Experiences with an Object Oriented Framework for Distributed Control Applications *

Raja Harinath† *University of Minnesota*
Jaideep Srivastava *University of Minnesota*
Jim Richardson *Honeywell Technology Center*
Mark Foresti *Air Force Research Laboratory*
{harinath,srivasta}@cs.umn.edu, richardson_jim@htc.honeywell.com, forestim@rl.af.mil

This paper describes our experiences with the design and implementation an object-oriented framework for distributed control applications. The salient features of our framework are briefly explained. Next, the paper explores the challenges faced in integrating commercial off-the-shelf (COTS) object-oriented products into the framework, and how it affected the design and implementation.

1 INTRODUCTION

In this paper we describe an object-oriented framework for distributed control applications. The control applications fall into two distinct categories, namely the command-and-control applications found in the military [Huang *et al*. 1996], and the process control applications found in industrial automation [Huang *et al*. 1996, Agrawal *et al*. 1996]. This framework is part of a distributed multimedia infrastructure being built by the University of Minnesota and Honeywell, Inc., in collaboration with the U.S. Air Force.

Figure 1 shows an example distributed control application. In this case, the map server, aircraft tracking, and target information are dynamic sources of information. Different parts of the organization are interested in different subsets of the aggregate information. For example, the strategic command is interested in target reconnaissance videos, and various views of the targets to help plan missions. The tactical command is interested in the status of current missions, and their feasibility, both in terms of resources, and timeliness. Having up-to-date information is crucial for good decision making in either scenario.

As this example illustrates, this framework has some special needs. First, changes in the situation being monitored must be propagated to the end user/application with appropriate timeliness and information quality guarantees. Second, a wide range of data types, including continuous media like audio and video, and others like text, images and records, must be managed. Finally, there is the requirement to use commercial object-oriented technologies as much as possible.

The remainder of this paper is organized as follows: In Section 2 we describe *active views*, a new framework for distributed control applications. In Section 3 we discuss the challenges faced in integrating commercial off-the-

*This work is supported by Air Force contract number F30602-96-C-0130 to Honeywell Inc., and via subcontract number B09030541/AF to the University of Minnesota.
† Corresponding author

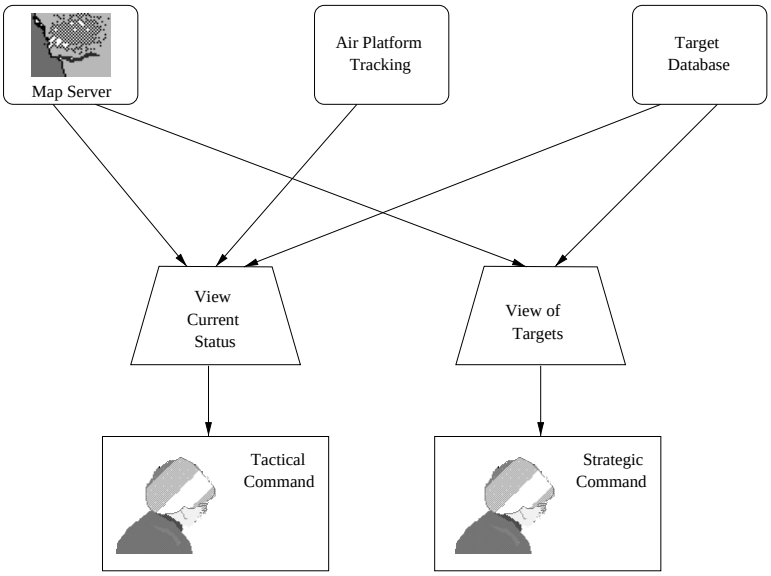

Figure 1: Example distributed control application

shelf products into our framework. In Section 4 we discuss some of the other technical issues we faced, and how they were addressed. In Section 5 we describe the experiences we've had so far.

2 ACTIVE VIEWS: A FRAMEWORK FOR DISTRIBUTED CONTROL APPLICATIONS

The principal goal of the framework was to develop a unified model to handle *periodic* (audio/video) and *aperiodic* (control information, updates to a database) streams of data. A secondary goal was to allow the user to use the framework as a black-box [Fayad and Schmidt 1997], with the ability to compose existing views in simple meta-programs, similar to the block-based programming metaphor [Huang *et al.* 1996]; at the same time providing a more sophisticated user with a useful programming environment.

This framework is called *active views*, where a user specifies his needs using a declarative view definition language. The view is *active* in the sense that updates to the actual situation (base objects) are propagated to the view with the specified qualitative requirements, commonly known as quality of service (QoS) parameters.

The model We consider a set of source objects, and a set of view objects whose state is a function of the states of some of the source objects. An example of such a function is database selection, clipping a video stream, or synchronizing an audio and video stream. Even the display window can be modeled as a view, the function being rendering of information on the screen.

We model streams as a sequence of state update operations. For example, for a database objects, the interface consists of `insert`, `delete`, and `modify` operations. For an MPEG stream, the operations could be `appendIFrame`, `appendPFrame`, and `appendBFrame`, that map directly to the frame types on the data stream. This allows us to uniformly handle the composition of objects that act upon the streams, be they periodic or aperiodic.

The framework The framework consists of a set of interface roles that allow the composition of objects through their interfaces, the Active View services, and a library of view objects.

Each role defines a set of interface requirements, and thus there exists a family of interfaces belonging to each role. The interface roles are:

- *Passive* – this role does not specify any interface requirements.
- *StateCopier* – an object exporting an interface in this role can be queried for its state information, and it responds with a set of operations that can be used to duplicate its state. For example, a database would emit a stream of inserts, one for each record. An MPEG stream would just emit the current image as a *I* frame.
- *Notifier* – this role is derived from the StateCopier role, and requires a Subscriber/Publisher interface. When an object subscribes to a Notifier, the StateCopier role is implicitly assumed by the Notifier. In addition, any later changes to the state of the Notifier object are passed on to all objects subscribing to it. Thus, in essence the Notifier provides an incremental StateCopier.
- *HistoryNotifier* – this role encapsulates the idea of a log, or state history. We can use this to query for an earlier state of an object. It can also be used to model an MPEG movie, for instance – the movie file could be modeled as a log of `append[IPB]Frame` operations.

Individual active view objects export one or more of these interfaces. An object with an Passive interface can be used as a target to a StateCopier of the same interface, or can subscribe to a Notifier of the same interface. An interface in the Passive role can be thought of as an "input" to the active view, and an interface with one of the other roles can be thought of as an "output" of the active view – the notions of "input" and "output" need not, but generally do, coincide with the direction of data flow between the objects.

An example interface would be a *Set*. An object with a *Set Passive* interface can handle `insert`, `delete`, and `modify` operations. It could be a database table, a database view, or a display. For example a map would export a Set interface, and would show a new marker on an `insert`, remove the marker on a `delete`, and move the marker on a `modify` operation. An object with a *Set Notifier* interface can essentially generate the Set operations – e.g., the source objects, or a filter (or view) object which can export both a Passive and a Notifier role.

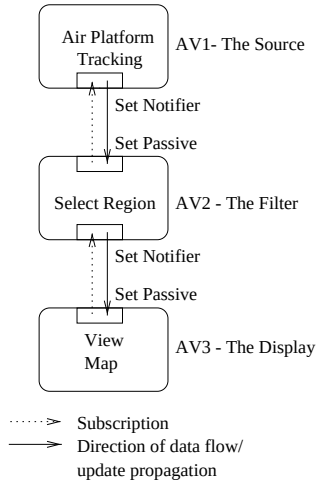

Figure 2: Example Active View Program

Example An example application is illustrated in Figure 2. The application is to show a map annotated with moving icons representing all the planes flying in a geographic region. Each of the objects *AV1*, *AV2* and *AV3* are active view objects. The AV1 object is a source, and exports a Set Notifier interface. AV2 exports both a Set Passive and a Set Notifier interface. The Set Passive interface of AV2 subscribes to the Set Notifier of AV1. Similarly, the Set Passive interface of AV3 subscribes to the Set Notifier of AV2. AV1 emits all changes to aircraft positions on its Notifier interface. AV2 filters those update notifications allowing only those that occur in the region of interest to AV3, the display. Each of these objects could be composed of other objects in a similar way. For example, AV3 could be composed of a map database and a display window that overlays a map with a set of markers.

3 USING COMMERCIAL OFF-THE SHELF (COTS) PRODUCTS

Due to many reasons, including development cost and standards in the application domain, we decided to use commercial off-the-shelf products for distributed object services, and for persistent object management. Choosing out of the many that fit our particular needs was quite important.

The COTS products that we picked were Iona's Orbix and Object Design's ObjectStore. The application domain standards mandated the use of CORBA [OMG 95]. We chose Orbix in view of its dominant presence in the Unix environment. The selection of an object database was much more difficult – since there are a plethora of products, but no standard. We chose ObjectStore, based on a survey of OODBMSes [Pazandak and Srivastava 1997].

The choice of these products strongly affected the design and implementation of our system.

Object Systems Interoperability We were faced with the problem of handling three different object systems, i.e., the one that comes with the programming language (C++), the distributed object management system (CORBA/Orbix), and the object database system (ObjectStore). There is enough difference in the three approaches and their abilities that we had to consider interoperability issues. For example, CORBA and ObjectStore have different object granularities – we could model an ObjectStore collection as a CORBA object, but the individual objects that comprise such a collection couldn't be easily fitted into the CORBA model.

The CORBA model defines an object by its interface, while the ObjectStore model is tied to the the implementation of an object. This *tension* both helped and hampered us. It helped us in that the two systems affected different parts of the design, and changes made for one did not affect the other too much. It hampered us by increasing the number of variables to deal with in the overall design and implementation.

Distribution A related issue was that CORBA and ObjectStore have different models of distribution – this strongly affected our model of distribution.

CORBA was mainly useful in making our framework support distributed applications. We discovered that, since our framework imposed certain requirements on the style of code written, making it distributed was relatively easy. The transition to using CORBA was reasonably painless, once we understood the conceptual differences in the object models.

Database Design ObjectStore's implementation exposed *reference* semantics, which are not naturally modeled by conventional database views. We thus had to make some basic changes to our model to accommodate this.

4 OTHER ISSUES

Some of the other issues that we considered in designing our framework are:

Choice of an object-oriented language The choices were C++ and Java. C++ has more mature off-the-shelf products, but Java claims to be the language of choice for distributed objects.

We felt that C++ should be the language used, since most of the COTS products for Java weren't mature enough at the time.

We used C++ templates extensively to enforce interfaces, without being too dependent on a class hierarchy to provide it. This was of great help when we had to modify our class hierarchy to accommodate Orbix and ObjectStore. Also, the concept of template *traits* helped us hide the differences in accessing persistent and non-persistent objects, and differences in accessing local and remote objects.

Real-time and QoS We envisioned the application being used interactively, with the presentation of multiple related data-types at the same time. For example, one window would show a map, another would show a video of the same location, and another would lists the resources in that area.

In enforcing real-time and QoS requirements [Wijesekara and Srivsatava 1996], we were strongly limited, since we had no good model of

the behavior of the COTS products that we used. Due to this, we have not yet addressed these issues in our implementation.

5 CONCLUSION

From conception to design to implementation, we learned and re-learned many of the lessons in [Schmidt and Fayad 1997]. We found that just deciding to use COTS technologies wouldn't immediately solve all our problems in the areas they address. This framework benefited strongly from being incrementally designed from earlier frameworks [Huang *et al*. 1996, Agrawal *et al*. 1996].

In conclusion, we believe that the use of COTS technology has added value to our project, and the benefits of using them outweigh the costs. The support for persistence and the support for distribution have allowed us to enhance and enrich our framework.

REFERENCES

AGRAWAL, M., KENCHAMMANA-HOSEKOTE, D. R., PAWAN, A., BHATTACHARYA, S., AND VAIDYANATHAN, N. High Performance Network Services for Multimedia-Integrated Distributed Control. Technical report, Honeywell Technology Center, Minneapolis, MN (July 1996).

FAYAD, M., AND SCHMIDT, D. Object-Oriented Application Frameworks. *Communications of the ACM*, 40, 10 (October 1997) 32–38.

HUANG, J., KENCHAMMANA-HOSEKOTE, D. R., RICHARDSON, J., SRIVASTAVA, J., FORESTI, M. P*resto*: A Multimedia Data Management System for Mission-Critical Applications. In *Proceedings of the 6th IEEE Dual-Use Technologies and Applications*, Utica, New York, IEEE (June 1996) 103–108.

OBJECT MANAGEMENT GROUP. *The Common Object Request Broker: Architecture and Specification.* Revision 2.0 (July 1995).

PAZANDAK, P., AND SRIVASTAVA, J. Evaluating Object DBMSs for Multimedia. *IEEE Multimedia*, 4, 3 (July-September 1997) 34–49.

SCHMIDT, D., AND FAYAD, M. Lessons Learned: Building Reusable OO Frameworks for Distributed Software. *Communications of the ACM*, 40, 10 (October 1997) 85–87.

WIJESEKARA, D. AND SRIVASTAVA, J. Quality of Service Metrics for Continuous Media. *Multimedia Tools and Applications*, 3 (Sep. 1996) 127–166.