Integration of CORBA Services with a Dynamic Real-time Architecture

Andreas Polze, Janek Schwarz, Kristopher Wehner[‡], and Lui Sha[‡]

December, 16, 1999

Departement of Computer Science
Humboldt University of Berlin
Rudower Chaussee 25
12489 Berlin, Germany
{apolze|schwarz}@informatik.hu-berlin.de

[‡]Departement of Computer Science
University of Illinois Urbana-Champaign
1304 W. Springfield Ave.
Urbana, IL 61801
{wehner|lrs}@cs.uiuc.edu

Abstract

The Common Object Request Broker Architecture (CORBA) is the most successful representative for an object-based distributed computing architecture. Although CORBA simplifies the implementation of complex, distributed systems significantly, support of techniques for reliable, fault-tolerant software, such as online replacement or replication is not within scope of today's CORBA or Real-time CORBA.

The Simplex architecture developed at the Software Engineering Institute at Carnegie Mellon University supports the online-replacement of software components within a fault-tolerant, real-time control application. Simplex middleware consist of both a hard real-time publisher/subscriber service and a soft real-time service to support human-in-the-loop supervisor control activities.

We have replaced Simplex' soft real-time services with CORBA services while retaining its hard real-time publication and subscription service. This composite approach allows us to take advantage of both the strength of

CORBA in distributed computing and Simplex' strength in hard real-time control applications.

In this paper, we will discuss the trade-off issues in this composite system approach and present the tele-laboratory experiment, which is based on the extended Simplex system.

1 Introduction

A significant class of distributed real-time applications consists of hard real-time sensor and actuator loops within a node and a soft real-time supervisory control for nodes involving human in the decision loop. Examples of these type of applications include manufacturing and transportation management systems.

At the supervisory level, the Common Object Request Broker Architecture (CORBA) is appealing. CORBA simplifies the implementation of complex, distributed systems significantly. From the timing perspective, it is also adequate for human-in-loop soft real-time requirements.

Within a node, there are hard real-time requirements. In addition, for advanced real-time systems, we would like to provide the capability of dynamically replacing software implemented controllers in a fault tolerant way, so that advanced control technologies can be easily and reliably introduced into long life cycle plants. Currently, neither CORBA nor its real-time extensions address the notion of fault tolerant dynamic replacement of real-time software components. Fortunately, these issues are addressed by the Simplex architecture [Sha96] developed at Software Engineering Institute. The Simplex architecture has been designed to tolerate timing faults such as overrun, programming system faults such as illegal addressing, and semantic faults due to modeling, algorithm design or implementation errors.

In this paper we present the integration of the Simplex's real-time fault tolerant online-replacement mechanisms with CORBA to create an application environment known as tele-laboratory. This project is our first step towards the development of the fault tolerant

online-replacement of CORBA components in a generalized Simplex architecture. Replacement decisions within Simplex are based on non-functional component properties, such as timing behavior and resource usage, which cannot be expressed easily using interface descriptions such as CORBA IDL. We briefly discuss the applicability of aspect-oriented programming techniques for description of these additional, non-functional component properties.

The remainder of the paper is organized as follows: Section 2 presents related work. Section 3 presents the composite system approach to implement online-replacement in a CORBA setting and discuss the tradeoffs between a system's openness and its predictable behavior. Section 4 describes a tele-laboratory case study.

Aspect-descriptions for components in a Simplex system are discussed in Section 5, whereas Section 6 gives directions for future work. Section 7 finally presents our conclusions.

2 Related Work

Fault-tolerant and Real-time 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* [OMG98a] issued in April 1998, OMG is seeking to incorporate existing approaches for software fault tolerance into future versions of CORBA. However, in contrast to our approach which focuses on online-replacement of components, most related work implements fault-tolerant behavior based on redundant execution and fault-masking. The integration of fault-tolerance techniques with real-time computing is currently not under consideration in the OMG standardization process.

Electra [Maffeis94]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 [Birman93]and HORUS [van Renesse94]systems.

ORBIX+ISIS is an extension of the commercial ORBIX [IONA]ORB implementation, which introduces concepts like object groups and group communication based on the ISIS [Birman93]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 [Chang97]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.

The Object Management Group (OMG) has founded a Real-time 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 [OMG99].

Work at the University of Rhode Island and the MITRE Corporations deal with syntactical extensions to CORBA IDL to express timing constraints [Squadrito98] [Thuraisingham96]. “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.

TAO is an innovative work on RT-CORBA where fixed priority real-time scheduling is tightly integrated into the system [Schmidt98] [Schmidt99]. 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 focuses rather on completely new, CORBA-based real-time systems, than on interfacing an existing real-time system with CORBA.

The Simplex Architecture

Current real-time computer software architectures do not support the safe and reliable insertion of new components and technologies into existing systems. The Simplex architecture [Sha96] has been developed for automatic control applications to support safe and reliable online upgrade of software components in spite of errors in new modules. This is important when introducing new control technologies into running systems.

To mitigate the risks in a control system upgrade, the Simplex architecture has been designed to tolerate timing faults such as overrun, programming system faults such as illegal addressing, and semantic faults due to modeling, algorithm design or implementation errors. Simplex handles timing faults and programming system faults by temporal and spatial encapsulations. Software semantic faults are handled by the use of analytic redundancy.

For real-time scheduling, the Simplex architecture is based on the generalized rate monotonic scheduling theory [Sha94]. Furthermore, the Simplex architecture assumes that the underlying operating system supports either the priority inheritance protocol or the priority ceiling protocol to circumvent priority inversion problems during the management of shared resources. A number of demonstration applications have been implemented using Simplex on top of the Lynx commercial real-time operating system, which supports POSIX 1003.1b.

The basic building block of Simplex Architecture (see Figure 1) is a replacement unit. A replacement unit is simply a process with a given communication template. Replacement units are organized into application units. An application unit typically has

communication and process management modules, and at least a replacement unit that implements the application functions. If the system is safety critical, a separate safety unit is needed. The safety unit is responsible for reliable operation and operation monitoring.

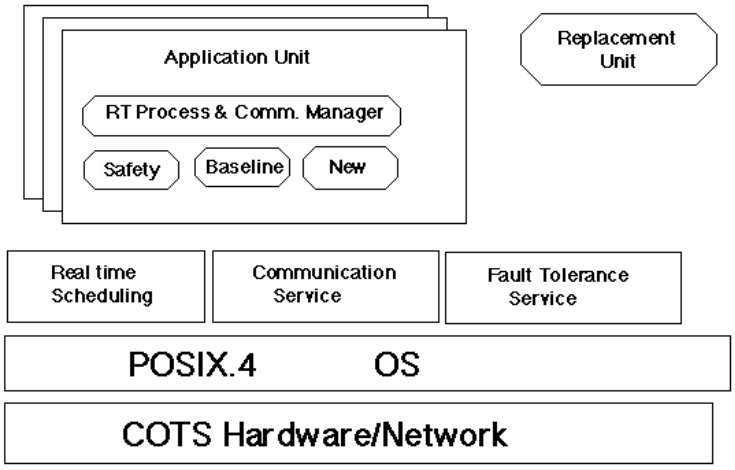


Figure 1: The Simplex framework

Simplex uses forward recovery for semantic faults in control applications. The system states under the control of the experimental controller must stay within the stability envelope of the safety controller. Intuitively, this is similar to a student pilot's flight test. The trainer will let the student pilot fly, even making some minor mistakes, as long as the plane remains in a state that is controllable by the trainer. The safety controller is designed to have a large stability envelope at the cost of performance whereas the experimental controller is designed to maximize the performance. They have different designs but analytically related design objectives.

The experimental controller is encapsulated in a replacement unit and can be replaced during runtime. The replacement of a component is not a real-time operation. However, once the component takes over the control, the computation and communication involving the component are hard real-time operations.

3 The Composite System Approach

Within context of Simplex, analytic redundancy and online-replacement have been proven as viable concepts for the construction of reliable dynamic real-time software systems. However, programming for the Simplex system requires adherence to a number of coding conventions. In order to support dynamic replacement, Simplex components (replacement units) have to support certain interaction patterns, which are expressed in actual component code, rather than in an interface definition language.

Introduction of CORBA IDL would significantly simplify programming of Simplex replacement units. However, since CORBA interactions have varying, unpredictable communication latency, some measures have to be taken to ensure Simplex' real-time behavior. Since the component replacement operation in Simplex is not a real-time operation, it can be easily replaced by a CORBA-based service to achieve a higher degree of flexibility and improved portability.

The open question is, how to interface the real-time operations with CORBA. There exist a number of object request broker implementations with real-time extensions, such as the TAO ORB. In certain system configurations, where a hard real-time control loop is statically distributed over several nodes, the RT-CORBA implementations with their specific communication protocols show great advantages over proprietary solutions. However, in systems, where the hard real-time loop is not distributed, such as in the tele-lab (see Section 4), it is sufficient if middleware provides a distributed multimedia user interface to the system and supports component management with soft real-time characteristics.

In case of our tele-lab scenario, both CORBA and RT-CORBA ORBs could have been used. However, the non-distributed nature of the real-time control loop does not require the use of RT-CORBA. In contrast to the current experimental RT-CORBA ORBs, standard CORBA implementations are more mature, better supported, and provide a more stable software development environment. The challenge in this case is the integration of standard CORBA with a proprietary real-time system. We introduce the notion of a "Composite System" to solve the latter problem.

Within the Composite System, components' interfaces are described using CORBA's interface definition language IDL. The construction of new replacement units is simplified. Additionally, many standard CORBA services (COS – common object services), such as naming service, event service, or domain specific services, can now easily be used from within the replacement units. On the other hand, we retain the real-time publication and subscription service of Simplex to ensure the timing predictability for safety controller and decision unit, Simplex's most vital components. Thus, we use Simplex's inherent concept of analytical redundancy to tolerate CORBA's unpredictable timing behavior.

To interface the real-time operations with CORBA we use a previously developed concept called "Composite Objects" [Polze98]. 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 that are usually hidden and abstracted away by CORBA visible and explicit.

The implementation of composite objects follows three basic design rules:

1. *noninterference:* General-purpose computing and real-time computing should not burden each other.

2. *interoperability:* Services exported by general-purpose computing objects and by real-time computing objects can be used by each other.

3. *adaptive abstraction:* Lower level information and scheduling details are available for real-time objects, but are transparent to non-real-time objects.

Composite objects consist of a real-time part and a non-real-time part. Design time and run-time guarantees can be given for execution of real-time methods. In contrast, methods in the non-real-time part are executed following a best-effort approach. Data replication and a weakly consistent memory management protocol (implemented via interprocess communication – such as POSIX message queues) are used to decouple real-time and non-real-time processing.

Composite objects establish timing firewalls [Polze97] between real-time and non-real-time (CORBA) computing, so that the non-real-time part cannot violate the real-time scheduling rules that are needed by the real-time part [Sha94]. Implementation details on Composite Objects and timing firewalls using a scheduling server can be found in [Polze97] and [Polze98].

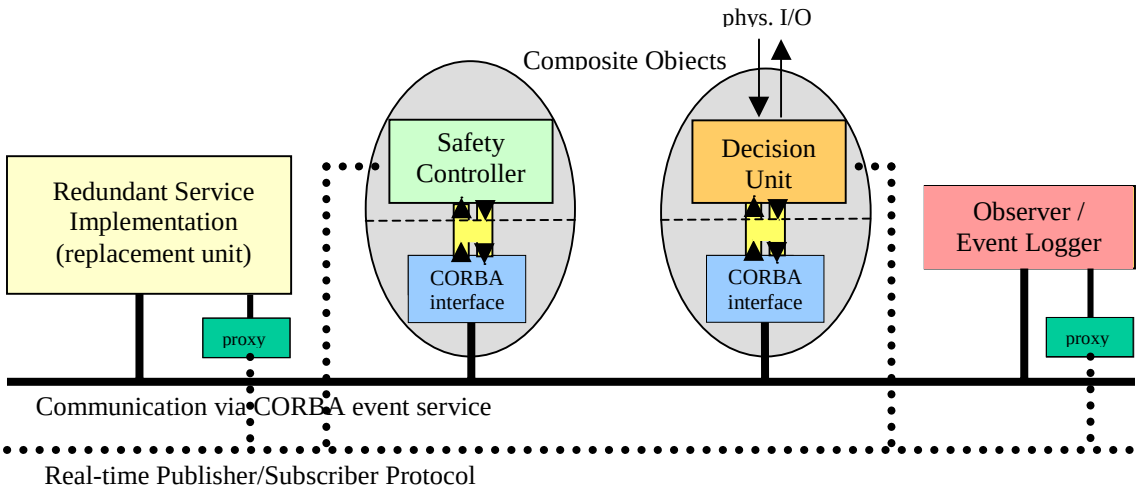


Figure 2: A composite system architecture for Simplex

In our approach towards a CORBA-based version of Simplex shown in Figure 2, we implement decision unit and safety controller inside the real-time part of composite objects. We use the real-time publisher/subscriber protocol for communication between safety and decision unit. This way we can ensure predictable timing behavior for interactions between these two essential components.

As in the original Simplex system, the safety unit implements the basic version of the desired service. Additional, more sophisticated and complex versions of a service now can be implemented as CORBA components. These components are described by their interfaces given in CORBA IDL. Multiple versions of the CORBA service implementations correspond to Simplex's replacement units and inside the decision unit Simplex's concept of analytic redundancy is used to judge a components output.

Since CORBA is involved, no guarantees about the timing behavior of the complex CORBA components can be given. However, since at least safety controller and decision

unit interact via the predictable real-time publisher/subscriber protocol, basic functionality of the service is provided in a timely fashion.

Communication among decision unit, safety controller, and additional replacement units (redundant service implementations) is based on the CORBA event service. This approach eliminates most of the dependencies on the proprietary real-time publisher/subscriber protocol. However, since the CORBA event service does not guarantee delivery of events, some measures have to be taken. A CORBA-compliant implementation of event service may use an unreliable transport (such as UDP/IP). However, for a given environment, we assume that at least an a priori known, fixed number of events will be delivered per given time unit. This means that a repeatedly (periodically) sent event eventually will reach its destination.

Proxy objects, as depicted in Figure 2, may interface CORBA components with the publisher/subscriber protocol. The support of an interface to these proxies is optional for a CORBA-based service implementation. Observation of a component's behavior through the proxy interface is greatly simplified by the fact that delivery of events via publisher/subscriber protocol is reliable in contrast to the CORBA event service. However, monitoring of a CORBA-based service implementation, which does not support the proxy interface is still possible using the Observer/Event Logger component, which periodically polls every other component for received events.

Finally, since CORBA communication is location-transparent, we can use the architecture depicted in Figure 2 to distribute the Simplex system in a heterogeneous environment. Simplex's predictable timing behavior is preserved as long as safety unit and decision unit are co-located on a real-time operating system (as in the original Simplex system).

4 Case Study

Experimentation with real physical systems is a key part in control education. Traditionally, the use of real physical systems requires students' physical presence in the laboratory. Our Tele-laboratory project allows students to remotely conduct physical

experiments via the Web easily and reliably, creating an integrated virtual and real laboratory education environment. First time users of our software might miss the “real” contact with the equipment, but on closer examination it is easy to see that there is no difference since users get visual feedback of the control device. Additionally, the software transmits debug output made by the controller. A simple time-sharing scheme ensures better usage of the equipment. The novelty of our project lies in:

1. The use of real experiments in addition to simulation and animation without being physically at the laboratory. This will be achieved through the implementation of real-time, fault tolerant control architectures that can guarantee stable performance while allowing users to download and test user written control modules.

2. Horizontal integration across engineering disciplines. Our laboratory network includes electrical, mechanical, hydraulic, computer science and other experiments that cross departmental boundaries. Our Tele-laboratories will be developed with a common interface that will allow students from differing disciplines to gain practical experience with a variety of control and distributed computing problems.

3. Integration with standard tools for controller development. Nowadays control engineers develop their software using off-the-shelf software, e.g. Matlab. Our laboratory software has interfaces to such programs and therefore allows students to continue to use the development tools they already know.

Our Tele-laboratory allows students to access equipment (control devices like an inverted pendulum or a robot) over the Web without having to travel to a physically remote location in order to carry out experiments. Using the software students can write a controller for a device, upload it to the controlling machine and try it out on physical equipment with no possibility of damaging the device.

In an instructional situation it is likely that the user-supplied controllers will malfunction, possibly in dangerous ways. Due to the reality of these dangerous malfunctions, it is

necessary that we take precautions to protect non-fail-safe laboratory equipment. Simplex provides this protection using the technique of analytic redundancy, and can detect and replace a malfunctioning controller with the predefined baseline controller to keep the equipment safe and in working order. The student is given feedback as to the state of their controller (running, failed), as well as the system states that led to the termination of their controller in the case of failure. In addition, the students are provided with a video stream of the physical device and graphs of its status. Facilities are also in place for the controller to transmit debugging information output during a run back to the student after execution completes (so as to not interfere with the real-time requirements of the student's controller).

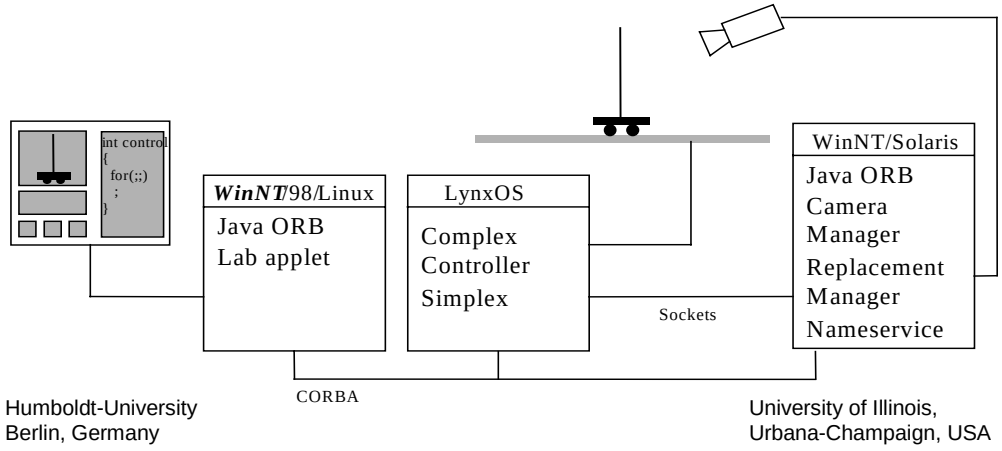


Figure 3: Tele-laboratory scenario

The scenario depicted in Figure 3 shows the inverted pendulum controlled by Simplex running on a LynxOS 3.0 system, along with the components of the Tele-laboratory system. Using our "Composite Objects" technology, the replacement manager encapsulates the user-interface of simplex and exports this functionality to CORBA objects. CORBA objects can request the compilation of controller code by communicating with the replacement manager, which also has an interface to the C compiler.

After successful compilation the user can request the execution of the controller. Simplex then executes the controller and provides the student with status information about the execution. The lab applet is the user-visible portion of the Tele-laboratory. The student

uses this application to load or develop controller code, for communicating with the replacement manager and watching the video and execution status feedback. The application provides quality of service choices to the student, in the form of multiple video streams of different bandwidth.

The actual scenario shown in Figure 3 relates to an experimental setting where the lab applet has been run on a computer at Humboldt-University of Berlin, Germany, whereas the Simplex control application was executed at University of Illinois, Urbana-Champaign. A Logitech Web camera has been used to capture the experiment's status. The WebVideo system from University of Ulm, Germany [Wolf97], has been set up to prepare and transmit the camera's video stream via the internet. Figure 4 shows a screendump of the lab applet during the experiment. In case this paper is accepted, we will demonstrate the tele-lab at RTAS.

Figure 4: Lab Applet during Tele-laboratory experiment

5 Description of a Component's non-functional Aspects

For the CORBA-based Simplex, we had to trade off data abstraction and encapsulation against implementation specific knowledge about a component's timing behavior, resource usage and interaction patterns. These non-functional component aspects are crucial for the predictable behavior of the online replacement mechanisms. In contrast to CORBA's interface definition language (IDL), which describes a component's functional interface, there is no general means to describe a component's non-functional properties.

We propose using the technique of aspect-oriented programming (AOP) [Kiczales97] to describe these properties. In contrast to a component, which can be cleanly encapsulated in a generalized procedure (i.e., object, method, procedure, API), an *aspect* describes a system property, which cannot be expressed in a standard construct used in object-oriented programming.

Within context of this paper we want to enumerate a number of aspects, which need to be taken into account for the generalized Simplex framework:

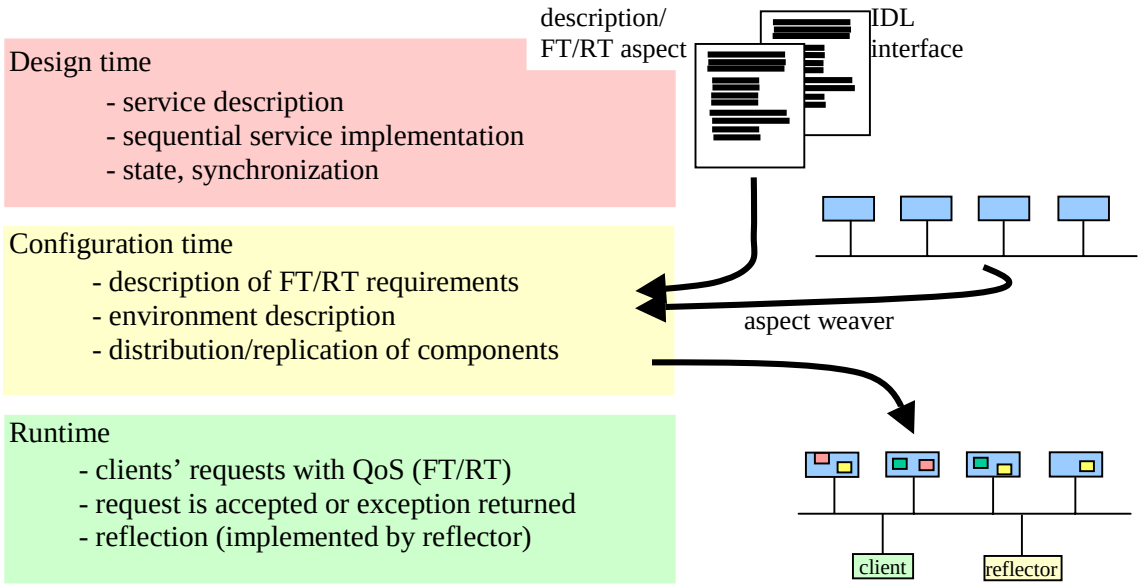
- The *fault-tolerance aspect* describes FT-techniques used by a component. It needs to be specified whether a component has internal state and whether a synchronization technique, such as checkpointing is being used. Also, criteria, which indicate that a freshly started component has reached steady state needs to be given.

- The *real-time aspect* describes a component's characteristics, such as timing behavior as expressed by average case and worst-case execution times, periods, dependencies, and resource requirements (memory, disk I/O) for all entities of the component's functional interface.

- The *replacement management* or *authority management aspect* as implemented jointly by safety and decision unit describes which algorithm should be used to judge about a component's internal state and whether the component functions

correctly. This aspect also gives criteria for the online-replacement of components.

As depicted in Figure 5, aspects are taken into consideration at three different levels in the software development process for a component-based system: The programmer specifies aspects like fault-tolerance technique used (FT) or worst case execution time



(RT; in terms of instructions) *at design time*. At *configuration time*, design-time information and aspect-information specific to the current environment (number of nodes, processing power, security characteristics) are taken into account to distribute components over the network and configure the system. This step can be automated using a tool such as an *aspect weaver*. Finally, at *runtime* the outcome of the previous steps is stored within an additional component – the *reflector*.

Figure 5: Description of a component's non-functional (fault-tolerance, real-time) aspects

The *reflector* is generated automatically and stores configuration information. It may be used for online reconfiguration (replacement) as a result of a client's request for specific quality-of-service (such as 2-out-of-3 majority voting or distributed execution in order to avoid common-mode failures). Additional work will focus on definition of an aspect language and weaver for the generalized Simplex.

6 Conclusions

The Simplex architecture developed at the Software Engineering Institute at Carnegie Mellon University supports the online-replacement of software components within a fault-tolerant, real-time control application. Simplex is based on the proprietary real-time publisher/subscriber protocol and a POSIX 1003.1b-compliant real-time operating system. Simplex implies complex coding rules for replacement units, the basic units of functionality.

Within this paper we have extended Simplex's online-replacement mechanisms and have discussed techniques for reliable replacement of services in a CORBA-based environment. Interfaces to redundant service implementations are completely described using CORBA IDL.

However, there is a tradeoff between a system's openness as provided by CORBA and its predictable timing behavior. We introduce the notion of a composite system, which consists of a predictable real-time part (safety controller, decision unit) and CORBA-based non-real-time part (redundant service implementations). Composite Objects are used to de-couple CORBA and real-time processing.

Aspect-oriented programming techniques are a promising way to describe non-functional aspects of CORBA components. We are currently studying aspect-techniques for component description in the distributed tele-laboratory case study.

Acknowledgements

This work is supported in part by the Office of Naval Research and in part by German Research Network (DFN). Many have contributed to the tele-lab project. In particular, we want to thank Mark Spong and Dan Block for their expertise in control engineering and the use of UIUC's control laboratory equipment. We also want to thank Liu Xue for the development of control software and Chris Parrott, Xiaoyan He, Mancang Tian and Joao Lopes in various aspects of system development and testing.

References

[Birman93] K.P.Birman; "The Process Group Approach for Reliable Distributed Computing"; Communications of the ACM, pp.37-53, Vol. 36, No.12, December 1993.

[Chang97] 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.

[IONA] see multiple articles at http://www.iona.ie

[Kiczales97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin; “Aspect-Oriented Programming”, In Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland. Springer-Verlag LNCS 1241. June 1997.

[Maffeis94] 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.

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

[OMG98a] OMG; "Fault tolerant CORBA Using Entity Redundancy RfP"; OMG Document orbos/98-04-01, at http://www.omg.org

[OMG98b] OMG, “The Common Object Request Broker: Architecture and Specification. Revision 2.3”, OMG (Object Management Group) document formal/98-12-01

[OMG99] OMG, “Realtime CORBA Architecture”, OMG (Object Management Group) document orbos/99-06-02.

[Polze97] Andreas Polze, Gerhard Fohler, Matthias Werner; “Predictable Network Computing”; in Proceedings of 17th International Conference on Distributed Computing Systems (ICDCS'97), Baltimore, USA, May 1997, IEEE Computer Society Press, pp: 423-431(9), ISBN 0-8186-7813-5.

[Polze98] Andreas Polze, Lui Sha; “Composite Objects: Real-Time Programming with CORBA”; in Proceedings of 24th Euromicro Conference, Network Computing Workshop, Vol.II, pp.: 997-1004, ISBN 0-8186-8646-4, Vaesteras, Sweden, August 25-27, 1998.

[van Renesse94] 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.

[Schmidt98] 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.

[Schmidt99] F. Kuhns, D. Schmidt, and D. Levine, “The Design and Performance of a Real-Time I/O Subsystem”, Proceedings of the 5th IEEE Real-Time Technology and Applications Symposium (RTAS), Vancouver ± Canada, June 2-4 1999.

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

[Sha96] L.Sha, R.Rajkumar, M.Gagliardi; "Evolving Dependable Real-Time Systems"; in Proceedings of 1996 IEEE Aerospace Applications Conference, IEEE Inc., February 1996, ISBN 0-7803-3196-6.

[Squadrito98] 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.

[Thuraisingham96] B.Thuraisingham, P.Krupp, and V.Wolfe; "Position Paper: On Real-Time Extensions to Object Request Brokers"; in Proceedings of Second Workshop on Object-Oriented Real-Time Dependable

Systems (WORDS), February 1996, Laguna Beach, CA, USA, IEEE Comp. Soc. Press, ISBN 0-8186-7570-5.

[Wolf97] Heiner Wolf, Konrad Froitzheim; “WebVideo – A tool for WWW-base Tele-cooperation”; in Proceedings of IEEE International Symposium for Industrial Electronics (ISIE), Guimaraes, Portugal, IEEE Inc., July 1997