. First, the EDF scheduler determines that the next server to run on the CPU is Server1. After determining that Server1 is to be run on the system, the ORB then allows the new request in the I/O queue of the scheduled server object, in this case Server1, to pass through the Object Adapter to the local job queue in Server1. The new request is enqueued in Server1's privately maintained queue and is scheduled according to the server object's scheduling policy. The next job, which is determined by the server object's scheduling policy, is scheduled and run on the processor.

3.2.3 Future Work

One major problem that such an open system would have is that since the server scheduler of each server object must be able to handle any scheduling policy, that scheduling policy will have to be written as part of the server object (by an application developer, for example) or as a CORBA scheduling-service object. This, however, would defeat one of the purposes of CORBA systems, namely to build less complex software by re-using previously built components. The scheduling-service object could not be run as a real-time CORBA object due to the real-time constraints that it must be able to handle. One notion is that CORBA services be extended to include real-time services which would include all major scheduling policies; however, this would mean that the ORB core would be much more complex and heavier, leading

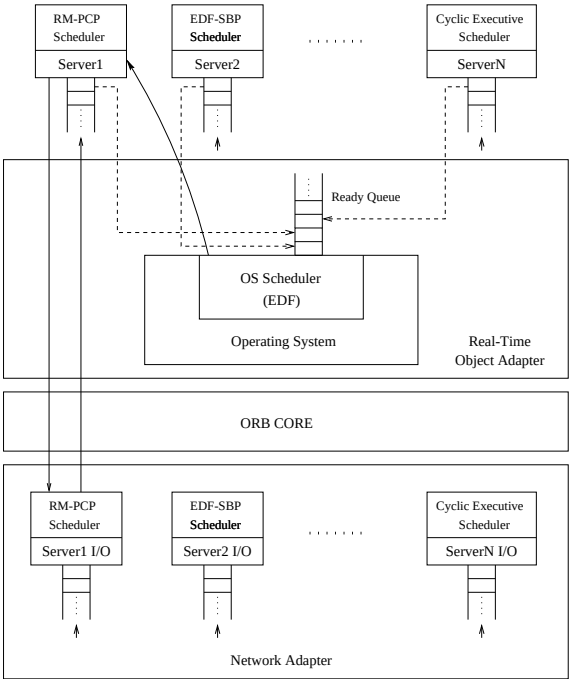

Figure 2: Run-Time Sequencing of the ORB System. Note: This figure does not show the scheduling brokers.

to slower execution, and applications would be constrained to use only these scheduling policies.

The critical component in the run-time operation of the system is the request dispatching. This includes the presentation layer conversions (de-marshalling) and the de-multiplexing to the appropriate server object. Instead of using a layered de-multiplexing scheme i.e., de-multiplex to server and then to operation, demultiplexing can be performed using the active demultiplexing scheme in O(1) time and thus is deterministic [16].

The presentation layer has been recognized as one of the major bottlenecks in high-performance communication sub-systems [2], and several optimizations have been suggested to improve performance, however most of these lead to improved, albeit non-deterministic performance. TAO, for example, caches pre-marshalled application data units to improve performance, this can however lead to non-determinism. Another tradeoff is between using compiled or interpreted code for marshalling. Although compiled code is more efficient, it tends to be larger, an important consideration in embedded real-time applications. The TAO system performs several off-line optimizations based on measures of run-time usage and then chooses between compiled or interpreted stubs and skeletons. In the above-proposed system, these optimizations are not possible as this analysis cannot be done on-line.

The above considerations are all the more important in the system proposed above as the request dispatch time must be subsumed into the virtual processor allocated time that each server has to run its request in and must be included in the admissions test. If there is any non-determinism at this stage, it may lead to incorrect admission and missed deadlines. Thus these problems must be given careful consideration.

4 Summary and Conclusions

In this paper, we compared and contrasted three end-to-end quality-of-service frameworks: TAO by Schmidt et al., NRaD/URI by Wolfe et al., and EPIQ by Liu et al. Both TAO and NRaD/URI use CORBA to provide a uniform view of a heterogeneous environment, but they take complementary approaches toward realizing a real-time CORBA. First, TAO uses static real-time scheduling (RM) to meet the hard and soft deadlines of an avionics system while NRaD/URI uses dynamic real-time scheduling (EDF) which amounts to a best-effort approach towards enforcing timing constraints. Second, in TAO, the scheduling is done off-line and is not amenable to unplanned changes during run-time whereas the scheduling in NRaD/URI is done on-line and allows tasks to enter freely without admission control. The EPIQ system is able to handle a wide range of real-time applications including soft and hard real-time using all kinds of scheduling policies, however the EPIQ has been designed for a uniprocessor machine and is not CORBA-compliant.

We further, provide suggestions, to allow the TAO system to handle the scheduling and re-configuration of client requests at run-time, consequently providing for a more dynamic system. Furthermore suggestions have been made to provide a more open environment for real-time application development in which real-time objects can be developed independently of each other, each with their own scheduling and resource allocation policy. These heterogeneous real-time objects may then be used together to efficiently build a complete real-time application.

Our proposals would thus extend real-time CORBA systems so that they may use application-dependent scheduling policies to enforce clients' timing constraints. Several areas of concern, such as how to handle all these scheduling policies without increasing program complexity have been recognized as areas in which future work must be done. More simulation

and implementation work will have to be done to gain better insight into the proposed changes to real-time CORBA.

References

[1] T. Abdelzaher, S. Dawson, W-C. Feng, F. Jahanian, S. Johnson, A. Mehra, T. Mitton, A. Shaickh, K. Shin, Z. Wang, and H. Zou. Armada middleware suite. In *IEEE Workshop on Middleware for Distributed Real-Time Systems and Services*, December 1997.

[2] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proceedings of the Symposium on Communications Architectures and Protocols (SIGCOMM)*, pages 200–208, September 1990.

[3] Z. Deng and J. W.-S. Liu. Scheduling real-time applications in an open environment. In *Proceedings of the 18th IEEE Real-Time Systems Symposium*, December 1997.

[4] W. Feng and J. W.-S. Liu. Algorithms for scheduling real-time tasks with end-to-end deadlines and input error. *IEEE Transactions on Software Engineering*, 23(2):93–106, February 1997.

[5] R. Ginis, V.F. Wolfe, and J. J. Prichard. The design of an open system with distributed real-time requirements. In *IEEE Real-Time Systems Symposium*, 1996.

[6] Object Management Group. The common object request broker: Architecture and specification, 2.0. Technical report, OMG, July 1995.

[7] Object Management Group. Realtime technologies: Request for information. Technical report, OMG, February 1997.

[8] Object Management Group. Responses to realtime technologies: Request for information. Technical report, OMG (www.omg.org), February 1997.

[9] D. L. Hull, W. Feng, and J. W.-S.. Liu. Enhancing the performance and dependability of real-time systems. In *IEEE Computer Performance and Dependability Symposium*, April 1995.

[10] J. P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm — exact characterization and average case behavior. In *Proceedings of 10th IEEE Real-Time Systems Symposium*, pages 166–171, December 1989.

[11] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

[12] J. W.-S. Liu, K.-J. Lin, W.-K. Shih, A. C.-S. Yu, J.-Y. Chung, and W. Zhao. Algorithms for scheduling imprecise computations. *IEEE Computer*, 24(5):58–68, May 1991.

[13] J. W.-S. Liu, K. Nahrstedt, D. Hull, S. Chen, and B. Li. Epiq qos characterization: A draft version. Technical report, University of Illinois at Urbana-Champaign, July 1997.

[14] K. Nahrstedt. Experiences with qos brokerage and enforcement. In *Proceedings of 2nd International Conference on Multimedia Information Systems*, April 1997.

[15] ObjecTime. Overcoming the crisis in real-time software. *White Paper*, 1997.

[16] D. C. Schmidt, D. L. Levine, and S. Mungee. The design of the tao real-time object request broker. *Computer Communications Journal*, Summer 1997.

[17] B. Sprunt, Sha L., and J. P. Lehoczky. Aperiodic task scheduling for hard real-time systems. In *Real-Time Systems: The International Jounal of Time-Critical Computing Systems*, volume 1, pages 27–60, January 1989.

[18] V. F. Wolfe, L. C. DiPippo, R. Ginis, M. Squadrito, S. Wohlever, I. Zykh, and R. Johnston. Expressing and enforcing timing constraints in a dynamic real-time corba system. Technical report, University of Rhode Island, June 1997.

[19] V. F. Wolfe, L. C. DiPippo, R. Ginis, M. Squadrito, S. Wohlever, I. Zykh, and R. Johnston. Real-time corba. Technical report, University of Rhode Island, June 1997.