Building Reliable Adaptive Distributed Objects with the Maestro Tools *

Alexey Vaysburd Ken Birman
Department of Computer Science
Cornell University

{alexey,ken}@cs.cornell.edu

July 3, 1997

Abstract

This paper presents the Maestro Tools, a distributed object layer built on top of the Ensemble group communication system developed at Cornell. The Maestro tools include a visual application development environment; an interface to and implementation of fundamental distributed object abstractions, such as CSCW (cooperative-work), client/server, and publish/subscribe objects; and a set of adaptive control/policy modules which are used to adjust Ensemble failure detection and group multicast protocols in accordance with the application's quality of service requirements. The Maestro tools have been used in a reliable CORBA system and in a number of other projects. Results reported in this paper are work in progress: about 2/3 of our goals have been achieved, but there is still work to be done before the full-scale object layer is completed.

1 Introduction

The rapid convergence of reliable distributed computing and object-oriented technologies is a reality of modern network computing. This convergence responds to many of the challenges confronting developers of distributed systems at all layers, from low-level network protocols up to the application. Complexity of development and management of reliable distributed systems tends to become overwhelming, even more so as the systems are deployed on a large scale over wide-area environments. The use of object-oriented paradigms hides system implementation details and makes integration easier when different system layers must interact within a distributed application. This provides the motivation for the Maestro tools.

*This research was supported by ARPA/ONR grant N00014-96-1-1014

Maestro is a distributed object layer built over the Ensemble group communication system [4] developed at Cornell University. It implements fundamental distributed-communication abstractions, including CSCW (cooperative-work), client/server, and publish/subscribe objects. These can be used directly as basic building blocks of distributed applications, or integrated within still higher-level distributed object technologies such as CORBA [5].

One of the goals of Maestro is to support smooth integration of object layers above it, and communication protocols below, as concerns both functional interfaces and system properties (quality of service). Maestro addresses the latter issue by providing a set of policy modules that control adaptivity mechanisms of underlying Ensemble protocols, in accordance with quality of service requirements specified by layers above Maestro. The approach taken in Maestro follows the general *multi-level masking adaptivity model*. Within this model, a system layer compensates for configuration/environment changes in the layers below so as to avoid having to reconfigure itself and report a configuration change to the layers above [8].

Finally, Maestro addresses the issue of system development methodology by providing a visual environment for rapid prototyping/development of distributed object-oriented applications. The environment, still under construction, is called Visual Maestro, and appears as a "wizard" (in terminology of Windows NT) that guides the developer through system design and configuration choices that need to be made in the process of developing a distributed application using Maestro/Ensemble. The wizard generates Maestro code ready to be compiled and linked within the application. The use of the "wizard" approach in Maestro should substantially simplify and speed up the development process for distributed applications.

Maestro tools have been used by the Electra system (a reliable CORBA implementation) [5], in the Quality of Service for CORBA Objects (QuO) project at BBN [8], and in a number of other projects. The Maestro API is the primary programming interface to Ensemble and Horus. The software is available as a part of the Ensemble distribution and can be downloaded from http://www.cs.cornell.edu/Info/Projects/Ensemble.

The rest of this paper is organized as follows. We discuss basic abstractions of the Maestro object layer in section 2. The Visual Maestro environment is described in section 3. Adaptive mechanisms and control policies of Maestro are discussed in section 4. We conclude with a discussion of further directions for use of Maestro in section 5.

2 Maestro Classes: Fundamental Abstractions

The distributed-object abstractions of Maestro are built upon protocol layers implemented in Ensemble [4]. Ensemble is the third-generation group communication system developed at Cornell as a successor to Isis and Horus [2, 7]. Ensemble allows to create communication entities called *endpoints* (a generalization of UNIX sockets), which can join *endpoint groups* and send multicast and point-to-point messages to each other. The protocols of Ensemble implement logical properties of group communication, such as *virtual synchrony* and *total ordering* [2].

Maestro captures fundamental distributed-object abstractions of Ensemble in a hierarchy of C++ classes. At the bottom of the hierarchy is the `GroupMember` class which implements basic group communication functionality, including methods for joining and leaving groups and sending messages. When an Ensemble event such as a group membership change or message delivery occurs, Maestro invokes the appropriate callback method of the corresponding `GroupMember` object. In the implementation of the `GroupMember` class, the callbacks are defined as functions with empty bodies. However, the default no-op callback behavior can be overloaded in application-specific subclasses of `GroupMember`. Note that system developers only have to write code for callbacks if their functionality needs to be modified. This effectively eliminates the ballast code in Maestro applications and speeds up the development process.

The `GroupMember` class implements most general semantics of group members and is a base class in the Maestro hierarchy. More specialized distributed-object abstractions, such as clients/servers and publishers/subscribers, are derived from `GroupMember`. For example, the `Server` class implements a *state transfer protocol* which is used to synchronize local states of server objects when new servers join the group or when several partitions of the same group (with potentially incoherent states) merge together. Group partitioning due to (transient) link failures is particularly common in distributed applications deployed over WAN's, and may potentially result in the corruption of the "global state" of the application. Maestro ensures consistency of the global state by integrating state transfer with the application layer above, and with replication protocols implemented by the communication system underneath [3].

3 Visual Maestro: Rapid Application Development

Visual Maestro is a "wizard"-style tool which aids in development of distributed object-oriented applications by guiding the programmer through a series of design choices which pertain to such dimensions as execution style, communication properties, quality of service requirements, and object state of the application. The programmer is also able to fill in the code for Maestro callback methods from within the wizard. Remember that callbacks are invoked when corresponding events are received by the object, which included such events as multicast or point-to-point message delivery, group membership changes, state requests, and others. The full set of callbacks that can be filled-in by the developer to implement application-specific functionality varies depending on the Maestro class to be used with the application. The choice of the specific class is made by Visual Maestro based on the execution style selected by the developer. Among possible options are CSCW (cooperative-work), client/server, and publisher/subscriber execution styles.

Visual Maestro allows the programmer to choose their consistency requirements for message delivery (available options include virtual synchrony, active replication, etc.) and global safety (whether or not support for primary partitions is required, and how the progress of non-primary partitions should be restricted). Another set of options concerns quality of service requirements of the application. The QoS parameters determine how Maestro policy controls will adapt the behavior of Ensemble protocol layers at the run time. Among possible QoS

requirements that can be specified by the developer are real-time guarantees on completeness of crash detection, accuracy requirements for failure detection, real-time liveness requirements, performability requirements, etc.

Based on choices made by the developer, Visual Maestro will generate the template code, which can be either compiled directly into the application, or be further modified by the developer as necessary. It is envisioned that for complex distributed applications off-line editing will be necessary, after the first step in the development process (generation of the template code with Visual Maestro) has been completed. However, the editing will mostly consist in connecting Maestro objects to other components of the application, without need to do manual configuration of Maestro itself.

4 Maestro Controls: Adaptive Protocol Policies

A non-trivial distributed application must be able to adapt to changes in the environment in order to perform satisfactorily. Adaptation is especially important over WAN's, where system conditions can fluctuate significantly, and transient link failures and partitions are commonplace. Ensemble provides a number of adaptive *mechanisms* which include the ability to change group membership configuration and switch protocol stacks on the fly [1, 6]. Maestro, in its turn, includes a set of *policies* which can be used to control Ensemble's configuration and adapt the behavior of protocol layers (specifically the layers that implement failure detection and reliable group multicast) at the run time.

Adaptation decisions are based on the environment status information available to Maestro and the quality-of-service requirements specified by the application. The aim of Maestro control policies is to maximize the performance of the application within the specified quality of service regions, and to compensate for adverse system conditions, following the masking adaptivity model of [8].

For example, Maestro may control message relaying and retransmission policies of Ensemble reliable group multicast protocols. Adaptation decisions will be made based on available point-to-point link stability information. By rerouting messages between member objects so as to bypass failed links, Maestro can compensate for partial disconnectivity of the group and thus avoid group-membership or quality-of-service reconfigurations of higher layers.

In another example, Maestro may decide to remove a member object from the group even if the object is available (in terms of link connectivity), provided it fails to comply with a group protocol in a way that may lead to a violation of the application's QoS requirements. For example, if a member object maliciously does not acknowledge messages, the entire group may be eventually blocked forever (once the sending windows become full and do not shift). Assuming the application has specified real-time liveness requirements, Maestro will intervene at some point (*before* the requirements have been violated) so as to remove the faulty member object from the group and thereby ensure the application can make progress.

5 Further Directions

The Maestro tools can be used directly, as an API, to develop reliable object-oriented distributed applications. Alternatively, Maestro can be encapsulated within another object-oriented layer that provides yet higher level of abstraction. For example, Maestro tools have been used in the Electra system [5], a CORBA Object Request Broker implementation. Electra is system-independent and can run over different transport layers. By running over Maestro/Ensemble, Electra takes advantage their group communication protocols and state transfer tools to increase reliability and availability of CORBA applications.

Maestro is also used (together with Electra) in the ongoing Quality of Service for CORBA Objects (QuO) project at BBN [8]. The goal of QuO is to extend CORBA by enriching IDL specifications with "system properties", specifically those pertaining to the application's quality of service (QoS) requirements. The adaptive control/policy tools of Maestro will play an important role in this context, by integrating QoS requirements of the QuO-enriched CORBA layer (above Maestro) with adaptive mechanisms and protocols of the Ensemble system underneath.

References

[1] K. Birman, R. Friedman, and M. Hayden. The Maestro Group Manager: A Structuring Tool For Applications With Multiple Quality of Service Requirements. Technical Report TR97-1619, Department of Computer Science, Cornell University, 1997.

[2] K. P. Birman. *Building Secure and Reliable Network Applications.* Manning Publishing Company and Prentice Hall, December 1996.

[3] R. Friedman and A. Vaysburd. Fast Replicated State Machines Over Partitionable Networks. In *Proc. of the IEEE 16th International Symposium on Reliable Distributed Systems*, Durham, NC, October 1997. To appear.

[4] Mark Hayden. *The Ensemble System.* PhD thesis, Department of Computer Science, Cornell University, Forthcoming, Dec. 1997.

[5] Silvano Maffeis. Adding Group Communication and Fault-Tolerance to CORBA. In *Proc. of the 1995 USENIX Conference on Object-Oriented Technologies*, Monterey, CA, June 1995.

[6] R. van Renesse, K. Birman, M. Hayden, A. Vaysburd, and D. Karr. Building Adaptive Systems Using Ensemble. Submitted for publication, June 1997.

[7] R. van Renesse, K. Birman, and S. Maffeis. Horus: A flexible Group Communication System. *Communications of the ACM*, 39(4):76–83, April 1996.

[8] J. Zinky, D. Bakken, and R. Schantz. Architectural Support for Quality of Service for CORBA Objects. *Theory and Practice of Object Systems*, April 1997.