Building Dependable Distributed Objects with the AQuA Architecture[1]

Michel Cukier, Jennifer Ren, and Paul Rubel
Center for Reliable and High-Performance Computing
Coordinated Science Laboratory and
Electrical and Computer Engineering Department
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801, USA
{cukier, ren, rubel}@crhc.uiuc.edu

David E. Bakken and David A. Karr
BBN Technologies
Cambridge, Massachusetts 02138, USA
{dbakken, dkarr}@bbn.com

1. Introduction

Providing fault tolerance to distributed applications is an important problem. The flexibility that software can offer makes it a natural choice for implementing a significant portion of the fault tolerance of dependable distributed systems. Furthermore, when the dependability requirements change during the execution of an application, the fault tolerance approach must be *adaptive* in the sense that the mechanisms used to provide fault tolerance may change at runtime. The AQuA architecture [Cuk98, Cuk99] provides a flexible approach for building dependable, distributed, object-oriented systems that support adaptation needed due to both faults and changes in an application's dependability requirements.

The AQuA architecture is composed of several components: Quality Objects, Proteus, Maestro/Ensemble, and gateways. In order to provide a simple way for application objects to specify the level of dependability they desire, the AQuA architecture uses the *Quality Objects* (QuO) [Zin97, Loy98] framework to process and invoke dependability requests. QuO allows distributed applications to specify quality of service (QoS) requirements at the application level using the notion of a "contract," which specifies actions to be taken based on the state of the distributed system and desired application requirements.

Proteus [Sab99] provides fault tolerance in AQuA by dynamically managing replicated distributed objects to make them dependable. It does this by configuring the system in response to faults and changes in desired dependability levels. The choice of how to provide fault tolerance involves choosing the types of faults to tolerate (crash failures, value, or time faults), the styles of replication to use (active or passive), the types of voting to use ("pass first," "leader only," or majority), the degrees of replication to use, and the location of the replicas on the different hosts, among other factors.

Proteus uses *Maestro/Ensemble* [Hay98, Vay98] as the underlying group communication system that provides reliable multicast, total ordering, and virtual synchrony. Communication between all architecture components is done using *gateways*, which translate CORBA object invocations into messages that are transmitted via Ensemble, and contain mechanisms to implement a chosen fault tolerance scheme.

[1] This research has been supported by DARPA Contracts F30602-96-C-0315 and F30602-97-C-0276.

2. Proteus Overview

Proteus is a flexible infrastructure developed to implement adaptive fault tolerance in the AQuA architecture. The organization of Proteus is shown in Figure 1. The replicated *dependability manager* is composed of two parts: an *advisor*, which makes decisions regarding reconfiguration based on reported faults and dependability requests from QuO, and a *protocol coordinator,* which, together with the gateways, implements the chosen fault tolerance approach. Depending on the choices made by the advisor, Proteus can tolerate and recover from crash failures, time faults, and value faults in application objects and the QuO runtime. *Object factories* are used to kill and start replicated applications, depending on decisions made by the dependability manager, and to provide information regarding the host to the dependability manager. *Gateways* provide two functions. First, they provide a standard CORBA interface by translating between process-level communication, as supported by Ensemble, and IIOP messages, which are understood by Object Request Brokers (ORBs) in CORBA. In this way, CORBA-based distributed applications written for the AQuA architecture can use standard, commercially available ORBs. In addition, gateways provide fault tolerance using *voters* and *replication protocols*. These services are located in the *gateway handlers*. Both active and passive replication of *AQuA objects* can be supported. AQuA objects are the basic units of replication in the AQuA architecture. Each one consists of a gateway, an application object, and a QuO runtime, if QuO is being used. In this context, the *application object* can be part of the distributed application itself, or part of the AQuA architecture that uses the services of a gateway. Several voting policies can be used: "pass first" (the first message received by the leader is sent out), "leader only" (only the leader sends out its message), or a classical majority vote. A more detailed presentation of Proteus can be found in [Sab99].

3. Group Structure

An important contribution of the AQuA architecture is the use of a set of interacting small groups to ensure reliable communication between different AQuA components. This group structure avoids the problem of scalability typically associated with the use of group communication and allows independent communication between groups. Four types of groups are used in AQuA: "replication groups," "connection groups," the "Proteus Communication Service (PCS) group," and "point-to-point groups." A *replication group* is composed of one or more replicas of an AQuA object. A replication group has one object that is designated as its leader, and may perform special functions. Each object in the group has the capacity to become the object group leader. A *connection group* is a group consisting of the members of two replication groups that wish to communicate. A message is multicast within a connection group in order to send a message from one replication group to another one. A connection group can define different communication schemes.

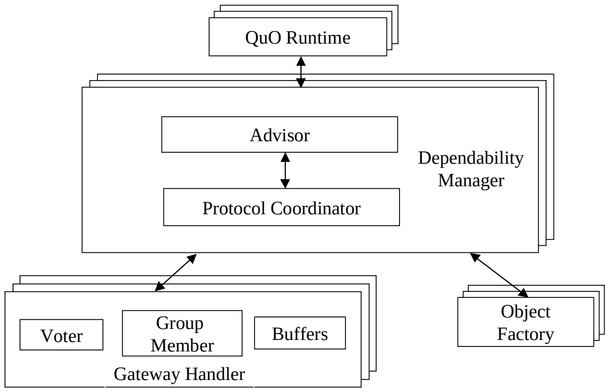

Figure 1: Proteus Architecture

Reliable multicast to the dependability manager is achieved using the *Proteus Communication Service (PCS) group.* The PCS group consists of all the dependability manager replicas, if it is replicated. The PCS group also has transient members. These transient members are object factories, AQuA applications, or QuO objects that want to communicate with the dependability manager. Through the PCS group, AQuA application objects provide notification of view changes, QuO objects make requests for QoS, and object factories respond to start and kill commands and provide host information updates. Finally, *point-to-point groups* are used to send messages from a dependability manager to an object factory. A communication scheme for tolerating crash failures that is based on the four types of groups and uses active replication and a "pass first" voting policy is described in [Sab99].

4. Interface to the Dependability Manager

The interface to the manager can be divided into two sets of methods: those used to communicate with the components in the AQuA system core composed of the dependability manager, the gateways, and the object factories, and those used by one or more objects to request and observe QoS, observe the state of the dependability manager, and observe and control the state of hosts. We now describe those methods (in the second set) available to an application or a QuO programmer for interaction with the dependability manager.

In particular, Proteus supports the development of objects that can make QoS requests from the dependability manager and also observe its actions. One of these object types, called the *QoS Observer/Requester,* can be used to make QoS requests to the dependability manager and can receive callbacks regarding the ability of the manager to satisfy the requester's requests. An example of an application that may contain a QoS observer/requester is QuO itself. Furthermore, since the dependability manager supports a standard, well-defined interface, an application object can also make QoS requests directly to the dependability manager.

The second type of objects the dependability manager supports is that of *Advisor Observers.* Advisor observers can "subscribe" to a variety of information used by the manager to make decisions, including information about faults detected and fine-grain information regarding actions taken by the manager.

Proteus also supports the development of a third type of objects, called *Host Observers/Controllers,* which receive information regarding the status of hosts that may be used to execute object replicas, and that can be used to make requests regarding which hosts can be used to execute replicas. Host observers/controllers can be used by an application or QuO to specify hosts that should not be used to execute replicas, if the application or QuO has information that leads it to believe that the replica should not be used. A more detailed description of these objects can be found in [Cuk99].

5. For More Information

More information on the AQuA project can be found at http://www.crhc.uiuc.edu/PERFORM.

Acknowledgments

We would like to acknowledge several other members of the AQuA and QuO teams, namely Mark Berman, Joe Loyall, Partha Pal, Chetan Sabnis, William H. Sanders, Rick Schantz, and John Zinky for many constructive discussions.

References

[Cuk98] M. Cukier, J. Ren, C. Sabnis, D. Henke, J. Pistole, W. H. Sanders, D. E. Bakken, M. E. Berman, D. A. Karr, R. E. Schantz, "AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects," Proc. of the 17th IEEE Symposium on Reliable Distributed Systems, pp. 245-253, West Lafayette, IN, USA, October 1998.
[Cuk99] M. Cukier, J. Ren, P. Rubel, W. H. Sanders, D. E. Bakken, D. A. Karr, "Building Dependable Distributed Applications using AQuA," submitted for publication.
[Hay98] M. G. Hayden, "The Ensemble System," Ph.D. thesis, Cornell University, 1998.
[Loy98] J. P. Loyall, R. E. Schantz, J. A. Zinky, D. E. Bakken, "Specifying and Measuring Quality of Service in Distributed Object Systems," Proc. of the First International Symposium on Object-oriented Real-time Distributed Computing (ISORC '98), pp. 43-52, Kyoto, Japan, April 1998.
[Sab99] C. Sabnis, M. Cukier, J. Ren, P. Rubel, W. H. Sanders, D. E. Bakken, D. A. Karr, "Proteus: A Flexible Infrastructure to Implement Adaptive Fault Tolerance in AQuA," Proc. 7th IFIP Working Conf. on Dependable Computing for Critical Applications (DCCA-7), pp. 137-156, San Jose, CA, USA, January 1999.
[Vay98] A. Vaysburd, K. P. Birman, "The Maestro Approach to Building Reliable Interoperable Distributed Applications with Multiple Execution Styles," *Theory and Practice of Object Systems*, vol. 4, no. 2, pp. 73-80, 1998.
[Zin97] J. A. Zinky , D. E. Bakken, R. E. Schantz, "Architectural Support for Quality of Service for CORBA Objects," Theory and Practice of Object Systems, vol. 3, no. 1, pp. 55-73, April 1997.