Contents

Session I
This is a techical session

REPLICATING CORBA OBJECTS: A MARRIAGE BETWEEN ACTIVE AND PASSIVE REPLICATION*

Pascal Felber
Xavier Défago
Patrick Eugster
André Schiper

Swiss Federal Institute of Technology
Operating Systems Lab.
CH-1015 Lausanne, SWITZERLAND
{ Pascal.Felber | Xavier.Defago | Patrick.Eugster | Andre.Schiper } .@epfl.ch

Abstract: Replication is a key mechanism for developing fault-tolerant and highly available applications. In this paper, we present a replication framework for replicating CORBA objects that combines the active and passive replication techniques. We show how we have used axiomatic properties of a consensus protocol together with a generic architectural framework to bridge the gap between active and passive replication. Our framework makes it possible to dynamically associate replication techniques to individual operations of a replicated object, without requiring the client to even know that the object is replicated. Augmenting CORBA with a replication service is a big step towards reliable and interoperable distributed systems.

Keywords: replication, CORBA, fault tolerance, high availability

1 INTRODUCTION

Distributed computing is one of the major trends in the computer industry. As systems become more distributed, they also become more complex and have to deal with new

*Research supported OFES, as part of the ESPRIT project OpenDREAMS (projects 20843 and 25262).

kinds of problems, such as partial crashes and link failures. To answer the growing demand in distributed technologies, several middleware environments have emerged during the last few years. These environments however lack support for "one-to-many" communication primitives. Such primitives greatly simplify the development of several types of applications that have requirements for high availability, fault-tolerance, parallel processing, or collaborative work.

In the context of the European ESPRIT project OpenDREAMS, which aims at providing an open, distributed, reliable framework for supervision and control systems, we have developed a CORBA service to address the dependability requirements of distributed applications. This service — called the *Object Group Service* (OGS) — uses replication techniques to make CORBA objects fault-tolerant and highly available. OGS is based on CORBA's communication and addressing mechanisms, and thus offers a *portable* and *interoperable* environment.

This paper presents the two best-known replication techniques — active and passive replication — how they relate with each other, and how they have been combined and unified in a single CORBA service. These replication techniques are quite different and marrying them is a challenging task. On the protocol level, we have reduced both replication techniques to the problem of distributed consensus, which acts as a common denominator for various distributed agreement protocols and for object replication. The replication protocols of OGS are based on a generic consensus service, which provides a safe and proved algorithmic infrastructure. On the architectural level, we have abstracted the particularities of active and passive replication to ultimately marry them in a unified replication framework. Unlike other existing systems, our environment makes it possible to use both active and passive replication techniques *at the same time* in a distributed application, and to dynamically specify the replication technique *on a per-operation basis*. This scheme provides a high degree of flexibility to the application developer, allowing him to benefit from the replication technique best adapted to the semantics of each individual operation.

The rest of this paper is organized as follows. Section 2 presents background concepts about CORBA and object replication. Section 3 describes the basic mechanisms and protocols used to provide object replication. We introduce the distributed consensus as a generic solution to distributed agreement problems and how it may be used as a basic building block for object replication. We then give an overview of the group membership and group multicast facilities used for implementing object replication. Section 4 describes how CORBA objects can be replicated using the active and passive replication techniques, and how the application developer can mix both techniques in his application. Finally, Section 5 presents some concluding remarks.

2 BACKGROUND

This section gives a brief overview of the two major topics on which this paper is based. We describe the CORBA architecture and the related notion of distributed objects, and we give a short introduction to the concept of fault tolerance through replication.

2.1 Distributed Objects and CORBA

The *Object Management Architecture* (OMA) [13], specified by the *Object Management Group* (OMG), is a conceptual infrastructure for building portable and interoperable software components. This infrastructure is based on open standard object-oriented interfaces. Figure 1 shows the five major parts of the OMA reference model.

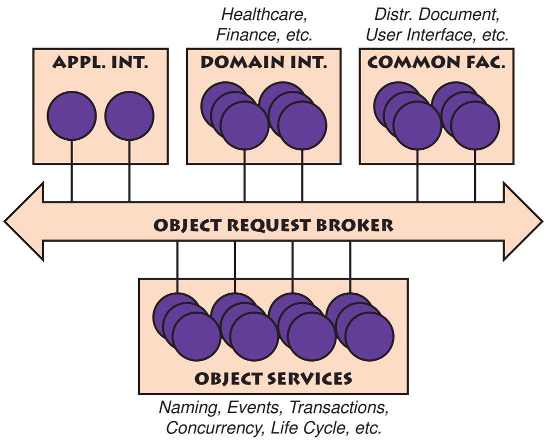

Figure 1 The OMA Architecture

Commercially known as CORBA, the *Object Request Broker* (ORB) is the communication heart of the OMA. It enables heterogeneous objects to transparently invoke remote operations and receive replies in a distributed environment. The ORB also provides the environment for managing objects, advertising their presence, and describing their meta-data. Each object interface is specified in the OMG *Interface Definition Language* (IDL), which is implementation independent. Clients use *object references* to identify remote objects and invoke their operations.

The *Object Services* are a collection of CORBA objects that support general-purpose functionalities. Services are not related to any specific application but are basic building blocks, usually provided by CORBA environments. Several services have been designed and standardized by the OMG.

The *Common Facilities* are a collection of interfaces and objects providing end-user-oriented capabilities useful across many application domains. The *Domain Interfaces* are meant to be used only in specific application domains. Finally, the *Application Interfaces* are interfaces specific to end-user applications.

2.2 Replication Techniques

Redundancy is a common practice for masking the failures of individual components of a system. With redundant copies, a replicated object can continue to provide a service in spite of the failure of some of its copies, without affecting its clients. In distributed systems, the two best known replication techniques are *active* and *passive* replication. Each of these techniques has its own advantages, and they are thus complementary. A brief description of both techniques is given below.

2.2.1 Active Replication. *Active replication* — also called the *state machine approach* — is a general protocol for replication management that has no centralized control [14]. All copies of the replicated object play the same role: they all receive each request, process it, update their state, and send a response back to the client (Figure 2). Since the invocations are always sent to every replica, the failure of one of them is transparent to the client.

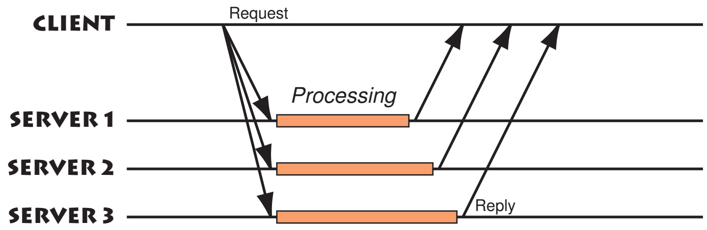

Figure 2 Active Replication

From the point of view of a client, all correct replicas should appear as having the same state. In order to guarantee this, all invocations sent by the clients should be treated in the same order by all correct replicas. This is ensured by a *total order multicast* primitive [8] — also called *atomic multicast* — that provides total ordering of messages sent to a set of destinations.

2.2.2 Passive Replication. With *passive replication* — also called *primary-backup replication* — one server is designated as the *primary*, while all other are *backups* [2]. The clients send their requests to the primary only. The primary executes the request, atomically updates the other copies, and sends the response to the client (Figure 3). If the primary fails, then one of the backups takes over.

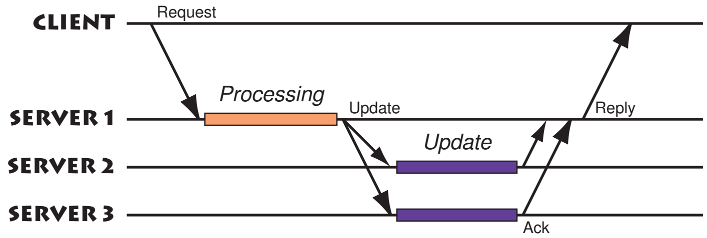

Figure 3 Primary-Backup Replication

2.2.3 Which Replication Technique is Better?. Active replication requires the operations on the replicated object to be *deterministic*. Determinism means that the outcome of an operation depends only on the initial state of the object, and on the sequence of operations performed by the object (history). If the operations on the replicated object are deterministic, the shared state of the replicated object remains consistent and all responses sent back to the client are identical. Thus, the client can simply wait for the first reply from any replica. An interesting property of active replication lies in the fact that a crash does not increase the latency experienced by a client.

Unlike active replication, passive replication does not waste extra resources through redundant processing, and permits *non-deterministic* operations. However, a crash of the primary may significantly increase the latency of an invocation. Furthermore, passive replication requires additional application support for the primary to update the state of the other copies.

There is a tradeoff between both replication techniques. Active replication provides a more predictable and generally faster response time than passive replication, and requires less application support. On the other hand, passive replication is more flexible since it does not require the servers to be deterministic. Since active and passive replication are complementary, *supporting both techniques* is a major benefit that permits the application developer to choose the technique best adapted to his problem.

2.3 Related Work

Several systems have been developed to support object replication based on group communication in a CORBA environment. Electra [10] and Orbix+Isis [9] use the Isis group communication toolkit [1] to implement both active and passive replication. The type of replication can only be specified statically (at compile-time) through object inheritance. It is not possible to dynamically specify the replication technique, nor to mix replication techniques for the same object. Eternal [12] uses the Totem group communication system [11] to provide object replication. Similarly to Electra and Orbix+Isis, Eternal supports active and passive replication at the object level, and not at the operation level.

3 SYSTEM SUPPORT FOR REPLICATED OBJECTS

This section presents the basic components that provide the system support for object replication. These components define replication protocols, as well as facilities for managing the structure, addressing, and invocation of replicated objects. They are used as a generic infrastructure for marrying active and passive replication in a unified framework. As depicted in Figure 4, the OGS replication infrastructure essentially consists of three basic components:

- The consensus service is the building block that provides the common algorithmic infrastructure for both replication techniques.

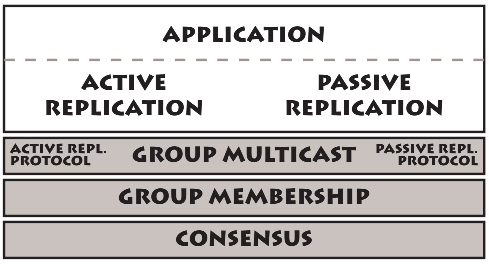

Figure 4 Components Architectural Overview

- The group membership service manages the changes in the composition of groups.

- The group multicast service provides the communication protocols required for active and passive replication. These protocols allow the client to issue requests to a group of actively or passively replicated objects.

These components are packaged as CORBA services, entirely defined in terms of IDL interfaces and usable between heterogeneous objects. They are not exclusive to object replication, and may be used in very different contexts. The OGS replication infrastructure is original in that active and passive replication invocation protocols are based on a common consensus algorithm, and are thus unified *at the protocol level*. The choice of the replication technique can be performed by the application developer on a per-operation basis.[1]. In other words, this choice can be driven by the operation semantics, depending for instance on whether the operation is deterministic or not. The components and protocols of the OGS replication infrastructure are described in the rest of this section.

3.1 Consensus: A Generic Building Block

Informally, a consensus allows several processing elements to reach a common decision despite the crash of some of them. The consensus problem is a central abstraction for solving various agreement problems (atomic commitment, total order, membership) [7], and thus for achieving fault-tolerance in distributed systems. Agreement problems are present in many distributed systems, such as systems based on virtual synchrony, transactions, or replication, but are generally implemented using ad hoc protocols.

Chandra and Toueg have proposed a consensus algorithm [3] that works in an asynchronous system augmented with an unreliable failure detector mechanism. This consensus algorithm allows us to easily implement active replication: the consensus

[1]Actually, the choice may even be made on a per-invocation basis.

algorithm is used by the replicated objects to agree on a set of messages to deliver and their respective ordering, thus ensuring that the copies of the replicated object remain consistent.

A passive replication algorithm based on consensus has been proposed in [4]. This algorithm is based on a variant of consensus, called $\mathcal{DIV}$consensus (Deferred Initial Values consensus). In contrast with the traditional consensus problem, $\mathcal{DIV}$consensus participants are not required to have an initial value when starting the consensus algorithm. Instead, the consensus algorithm asks the participant for its initial value when needed. This property makes it possible to also implement passive replication with an algorithm that solves consensus.

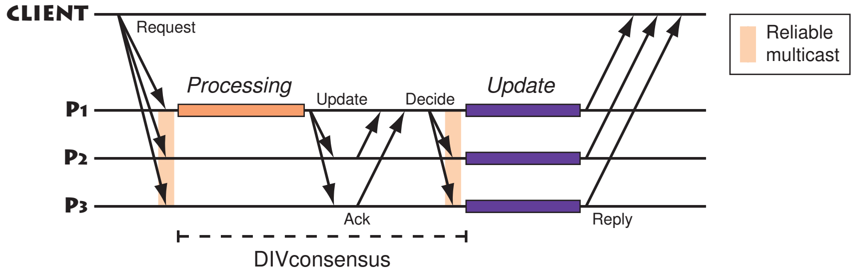

Figure 5 Passive Replication Based on $\mathcal{DIV}$consensus

Informally, the passive replication protocol works as follows: the $\mathcal{DIV}$consensus algorithm is based on a rotating coordinator paradigm, and proceeds in asynchronous rounds. In every round, a different process plays the role of the coordinator. This scheme makes it possible to use the current coordinator as the primary, and to select another primary when entering a new round (which occurs only if the current coordinator is suspected). In addition, the $\mathcal{DIV}$consensus algorithm takes care of delivering an update message to all non-coordinator participants, which play the role of backups in passive replication. This behavior is depicted in Figure 5, in which $p1$ is the primary and the coordinator of the $\mathcal{DIV}$consensus algorithm, while $p2$ and $p3$ are the backups. A more detailed description of the $\mathcal{DIV}$consensus algorithm and its application to passive replication can be found in [4].

We have used the $\mathcal{DIV}$consensus algorithm presented in [4] for providing a generic consensus service that lets a set of CORBA objects agree on application-specific data. Basing agreement protocols on a generic consensus service bears many advantages. It decouples the algorithmic aspects from the application-specific functionalities, and thus promotes the system's modularity and reusability. In addition, it enables efficient implementations of the protocols as well as precise characterization of their liveness [7]. The consensus is a well-known problem for which there exist proved algorithms; using these algorithms ensures that the safety of the system can not be violated.

3.2 Object Groups: Managing Replicated Objects

The key idea of *group communication* is to gather a set of processes or objects into a logical group, and to provide primitives for sending messages to all group members at the same time with various ordering guarantees. A group constitutes a logical addressing facility since messages can be issued to groups without having to know the number, identity, or location of individual members (Figure 6). Groups have proven to be very useful for providing *high availability* through *replication*: a set of replicas constitutes a group, viewed by clients as a single entity in the system.

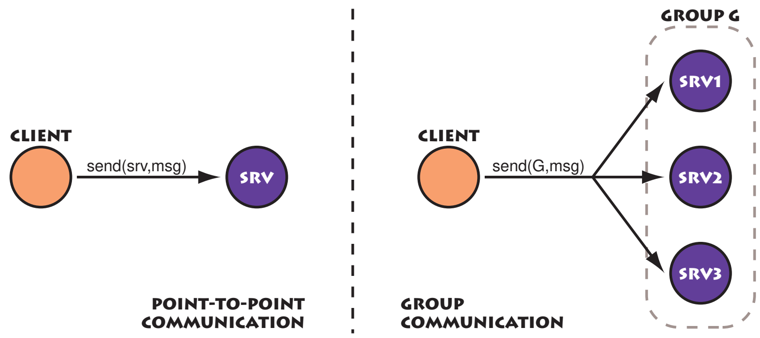

Figure 6 Point-to-Point vs. Group Communication

In order to provide group communication in a CORBA environment, we have specified and implemented a generic group service [5, **?**], which defines interfaces for gathering sets of CORBA objects into logical groups. The group service implements essentially two functionalities: *group membership* and *group multicast*.

Group membership manages the life cycle of object groups. It maintains the updated list of all correct group members and provides support for joining and leaving groups, and for view change notification. The *view change protocol* occurs each time the composition of a group changes, and ensures that every correct member of the group receives a *view change notification*, indicating the new composition of the group as a list of group members with mutually consistent rankings.

Group multicast provides primitives for sending invocations to groups instead of singleton objects. OGS provides a rich set of group multicast primitives, adapted to various types of applications. OGS implements groups as open structures, allowing non-member objects to issue multicast invocations to groups. Multicast primitives can be classified according to their degree of *reliability* and their *ordering* guarantees. A reliable multicast guarantees that all correct group members deliver the same set of messages *(agreement)*, that this set includes all messages multicast to the group by correct objects *(validity)*, and that no spurious messages are ever delivered *(integrity)* [8]. Reliable multicast, in itself, does not ensure that group consistency is preserved; it is generally combined with an ordering guarantee. As a matter of fact, the state of an object generally depends on the order in which it receives requests.

OGS provides several types of multicast primitives with various reliability and ordering guarantees. In particular, two protocols are essential for active and passive replication. The *active replication protocol* is based on a totally ordered multicast, which guarantees that reliable multicasts are delivered in the same order to all target objects. In addition, it ensures that the client receives a reply. The *passive replication protocol* guarantees that one copy (the primary) receives and handles the original invocation, that all other copies (the backups) receive an update message from the primary, and that the client receives a reply. In OGS, a protocol may be associated with any operation of a server's interface, without the knowledge of the client. This scheme provides more flexibility that in comparable systems since it allows the developer to choose a protocol that matches the operation's semantics.

OGS provides group transparency to clients through its typed invocation interface. Clients can issue invocations to an object group as if they were invoking a single object. The client directly invokes operations of the server's interface using static stubs, and OGS delivers multicasts by directly invoking the relevant operation of the server, using static skeletons. OGS transparently filters messages and returns a single reply to the client. This scheme is very useful when using groups for replication, since it permits to hide *replication* — and not only the *replication technique* — from the client. The application can invoke a single object or an object group without even a re-compilation of the client's code. In addition, the application developer does not need to perform the marshaling and unmarshaling of the request (these operations are performed transparently by OGS), and can benefit from the type safety of CORBA's static invocation interface.

4 MARRYING ACTIVE AND PASSIVE REPLICATION

This section presents the architectural framework of our CORBA replication service, and how its class hierarchy maps to the algorithmic infrastructure of Section 3. We describe how groups may be used for replication, and how we have abstracted the architectural particularities of active and passive replication, to ultimately marry them in a unified framework.

4.1 From Groups to Replication

The class hierarchy of our replication service is shown in Figure 7. The figure depicts the classes, their operations, and their associated protocols. The top-level `Groupable` interface is shared by all group members, independently of whether groups are used for replication or not. It essentially defines operations for view change notification (invoked each time a member joins or leaves a group), and protocols for group communication (such as reliable or totally ordered multicast).

As mentioned previously, object groups are considered an adequate paradigm for object replication: all the copies of a replicated object are gathered into a group, which is used as an addressing facility. An additional property of a replica lies in the fact that all copies *must* be kept identical. Therefore, a state transfer mechanism is necessary when a groupable object is used as a replica. Figure 7 illustrates this requirement by

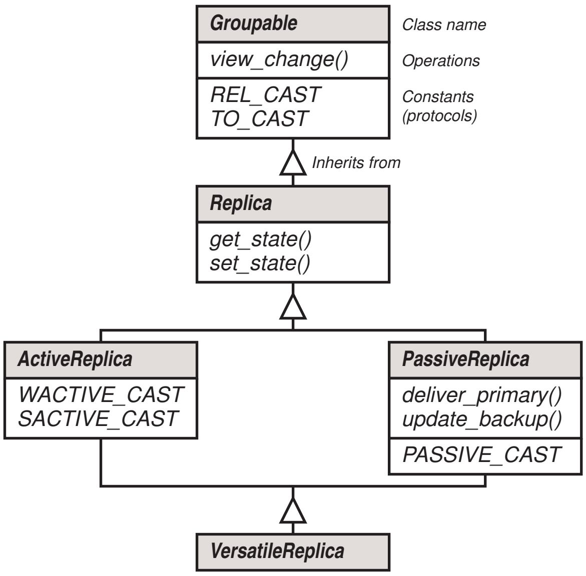

Figure 7 Interfaces of the Object Replication Service

defining an abstract `Replica` interface that inherits from `Groupable`, and that adds operations for transferring the state. These operations are asynchronously invoked by the service when a new member joins a group.[2]

4.2 Groups for Active Replication

Active replication is straightforward to implement using group membership and group multicast primitives. The active replication protocol consists of a totally ordered multicast primitive used for replica invocation, and a point-to-point communication primitive for sending the reply back to the issuer.

It is possible to weaken the active replication protocol for situations in which the replicated object is not modified by an invocation. Indeed, a totally ordered multicast may not be necessary for read-only operations, and the client may want to invoke the operation on the closest/fastest replica. Therefore, we introduce two active replication protocols: *strong* and *weak* active multicast. The former ensures total order while the latter does not. These protocols (`SACTIVE_CAST` and `WACTIVE_CAST`) appear in the `ActiveReplica` interface in Figure 7.

The active replication model is symmetrical: all group members have the same behavior, which is to accept a request, process it, and optionally return a reply. Repli-

[2]When CORBA implementations will provide facilities for passing *objects by value*, one could use these facilities for state transfer instead of the `get_state()` and `set_state()` operations.

cation is thus *transparent* to the code that implements application-specific operations: OGS directly invokes the target operation and request processing is performed the same way whether the object is replicated or not. Therefore, no extra operations are defined on the active replica interface.

4.3 Groups for Passive Replication

Unlike active replication, the passive replication distinguishes between a primary object which processes the requests, and the replicas which only receive updates from the primary. Therefore, the passive replication protocol must deliver the original message to the primary only, and send the updates to the backups. In addition, the protocol must be able to switch to another primary in case the current one fails. This behavior is implemented by the $\mathcal{DIV}$consensus algorithm, as described in Section 3.1. The resulting passive multicast protocol (`PASSIVE_CAST`) appears in the `PassiveReplica` interface in Figure 7.

At the interface level, passive replication requires more application support than active replication. The reply sent by a passively replicated object to a client cannot be used for updating the backups, since the reply does not generally contain the update information required by the backups (e.g., an operation may modify the server's state without returning a reply to the client). Since the part of the state that is modified by an operation is not known by the service, the application must provide explicit support for the state update. This is achieved by adding two operations to the `PassiveReplica` interface (Figure 7): `deliver_primary()` to deliver a request to a primary and obtain both a reply for the client and an update for the backups; and `update_backup()` to update the backups based on the information given by the primary. Note that the `deliver_primary()` operation must analyze the client's request and invoke the target operation of the server's interface, thus requiring extra work in the server's implementation. However, this code does not depend on the server's interface and can be written in a generic way.

OGS offers an alternate solution to the state update problem, that does not require implementing the state update operations. The application developer can request that OGS directly invokes the operations of the server's interface, and uses the state transfer mechanism to perform a *full* state update for each operation. If the state is very small (e.g., a single value), it is reasonable to use this mechanism to update the backups in a transparent way after processing each request.[3]

4.4 A Unified Replication Framework

As depicted in Figure 7, our replication framework defines a common unified `VersatileReplica` interface that inherits from both the active and passive replica interfaces. Application objects that support this interface can thus act as an active or

[3]Orbix+Isis [9] uses another approach, which requires modifying the implementations of the operations so that they explicitly check whether they are primary or backup and act accordingly. This approach requires additional code to be written for each operation, and it prevents to reuse the code of server's implementations.

passive replica. Since the replication protocols are declared higher in the hierarchy, a versatile replica can use any of these protocols. The resulting class hierarchy provides the architectural framework for marrying active and passive replication.

The protocol associated to each individual operation can be dynamically specified by a replicated object. In addition, the object can specify informations about operation properties (such as commutativeness), which can be used by the replication protocols for performing optimizations. For instance, two commutative requests do not need to be totally ordered with active replication. Another optimization consists in parallelizing request processing with passive replication by using different primaries for operations that modify non-overlapping parts of the replica's state.

An interesting property of OGS is that the client multicast protocol is identical for all replications techniques and multicast semantics. This property makes it possible to completely hide the replication policy from the client, and to dynamically modify the replication techniques on the server side.

5 CONCLUSION

In this paper, we have presented a replication framework that combines the best of both worlds: active replication for its ease of use and its more predictable response time, and passive replication for its efficient resource usage and its ability to work with non-deterministic objects.

On the protocol side, the marriage between active and passive replication is provided by our generic consensus service that acts as a common denominator between both replication techniques. On the architectural side, our unified replication framework defines a class hierarchy that makes this marriage possible. Active replication is essentially provided by composing group membership, state transfer, and an active replication protocol. Similarly, active replication is built using group membership, state transfer, state update, and a passive replication protocol.

Our architecture is unique in that it makes it possible to dynamically specify the replication technique to be used for any operation, while most existing systems enforce the replication technique to be specified statically and at the object level. We believe that, although we developed this framework in the context of CORBA, our architecture is valuable for various middleware environments.

References

[1] K.P. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):36–53, December 1993.

[2] N. Budhiraja, K. Marzullo, F. Schneider, and S. Toueg. *Distributed Systems*, chapter 8: The Primary-Backup Approach, pages 199–216. Addison-Wesley, 2nd edition, 1993.

[3] T.D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[4] X. Défago, A. Schiper, and N. Sergent. Semi-passive replication. In *Proceedings of the 17th Symposium on Reliable Distributed Systems (SRDS-17)*, West Lafayette, Indiana, USA, October 1998.

[5] P. Felber, B. Garbinato, and R. Guerraoui. The design of a CORBA group communication service. In *Proceedings of the 15th IEEE Symposium on Reliable Distributed Systems*, pages 150–159, October 1996.

[6] P. Felber, R. Guerraoui, and A. Schiper. The implementation of a CORBA object group service. *Theory and Practice of Object Systems*, 4(2):93–105, 1998.

[7] R. Guerraoui and A. Schiper. Consensus service: a modular approach for building agreement protocols in distributed systems. In *Proceedings of the 26th International Symposium on Fault-Tolerant Computing (FTCS-26)*, pages 168–177, June 1996.

[8] V. Hadzilacos and S. Toueg. *Distributed Systems*, chapter 5: Fault-Tolerant Broadcasts and Related Problems, pages 97–145. Addison-Wesley, 2nd edition, 1993.

[9] IONA and Isis. *An Introduction to Orbix+Isis*. IONA Technologies Ltd. and Isis Distributed Systems, Inc., 1994.

[10] S. Maffeis. *Run-Time Support for Object-Oriented Distributed Programming*. PhD thesis, University of Zurich, February 1995.

[11] L.E. Moser, P.M. Melliar-Smith, D.A. Agarwal, R.K. Budhia, and C.A. Lingley-Papadopoulos. Totem: A fault-tolerant multicast group communication system. *Communications of the ACM*, 39(4):54–63, April 1996.

[12] L.E. Moser, P.M. Melliar-Smith, and P. Narasimhan. Consistent object replication in the Eternal system. *Theory and Practice of Object Systems*, 4(2):81–92, 1998.

[13] OMG. *The Common Object Request Broker: Architecture and Specification*. OMG, February 1998.

[14] F. Schneider. *Distributed Systems*, chapter 7: Replication Management using the State-Machine Approach, pages 169–197. Addison-Wesley, 2nd edition, 1993.

Biography

Pascal Felber received the Ph.D. degree in Computer Science from the Swiss Federal Institute of Technology, Lausanne (EPFL), in 1998. His main research interests are in the area of object-based and dependable distributed systems. He is currently member of the Application Server Group of Oracle Corp. He can be reached at *pfelber@us.oracle.com*.

Xavier Défago graduated from the EPFL in 1995. After working for a year on parallel operating systems at NEC C&C Central Research Labs. in Kawazaki (Japan), he joins Prof. André Schiper's Operating Systems laboratory as a research assistant and a PhD student. His main research interests are in the areas of fault-tolerant distributed systems, distributed algorithms, operating systems principles, and real-time systems.

Patrick Eugster obtained his diploma in Computer Science from EPFL in 1998, after which he joined the Operating Systems laboratory of Prof. André Schiper. As a research assistant and PhD student, he currently works on a European research project (OpenDREAMS II). Among his interests are fault-tolerance, distributed systems, middleware and object oriented concepts.

André Schiper has been professor of Computer Science at EPFL since 1985, leading the Operating Systems laboratory. He has worked on the implementation of communication primitives for the ISIS system, developed at Cornell University. His research interests are in the areas of fault-tolerant distributed systems and group communication. In the context of CORBA, his group is currently working on the specification and the implementation of a replication service. André Schiper is a member of the ESPRIT Basic Research Network of Excellence in Distributed Computing Systems Architectures (CaberNet).