Building Adaptive and Agile Applications Using Intrusion Detection and Response *

Joseph P. Loyall, Partha P. Pal, Richard E. Schantz
BBN Technologies
10 Moulton Street
Cambridge, MA 02138
{jloyall, ppal, rschantz} @bbn.com

Franklin Webber
410 West Green Street, #1
Ithaca, NY
Franklin.Webber@computer.org

Abstract

Traditional Intrusion Detection Systems (IDSs) mostly work off-line, without any direct runtime interaction or coordination with the applications (and with other IDSs) that they aim to protect. Including intrusion detection and response in the repertoire of an adaptive application extends its range of adaptivity and increases its chances for survival. In this paper we show how intrusion detection and response can be used to build agile, intrusion-aware applications under the Quality Objects (QuO) adaptive distributed middleware framework.

1. Introduction

Most current intrusion detection research focuses on detecting and recovering from intrusions on hosts or networks, rather than survivability of the applications running on them. There has recently been effort to enable intrusion detection systems (IDSs) to interoperate [23], but for the most part, current IDSs work in isolation from other IDSs, the applications that they are protecting, and the security managers whose policies they can influence.

We have developed a framework, Quality Objects (QuO), for building applications that are aware of their environment and can adapt to changes in it. QuO applications can specify their non-functional requirements (e.g., security, performance, or dependability requirements), measure what is being provided, access interfaces for controlling the desired level of service, and adapt to changes in levels of service. While this research was originally performed in the areas of network quality of service and open implementation, we have also been applying it to the areas of survivable applications and security.

Using the QuO framework, we support the following desirable system level behaviors to improve the survivability of applications:

- *The development of intrusion- and security-aware applications*. These applications can aid IDSs and security managers, by recognizing application-level patterns of usage that might indicate intrusions or security breaches. QuO includes support for inserting probes throughout an application's implementation for measuring the level of service provided. These probes can also gather information useful to IDSs and security systems, both for recognizing intrusions and for gathering information about their causes and sources.

- *The development of survivable applications*. These applications can adapt to changing conditions in their environment, including reported intrusions and changes in security policies. This enables them to avoid potential intrusions, continue in the face of degraded service, and recover from intrusions and faults.

- *Integration and interfacing of multiple IDSs at the application level.* While many IDSs are good at detecting certain types of intrusions, a 1998 DARPA ISO evaluation showed that multiple IDSs cover a larger space of potential intrusions [5]. However, most IDSs are not designed to work in conjunction with others. An application built within the QuO framework can interface to multiple mechanisms and managers, including multiple IDSs. QuO provides a capability, called system condition objects, for providing a common interface to mechanisms and managers that have proprietary interfaces. In this manner, QuO applications can access information from multiple IDSs that detect different types of intrusions. Let us emphasize that *the idea of in-*

*This work is sponsored by DARPA under contracts no. F30602-97-C-0276 and F30602-98-C-0187.

terfacing uniformly with multiple IDSs at the application level is not to come up with a better IDS, rather to increase the coverage and security of the application. This is complementary to the Common Intrusion Detection Framework (CIDF) effort [23], which is developing a framework for IDS-to-IDS communication with an aim to perfect the art of intrusion detection. CIDF does not provide any support for application-IDS cooperation.

- Integration of IDSs and other resource managers. IDSs and other managers, such as security policy managers or dependability managers, perform complementary activities and could cooperate to provide higher levels of service. For example, a security policy manager could use intrusion detection information provided by an IDS to dynamically determine whether to move to a stricter level of access control. Likewise, an IDS could use information from a fault detection or dependability manager to determine where and what type of intrusions to look for. QuO provides support for building applications that access managers in many different complementary dimensions (e.g., security, intrusion detection, and dependability) to achieve higher levels of service and adaptability.

This paper describes the QuO framework and our initial experiments in building adaptive, agile, survivable applications using it. We describe our experiences to date in integrating IDSs, security managers, and other mechanisms and managers. Section 2 provides an overview of the QuO framework. Section 3 describes the integration of an IDS into adaptive QuO applications. Section 4 describes the integration of a dependability manager into the framework for the purpose of intrusion detection and recovery. Section 5 describes the integration of a security manager into the QuO framework. Section 6 describes experience to date from these research and integration efforts. Section 7 describes related research projects. Finally, Section 8 presents some concluding remarks and future planned activities.

2. Overview of QuO

The distributed object computing (DOC) paradigm is the most advanced, mature, flexible context available today for the development of large-scale, network-based systems. In DOC, software is broken up into collections of objects dispersed throughout the network, and client objects invoke operations on servant objects to accomplish the interactions and functionality needed to achieve the goals of a running application. The CORBA DOC model [18] is illustrated in Figure 1. A client object invokes a method call on a remote object as if it were a local object. The method call is handled by a local stub, or proxy, which marshals the data for delivery to a local Object Request Broker (ORB). The ORB sends the method call across the network using an Inter-ORB Protocol, such as the Internet Inter-ORB Protocol (IIOP), to an ORB local to the servant object. That ORB delivers the method call to a skeleton, which unmarshals the data and delivers the method call to the object's implementation. Upon completion of the method's processing, any return values are delivered back to the client by the reverse process.

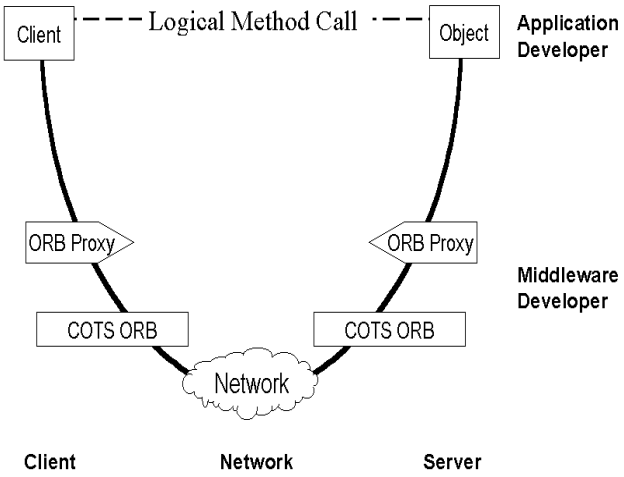

Figure 1. The CORBA Distributed Object Model

DOC middleware effectively hides many of the complexities of distributed computing such as remote location interoperability, heterogeneity, common services, and synchronization, exposing only the functional interfaces of components. However, there are increasingly more distributed applications that must control or react to how services are delivered, not just what services are delivered. Applications such as national security, military, health care, medical, multimedia, and financial systems often have critical requirements, such as security, dependability, and realtime performance. DOC middleware falls short in providing support for these requirements (as does all other forms of middleware) because it hides the details necessary to specify, measure, and control quality of service (QoS) and does not provide support for building systems that can adapt to changes that affect QoS. Because of this, developers of critical applications often find themselves programming around the distributed object infrastructure, effectively gaining little or no advantage from the middleware. The problem gets worse when an application is distributed over a WAN, which is inherently more dynamic, unpredictable,

and unreliable than a LAN.

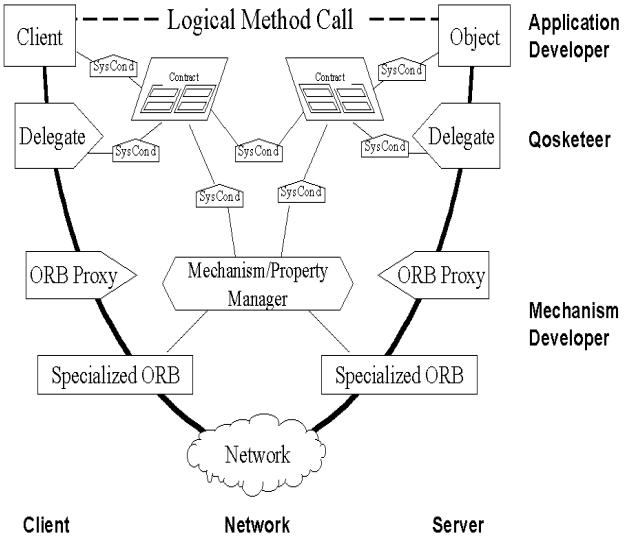

Figure 2. The QuO Remote Method Call Model

We have developed Quality Objects (QuO), a DOC framework for developing distributed applications that can specify their QoS requirements, the system elements that need to be monitored and controlled to measure and provide QoS, and behavior for adapting to changes in QoS. In this way, QuO opens up distributed object implementations [10], providing control of both the functional aspects of a program and its implementation strategies, which are often hidden behind the functional interfaces. QuO provides the capabilities for developing DOC applications that can do the following in addition to their functional behavior:

- Specify operating regions and service requirements. This can include specifying the levels of desired security or intrusion awareness (which might change dynamically based upon changes in the environment), operating modes, normal and abnormal operating regions, and known attack signatures.

- Measure environmental and system conditions. The application can insert and utilize probes in the system in order to measure resources, characteristics, and behavior. The application can also receive information from IDSs, security policy managers, and other property managers.

- Access to control interfaces. The application can pass information to IDSs, security policy managers, and other property managers to request levels of service and to notify of events that might indicate intrusions or other problems. The application can also access system resource management control interfaces to achieve its desired level of service.

- Adapt and reconfigure. The system can adapt to changing conditions at all levels, coordinated through the QuO middleware. For example, in response to an intrusion alert from an IDS, a security policy manager might respond by tightening its access control. Meanwhile, the QuO middleware can reconfigure so that the application is using servant objects only on trusted hosts. If this causes overloading of the trusted hosts, the application can adapt by changing to a mode in which it is only performing critical operations or is accepting the degraded performance.

The QuO functional path, illustrated in Figure 2, is a superset of the CORBA functional path illustrated in Figure 1.

The operating regions and service requirements of the application are encoded in *contracts*, which describe the possible states the system might be in and actions to take when the state changes.

QuO inserts *delegates* in the CORBA functional path. The delegates project the same interfaces as the stub (client-side delegate) and the skeleton (server-side delegate), but support adaptive behavior upon method call and return. That is, the delegate checks the state of the system, as recorded by a set of contracts, and chooses a behavior based upon it.

System condition objects provide interfaces to system resources, mechanisms, and managers. They are used to measure the states of particular resources, mechanisms, or managers that are relevant to contracts in the system and to pass information to control interfaces to achieve the levels of desired services. System condition objects provide the ability to access different IDS or security interfaces in a consistent manner. They also play a role in translating between application-level concepts, such as critical operating modes, to resource and mechanism-level concepts, such as encryption methods or access descriptions. Higher-level system condition objects can interface to other, lower-level system condition objects, forming a tree of system condition objects that translate mechanism data into application data.

QuO provides a suite of Quality Description Languages (QDL), similar to CORBA's Interface Description Language (IDL), and code generators, similar to the stub and skeleton generators of IDL compilers, for describing and generating the components of QuO applications [13], [14]. In addition, QuO provides a run-

time kernel, which coordinates contract evaluation and provides other runtime QuO services [26]. QuO also provides an extensive library of instrumentation probes, as well as the support to insert them throughout the remote method invocation path, for gathering performance, statistic, and validation information.

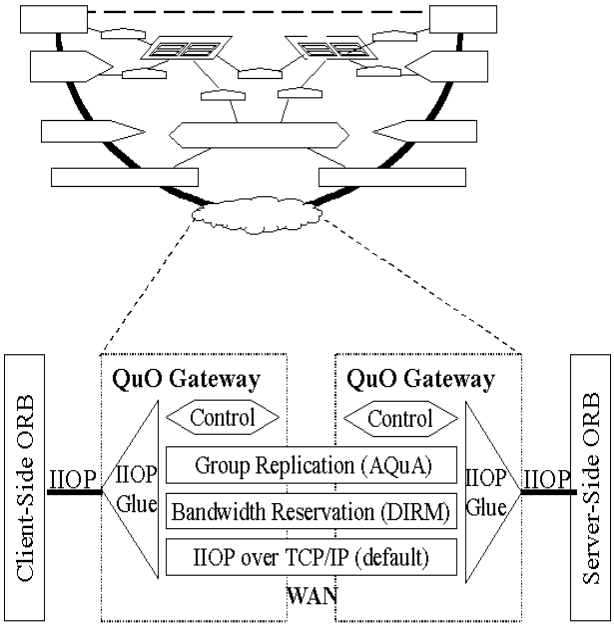

Figure 3. The QuO Gateway

QuO also provides a general object gateway component, which supports interfacing to below-the-ORB mechanisms and special-purpose transports, as well as providing a lower level point for object level decision making. The object gateway, illustrated in Figure 3 and described in more detail in [20], intercepts IIOP messages from the client side ORB and delivers IIOP messages to the server side ORB (on the message send; on the message return the process is reversed). In the middle, it translates the IIOP into the special purpose transport protocol (e.g., group multicast in a replicated, dependable system) or performs appropriate adaptation or control (e.g., in an access control system, authenticating the sender and verifying access rights to the destination object).

3. Integration of IDS with QuO

By integrating IDSs with QuO we add intrusion detection and response to the repertoire of QuO applications' adaptivity. As a result, agility and survivability of QuO applications are enhanced. In this section, we describe an experiment demonstrating how a commercial IDS, a simple custom developed IDS, and application-specified intrusion detection are all integrated to provide intrusion awareness and adaptive behavior in response to intrusion detection at the application level. This application is fairly simple, but illustrates a number of important capabilities, including the following:

- Integration of commercial and non-commercial IDSs using the QuO framework.
- An application seamlessly interfacing to multiple IDSs, enabling the IDSs to cooperate through the application layer and increasing intrusion coverage.
- An application participating in the intrusion detection process, by recognizing conditions that can indicate intrusions but that are not detected by the IDSs.
- An application adapting to survive potential intrusions, triggered by outputs of the IDSs.

In contrast, the application can use the IDSs as mechanisms to turn on, turn off, or change the level of intrusion detection provided based upon its operating mode and security needs.

3.1. Overview of the Experimental Survivable Application

We developed an example application to demonstrate IDS integration with QuO. This experimental application implements a simple inventory with a fixed set of inventory items, as illustrated in Figure 4. The inventory data is stored as files in a designated data directory. Two servers manage the inventory: one more secure than the other. The client program, representing the inventory control system, provides a user interface through which users can identify themselves (i.e., log in), add or consume items in the inventory, and log out. Both servers can respond to requests from the clients, but the more secure one authenticates (using a simple authentication scheme) each request and grants access only to certain clients. This is an example of the alternative behaviors that the QuO middleware is intended to mediate. In normal mode, all client requests are serviced by the non-authenticating (and therefore faster) server. As conditions indicate that intrusions are more likely, the inventory control system adapts to use the authenticating server and then, eventually, may cut off all access to non-privileged users. The client and server programs are simple CORBA objects. No intrusion detection or adaptation is programmed into them. For this example, all adaptation is built into the QuO middleware layer. We utilize three intrusion detection instruments in this example:

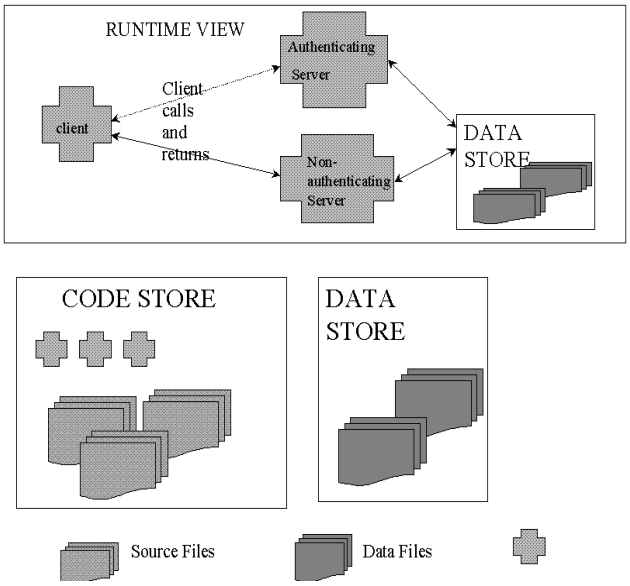

Figure 4. Storage and Runtime View of the Inventory Application

- Tripwire, a commercial file system integrity checker [11];
- FileCounter, a simple, custom developed directory access checker; and
- Specifications, encoded into the QuO contract regions, indicating the expected round trip response time range and recognizing when client requests are being abnormally delayed (possibly because they are being intercepted, or because of host or network attacks). Note that such a delay, by itself, is not a good indicator for an intrusion: a benign network congestion could cause a false positive. This merely serves as an example of an indicator of potential problems for an intrusion aware application.

We use Tripwire to monitor the file system section that stores the source and executable code of this application and FileCounter to monitor the data directory. We use system condition objects to interface to Tripwire and FileCounter, each of which normally provides its own custom interface. These system condition objects project values from the IDSs to the QuO layer and provide common access to the control interfaces provided by the IDSs. Tripwire's system condition object (called IDSValue) projects a value to the QuO contract indicating whether the integrity of the code store has been violated (in Tripwire's view). Similarly, FileCounter's system condition object (called FileAddedOrDeleted) projects a value indicating to the QuO contract indicating whether a data file has been lost or added.

Deviation from normal operating behavior often points toward potential problems. For instance, if a server returns a value that does not make any sense in the current context, the client may become suspicious that the server has been compromised. Similarly, if it takes an abnormal amount of time to fulfill a request to the inventory server, the client may become suspicious that there is a problem in the network, a host, or in the server. It is straightforward to encode such normal operating ranges in QuO's contract regions and to specify adaptive behavior to trigger when the application falls outside normal ranges. In this example, we use the contract and a simple system condition object (called TimeTaken) to measure the average round trip time of method calls and watch whether it falls outside of the expected range. As we stated earlier, there may be a variety of causes of this abnormal behavior, only some of which are the result of intrusions. Determining the actual cause falls somewhere between system troubleshooting and intrusion detection.

3.2. Basic Integration Architecture

The example application includes the three system condition objects described in Section 3.1: IDSValue, FileAddedOrDeleted, and TimeTaken. IDSValue, illustrated in Figure 5, provides the QuO interface to Tripwire. Tripwire can be initialized to monitor specific sections of a file system for particular attributes, such as permissions or modification times, of the files and directories in that section. Tripwire computes a database upon initialization. At runtime, it recomputes the database, compares the newly computed database against the initial one, and presents a result set that indicates what occurred in the file system section between the runs.

In our example, we wrap Tripwire with a CORBA object interface that runs Tripwire periodically and analyzes its output. If there is any change in the file system section this CORBA object returns a value 1, otherwise it projects a value 0. The IDSValue system condition object is hooked to this CORBA object. One of its threads polls Tripwire's CORBA wrapper to get the latest value. The other thread responds to requests from QuO contracts for the latest value.

Of course, if the IDS system were a CORBA object already, then no CORBA wrapper is necessary. The other IDS component is a CORBA object that monitors files in a directory. FileCounter produces a value 1 if a file is added or deleted in that directory, 0 otherwise. We have used this as a simple custom developed IDS and integrated it with QuO along with Tripwire, in order to

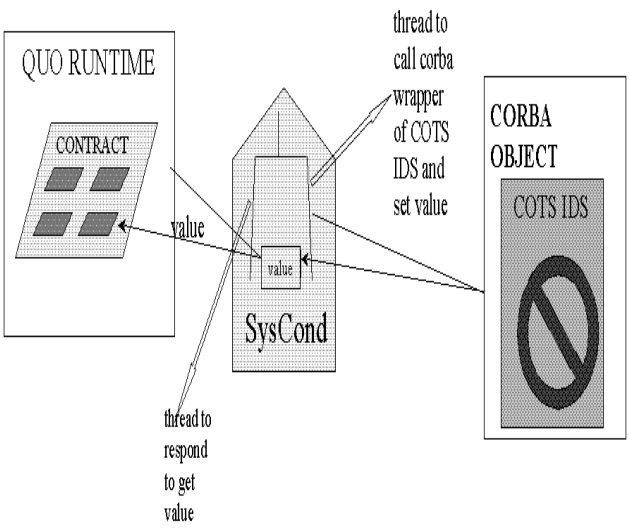

Figure 5. Interfacing a System Condition Object with a COTS IDS

experiment with multiple IDS inputs.

The QuO contract used in this example defines the following operating regions, each defined in terms of the system condition objects described above:

- NORMAL : $(TimeTaken < 500ms)$ and $(IDSValue = 0)$ and $(FileAddedOrDeleted = 0)$
- TIME SUSPECT : $(TimeTaken >= 500ms)$ and $(IDSValueSC = 0)$ and $(FileAddedOrDeleted = 0)$
- ACCESS SUSPECT : $(IDSValueSC = 1)$ xor $(FileAddedOrDeleted = 1)$
- INTRUSION LIKELY : $(IDSValueSC = 1)$ and $(FileAddedOrDeleted = 1)$

The application adapts its behavior based on the current region as follows:

- NORMAL region: client requests are forwarded to the non-authenticating server.
- TIME SUSPECT region: A warning message is displayed to the user notifying of the unusual delay and urging caution in using the inventory system. The client's requests are still forwarded to the non-authenticating server.
- ACCESS SUSPECT region: A warning message is displayed to the user stating that a potential access violation is detected and that requests may or may not be granted. Clients' requests are now forwarded to the authenticating server, which grants access only to privileged users.
- INTRUSION LIKELY region: A warning message is displayed stating that it is highly likely that there was an intrusion that could have compromised the code and data store of the application, and all client requests are returned without making any remote call. This implies that only some inventory operations (i.e. that could be handled locally, for example a query about an inventory item could be answered, with some degree of accuracy, based on the value last seen) are available at this region. One can extend the range of available operations by using various technologies such as object caching [27] or maintaining a local replica of the inventory server.

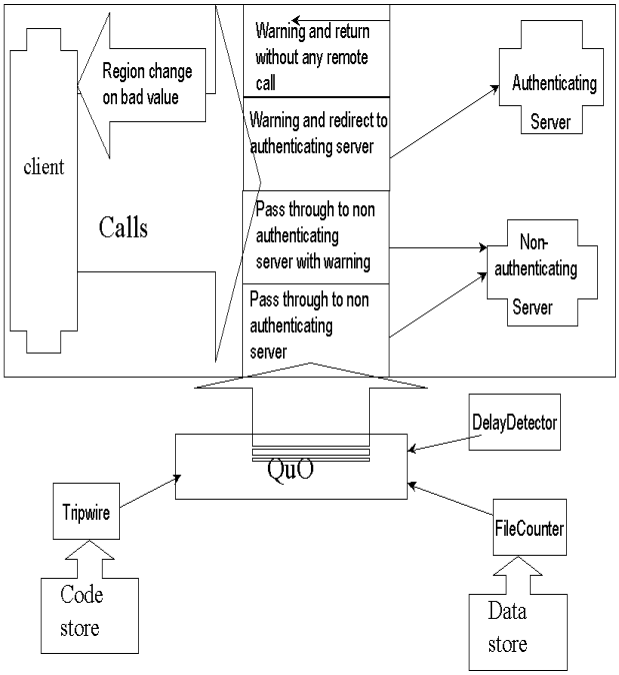

Figure 6. Intrusion Aware Inventory Application and Its Runtime Behavior

In addition, if at any time the return value is negative (under normal circumstances, the server should never return a negative value) that value is reported to the user and the contract region is switched to ACCESS SUSPECT. All of these adaptive behaviors are specified in QuO's specification languages. Figure 6 presents a pictorial representation of the full IDS aware inventory application after the QuO-IDS integration.

4. Integration of QuO and Other Property Managers

There are numerous tools and mechanisms, which we generically refer to as property managers, that manage low level system resources. These property managers provide the capabilities that QuO applications must measure and control in order to achieve and adapt to levels of service in the system. For example, we have developed a bandwidth management system that uses RSVP to reserve network bandwidth, providing improved network response for the application [1]. Similarly, we have built an availability example around the Proteus dependability management system [3], which uses group communication, replication, and fault recovery to provide higher levels of availability to the application. These integrated property managers can be used to develop adaptable, survivable applications in the following ways:

- They can provide information indicating anomalous behavior and its causes. In general, more precise information means improved adaptive response. For example, unusual delay (which we have used as an indicator of a potential problem) could be caused by the network, by a compromised object, a crashed object, a compromised host or a crashed host. Given additional information, the application could adapt intelligently.
- They could provide the application more adaptation opportunities. For example, if it is the network that is the source of an abnormal delay, the QuO application can attempt to reserve bandwidth, if such a manager is available.

We have begun integrating the Proteus dependability manager [19] into the QuO-IDS example described in Section 3, in order to illustrate how other property managers working in concert with IDSs and adaptable applications can produce more flexible, survivable applications.

The QuO-IDS integration example as described in Section 3, although survivable in the face of some types of intrusions, is hardly dependable. If one of the server objects dies, the whole application dies. Using the Proteus dependability manager, it is possible to make the application more dependable in the sense that it can tolerate a certain number and type of faults. Proteus achieves that by replicating the server objects on multiple hosts. However, in the course of its fault recovery, Proteus usually hides faults from the application. That is, when an object crashes, Proteus restarts it and updates its state, to maintain a level of dependability transparent to the application. In the context of a survivable, intrusion-aware application, fault masking may hide potential clues for intrusion.

In conjunction with our research, the University of Illinois has developed a fault notification interface for Proteus. Using this interface we have developed a CORBA object, called FaultObserver, that receives notification from Proteus about faults such as the unsuccessful start of a replica, crash of a replica, and crash of a host. Each notification consists of a set of fault information which can be stored and analyzed to recognize patterns of failures that might indicate an intrusion. The following are two examples conditions that FaultObserver currently recognizes:

- POTENTIAL INTRUSION OF HOST x: this indicates that either the host named x crashed or replica start attempts on this host were unsuccessful.
- POTENTIAL COMPROMISE OF OBJECT o: this indicates that either a replica of object o has crashed or attempts to start a replica of object o have been unsuccessful.

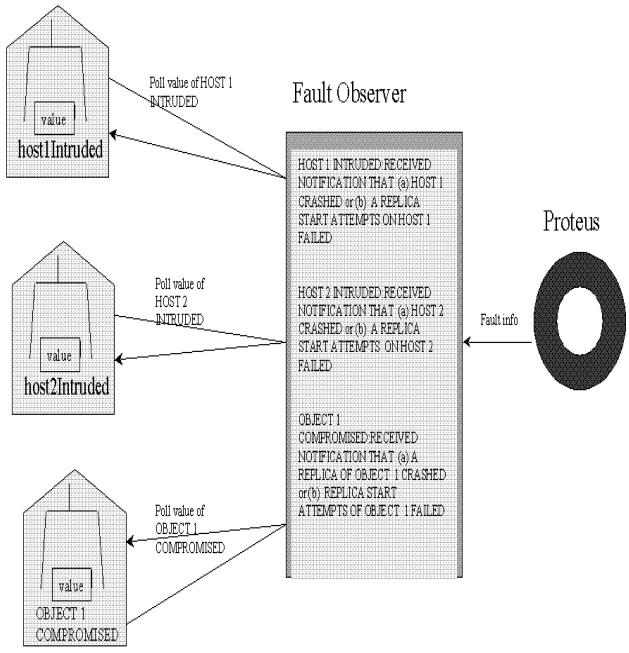

Figure 7. Integrating Proteus in the Context of Intrusion Detection and Response

Figure 7 shows three System Conditions that we have hooked up to the FaultObserver object. Each of the top two projects the value of the POTENTIAL INTRUSION conditions for one of the two replication hosts and the bottom one projects the value of the POTENTIAL COMPROMISE condition for a server object.

Let us consider a simple client-server application that uses Proteus to replicate the server and in addition, also maintains a non-replicated server. Replication provides the fault tolerance and dependability, whereas the non-replicated server makes it possible for the application to bypass the replication mechanism if it chooses to. A contract for this application may include the following regions predicated on the system condition objects described above:

- HOST SUSPECT: the HOST INTRUDED condition is true for one or both replication hosts.
- SERVER SUSPECT: the OBJECT COMPROMISED condition is true for the non-authenticating server.

The application's adaptive behavior may include:

- If the application is in the SERVER SUSPECT region, the client's requests will be redirected to the different non-replicated server object.
- If the application is in HOST SUSPECT region, Proteus will be asked not to place replicas in the intruded host(s).

In addition to the notification of crash faults, which Proteus currently provides, the University of Illinois is also working on providing notification for other types of faults, such as timing and value faults, that could prove useful for a survivable application.

5. Integration of QuO and Security

The survival of a QuO application depends on more than just QuO. Attacks on QuO's environment, including operating systems, networks, and the CORBA implementation, all have the potential to completely disable QuO and any application it supports. We assume that the environment has some resistance to such attacks. We do not assume, however, that the environment offers uncircumventable security, because such security is not commonly available. We rely on the environment to slow down attackers and make their attacks visible to IDSs. QuO applications can then respond to many attacks by adapting and reconfiguring, as described in previous sections.

To enhance the defenses of a QuO application we also offer access control. The application designer can use access control at the CORBA level to ensure that only authorized users and programs may invoke the application and that unauthorized interference with the application's internal mechanisms is not possible. QuO integrates access control technology, in the form of Network Associates' OO-DTE, for this purpose.

5.1. Object-Oriented Domain Type Enforcement (OO-DTE)

OO-DTE [24] is an object-oriented, policy-driven mechanism for fine-grained access control in distributed systems. It is policy-driven because it bases access control decisions on a single, explicit, written policy governing an entire distributed application. The application developer describes the access controls once, and OO-DTE enforces these controls consistently at all locations where the application runs. This approach eliminates the need for a developer to set operating system access controls manually on every host.

OO-DTE is object-oriented because it controls access in terms of objects and the clients that use them. It is fine-grained because it allows control over access to each object and each object method individually. The protection it offers is therefore more flexible than that offered by firewalls, for example.

OO-DTE does not assume that an application's components are all trusted to the same degree. Instead, each client and object must authenticate each other using cryptographic means (currently SSL [16]).

5.2. Protecting QuO Applications

QuO applications use the OO-DTE mechanisms directly for protection. The developer writes a security policy that controls both the access of users to the application and access of application components to each other. The security policy refers to methods declared in CORBA IDL, and in this way, it is like QuO's other specification languages. The security policy must cover all of the interfaces used in the application, including those used by QuO callbacks and by QuO delegates for adaptive behaviors.

5.3. Protecting QuO Infrastructure

The QuO infrastructure, consisting of QoS contracts, system condition objects, and a kernel, is built from the same CORBA mechanisms as QuO applications. Just as application clients use CORBA to invoke methods on application objects, so QuO delegates use CORBA to invoke methods for accessing system condition objects, for initializing contracts, and for causing the kernel to evaluate contracts.

The QuO infrastructure is therefore subject to the same kinds of attack as is every application. For example, a malicious program could try to trigger QuO's adaptation mechanisms at the wrong time by changing the value of a system condition, or to disable QuO altogether by changing QoS contracts. We protect the QuO infrastructure using OO-DTE access control just as for applications. The QuO infrastructure code must use an

ORB or ORBs with OO-DTE enabled (in fact, if this were not so, OO-DTE would not work because the infrastructure and the delegates could not establish mutual authentication) and the security policy must describe access control for infrastructure methods. The policy must prohibit the delegates from damaging the infrastructure but still give them the access they need for adaptation.

The need to protect the QuO infrastructure has implications for the design of QuO. We assume that threats to QuO come from application software and not from code within the QuO infrastructure we supply. Then QuO must not allow application software to run in the same process address space as the QuO infrastructure. OO-DTE cannot protect QuO from malicious code in the same address space because that code may bypass CORBA altogether and directly access QuO code and data structures. So the infrastructure we supply with the QuO distribution can be considered part of a trusted computing base (TCB) [4] and all untrusted extensions to that infrastructure and applications are outside this TCB and must run in other address spaces. For example, the QuO kernel cannot be run securely in the same process as an application client, even though to do so would enhance performance.

5.4. Dynamic Access Control

Access control, just like other QoS properties managed by QuO, may need to be changed at runtime. For example, if an IDS notifies QuO of a possible intrusion, it may be desirable to go into an alert mode that allows the intruder to be more easily identified. Nonessential processes may be stopped, access controls tightened, and other QoS attributes set to preestablished values. In this example and others, the security policy is simply one aspect of QoS.

There are currently two ways to change access controls dynamically in QuO:

1. using OO-DTE's policy distribution tools;
2. using QuO's QDL languages.

For the first approach, a new policy is pushed from a central security policy manager component to all OO-DTE interceptors, which then begin implementing it. For the second approach, QuO's specification languages are used to specify an alternate behavior in which access to some method is denied, as illustrated in the adaptable, survivable example in Section 3.

The approach of changing access controls is preferable. It applies globally, whereas the second approach is purely local to the delegates affected by the specification. The first approach is also more trustworthy than the second, where the code implementing the access control runs in a delegate in the client's address space, and therefore may be circumvented if the client is malicious.

6. Experience to Date

Although this paper reports on work in progress, we have to this point developed some experience about the nature of application assisted intrusion detection, and the feasibility of the approach to survivability which integrates together a number of more localized protection and security mechanisms to achieve more effective coverage. In this section we discuss four of these areas: using multiple complementary IDSs, integrating off the shelf IDSs, integrating security property management, and application strategies that can complement infrastructure based detection and protection mechanisms.

6.1. Multiple, Complementary IDSs and Managers

It has been shown from an experiment conducted by MIT Lincoln Laboratory for DARPA that multiple IDS systems can be more effective in identifying real attacks. MIT LL evaluated a number of IDSs, testing them on a number of different types of attacks, and scoring each according to the number of attacks that it detected and the number of false alarms it raised. The results indicated that while none of the IDSs overwhelmingly detected most of the attacks, the (hypothetical) combination of the best detectors for each attack resulted in more than two orders of magnitude reduction in false alarm rates while improving detection accuracy over commercial and Government keyword-based systems [5]. This provides the motivation for a model where there are multiple IDSs operating concurrently, and the need to organize and integrate their operation to achieve intended application oriented improvements in survivability.

One of the strengths of the QuO framework is that it provides simplified support for interfacing to multiple managers and mechanisms. Part of our current development and experimentation involves integrating multiple managers and mechanisms. The first example we developed, described in Section 3, combined a commercial off the shelf IDS with a custom developed (but simplified) one tailored for the specific need. We are now in the process of developing a demonstration application that uses the Proteus dependability manager in combination with the Tripwire IDS. This has two potential application survivability benefits. First, fault detection information collected routinely as part of dependability support can be used to aid the intrusion detection system, and second, the reconfigurability of replication assets can be used to help recover from detected intrusions.

The examples that we have developed, despite having limited interaction between managers, have delivered

the expected benefits. Some managers, such as multiple IDSs, security policy managers, and dependability managers are complementary and can produce higher levels of service when used together.

6.2. Integrating COTS IDSs

If the concept of using multiple special purpose IDS systems in concert is to be viable and extensible, then we need to be able to take off the shelf IDS systems and easily insert them into various application contexts. To test our approach to this type of integration, we used the example described in section 3. The collection of intrusion detection systems available to us was limited, largely to those sufficiently mature under development as part of the DARPA Information Survivability program, and those inexpensively available commercially. From these we chose two to work with: Tripwire, a commercially available ID discussed earlier (and successfully integrated with our concept example), and JAM, an experimental ID under development within the DARPA program.

JAM [25] is essentially a classifier system that employs learning and meta-learning techniques to build and refine the classifier. JAM has been used successfully to learn intrusion patterns in system traces [5].

One of the major problems we encountered in trying to integrate JAM with QuO is a mismatch of modes of operation. JAM currently operates in a batch mode. Although it is possible to ask it to classify a data set in an interactive manner, the current version does not provide any easy way to do it. Off-line usage provides only a small experimental footprint to complement the current runtime adaptation in QuO, either as a source of inputs for contract evaluation or as a mechanism to provide its service. Because JAM is geared towards stand alone usage with a GUI and not embedded usage as envisioned in integrating into a QuO environment, it did not have an appropriate API which could easily be used to integrate into QuO's system condition constructs. Additionally, it turns out to be a complex job to create the data definition and data sets that JAM would need in the context of and training for a new application such as integrating with QuO. Because of these issues, the experiment to integrate JAM as one of the ID systems in our adaptive application context has been postponed, pending the addition of online usage interfaces to JAM.

In general, the issues we encountered in integrating with Tripwire and in trying to use JAM fall into two broad categories:

- Interface Requirements: How does an application interface with a COTS IDS? The best possibility would be a runtime service provided by the COTS IDS. A programmable API would be the next best choice. The minimal requirement is that it should be possible to run the IDS from a CORBA object and communicate results in and out. A runtime service or a programmable APIs make this task easier, but human oriented interfaces have been encapsulated successfully as well, most often with less flexibility.

- Integration Architecture: What is an appropriate level of integration? We think that the way we have architected the integration by means of a CORBA wrapper that acts as a peer of a QuO component is a general pattern of usage that will be repeated with other COTS systems. Depending on the layer in which the QuO component operates, the COTS system may provide inputs to contract evaluation or may provide some service. If the COTS system is already a CORBA object itself, no CORBA wrapper will be needed.

6.3. Integrating Security Property Management

Based on our experience thus far with using OO-DTE in QuO, we can make several observations.

First, we learned that incorporating OO-DTE access control into QuO was straightforward. Because OO-DTE is implemented as CORBA interceptors, integrating it required only minimal changes to QuO code. Using OO-DTE successfully was largely a matter of setting up the policy and cryptographic certificates correctly for each application.

Second, we expect that using OO-DTE will no longer be straightforward in the presence of mechanisms that support other properties. For example:

- Although access control in the presence of fault tolerant replication is conceptually simple (just give each replica the same access rights), the actual implementation appears harder. In addition to handling application-level invocations, the replicas must run some replica coordination protocol. It is not yet clear what access rights are required in this protocol.

- Using access control in the presence of a realtime ORB [21] will mean porting the access control mechanisms to an alternative ORB and ensuring interoperability between ORBs.

Third, building security-aware QuO applications will mean allowing applications to have direct access to the security policy to inspect it and possibly to modify it. Currently this access is not possible. To make it possible we must encapsulate the OO-DTE policy in a CORBA object and define access methods for the policy. Once

that is done, the access rights themselves must be access controlled according to some meta-level security policy.

6.4. Applications Participating in Intrusion Detection

Our work with the QuO framework, various property managers, and the integration of ID and security has shown that many of the QoS properties with which an application is concerned are the same QoS properties that can indicate an intrusion or attack. For example, by definition denial of service attacks will manifest themselves by an application losing some service upon which it is dependent. Likewise, flooding attacks, attacks on particular hosts, networks, or processes can manifest themselves as changes in the system conditions monitored by QuO applications.

We illustrated in our example application described in Section 3 that QuO applications can specify normal and abnormal patterns of behaviors in their regions and recognize when the system is operating outside these regions. Many of these, especially if coded carefully, can indicate patterns of attack. Slow service, loss of network resources, abnormal responses by server objects, etc. can all indicate potential attacks. The application can aid IDSs by alerting them toward potential intrusions that should be analyzed or by indicating conditions in which multiple IDSs should be deployed.

In addition, QuO's system condition objects and instrumentation normally used for bottleneck identification and to drive resource management decisions, can also be used to collect system information over time that might recognize intrusions that are difficult to recognize from small windows or groups of events, such as slow degradations in service. This information can feed into off-line analysis capability, such as that currently provided by JAM. One weakness of anomaly detection IDSs is that they can be trained by intruders over time to recognize anomalous behavior as normal. An application could aid in detecting these types of intrusions by collecting information indicating slow, deliberate degradations of service or changes in behavior patterns.

Finally, we have also shown, in the example application described in Section 4, that other property managers and mechanisms can be useful in intrusion detection. We have concentrated our initial efforts on using the Proteus dependability manager, which in normal usage would attempt to mask faults that could indicate intrusions, to collect information about fault patterns, as a means of helping recognize intrusions. However, other mechanisms, such as resource management, realtime scheduling, and instrumentation, could also be focused on the job of intrusion detection. We intend to continue these experiments, by using Proteus, OO-DTE, and combinations of IDSs in concert to create more intrusion-aware, adaptive, survivable applications.

7. Related Work

7.1. OO-DTE

Section 5.1 describes the OO-DTE access control technology that we are using in QuO. Other related technologies, however, are also under development at Network Associates:

- an alternate implementation of OO-DTE that depends on a modified Unix kernel for greater security assurance;
- an alternate implementation of OO-DTE that offers coarser granularity access control but depends on a firewall for enforcement.

Enhancements to OO-DTE that would support multicasting are also under consideration. Each of these OO-DTE technologies offers possible improvements to QuO security.

7.2. Intrusion Detection System Research

There are numerous research efforts developing intrusion detection systems. Most of these are anomaly detection systems, misuse detection systems, or a combination of the two. Anomaly detection systems identify the normal behavior of a system, often through training, and detect behavior that deviates from normal. Misuse detection systems detect known patterns of attack, such as exploitation of known security holes or recognition of virus signatures. For example, Columbia's JAM is an agent-based misuse detection system that uses known patterns of fraudulent use of transaction systems and models of anomalous or errant transaction behaviors to protect financial information systems [25]. MCNC's JiNao is an anomaly detection system that identifies normal profiles of network routing and management protocols and monitors the execution of protocols in routers and switches to recognize deviations from the normal profile [9]. SRI's Emerald system is a combination anomaly and misuse detection system [17]. UC Davis has developed a prototype called GrIDS [22] that uses a graph based approach to detect anomalous activities on host computers and network traffic between them.

Most of these IDSs work by examining patterns of system calls or network traffic. In almost every case, the IDS is working on behalf but completely independent of the applications, with no interaction or involvement from the applications. In contrast, we are examining the ways in which the interaction and cooperation between IDSs, applications, and other property managers

can improve both the detection by the IDSs and the survivability of the applications. In a slightly related way researchers at RST Corp. are using application program behavior profiles for intrusion detection [8].

Computational immunology is a special case of anomaly detection based on an analogy with biological immune systems. In this approach, an IDS creates a knowledge of "self" through training, with the intent of distinguishing that "self" from "other", i.e., system attackers. Work in this area is being done at the University of New Mexico [7] and ORA. The latter have developed a CORBA immune system that defines "self" in terms of correlations between method invocations.

7.3. QoS/Quorum

There are a number of other complementary research efforts in QoS for distributed systems, many being performed under the auspices of DARPA's Quorum program. Similar to the University of Illinois' and BBN's work in dependability [3], the Eternal project is examining the use of replication and group communication in CORBA applications [15]. The DIRM [1] project created a capability for distributed applications to reserve network bandwidth in wide-area networks, and adapt during runtime to changing network resource requirement and availability. The Darwin [2] project concentrates on network resource management and the QoSME [6] project developed a Quality of Service (in the network context) management environment. Finally, the TAO [21] and Time-triggered Method Objects [12] projects are examining the realtime aspects of QoS in distributed systems.

7.4. CIDF

The Common Intrusion Detection Framework (CIDF) effort [23], sponsored by DARPA, is developing a common data format and encoding scheme for communicating intrusion event, analysis, and response information between IDSs and IDS components. CIDF's ID data format, called Generalized Intrusion Detection Objects (GIDOs), can represent system events (such as system log entries), the results of event analysis by IDSs (such as the recognition of a potential intrusion), and responses to events in a standard format that can be transported compactly and understood by other CIDF-compliant components. An Internet Engineering Task Force (IETF) working group, the IETF Intrusion Detection Working Group (IDWG), has grown out of the CIDF effort. CIDF concentrates on the interaction and cooperation between IDS components. Our research is complementary to this, by examining the interaction between IDSs and applications and IDSs and other property managers.

8. Conclusions and Further Work

This work is at the intersection of three distinct but related themes. First, is the integrated resource management theme, under the organizing paradigm of improved Quality of Service, one dimension of which is concerned with security attributes. The second is adaptable application behavior, motivated prominently by the changing operating environments and QoS requirements frequently found in modern, highly internetworked distributed applications. Third is the advancement of improved infrastructure for identifying, isolating and responding to information attacks leading to more survivable applications. We described experimental results in concept demonstrations linking these three ideas as a means of describing the technical concepts underlying each.

Our interim conclusions to date are along each of these threads, and have reinforced the original notions that led to this work:

1. Security issues can indeed be developed and controlled within a common QoS umbrella, making it both more feasible and practical to coordinate security strategy along with resource management strategies for other common attributes such as dependability and real time performance.

2. Adaptive behavior, along with infrastructure to support it, is both feasible and practical, as a means of providing more user satisfying application behavior under changing circumstances and requirements.

3. An environment where applications work in a cohesive and complementary way to the infrastructure components that are servicing them is both desirable and feasible, and opens up a wide space for tradeoff studies regarding the most effective complementary coverage of multiple objectives and performance and cost constraints.

At a more detailed technical level, our conclusions include that the QuO environment can be effectively used to support adaptive security policy, interoperation of off the shelf ID systems for improved coverage, and reconfigurable application behavior, which can lead to a much more survivable set of application services.

There are a variety of next steps being pursued. Chief among these are:

1. Trial usage of the experimental middleware and property managers available currently, to refine and evaluate both the software engineering and performance issues of QuO and the integrated mechanisms.

2. Enabling the technology for developing and packaging useful, reusable adaptive behaviors, so that they may be exported from one environment to another.

3. Larger scale experiments with multiple, more powerful ID systems providing varying degrees of complementary and overlapping coverage, and the integration with more powerful and flexible response mechanisms to control adaptive reconfiguration.

4. Methods for coordinating application level information collection and behavior with system level resource management policies and strategies for the various QoS dimensions under investigation.

5. Integrating the individual viewpoints of the system components into a more cohesive overall system behavior.

9. Acknowledgements

The authors would like to acknowledge the other members of the QuO team: John Zinky, Mark Berman, David Karr, James Megquier, Richard Shapiro, David Bakken, and Rodrigo Vanegas; members of the University of Illinois Proteus and AQuA teams: William Sanders, Michele Cukier, and Jennifer Ren; and members of the OO-DTE development team at NAI Labs at Network Associates: Durward McDonell, David Sames, and Gregg Tally - all of whom contributed to the work described in this paper.

References

[1] BBN Distributed Systems Research Group, DIRM project team. DIRM technical overview. Internet URL http://www.dist-systems.bbn.com/projects/DIRM, 1998.

[2] P. Chandra, A. Fisher, C. Kosak, T. S. Ng, P. Steenkiste, E. Takahasi, and H. Zhang. Darwin: Resource management for value-added customizable network service. In *Proceedings of the Sixth IEEE International Conference on Network Protocols (ICNP'98)*, Austin, October 1998.

[3] M. Cukier, J. Ren, C. Sabnis, D. Henke, J. Pistole, W. Sanders, D. Bakken, M. Berman, D. Karr, and R. E. Schantz. AQuA: An adaptive architecture that provides dependable distributed objects. In *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, pages 245–253, October 1998.

[4] DOD. *Trusted Computer System Evaluation Criteria*. United State Department of Defense, Washington D.C., December 1985.

[5] Dynacorp IS. Results of the DARPA evaluation of intrusion detection systems. Internet URL http://www.dyncorp-is.com/darpa/meetings/id98dec/Files/MIT-LL1999.

[6] P. Florissi. QoSME: Quality of service management environment. Internet URL http://www.cs.columbia.edu/dcc/quosockets/, 1998.

[7] S. Forrest, S. A. Hofmeyr, and A. Somayaji. Computer immunology. *Communications of the ACM*, 40(10), October 1997.

[8] A. K. Ghosh, A. Schwartzbard, and M. Schatz. Using program behavior profiles for intrusion detection. In *Proceedings of the Workshop on the State of the Art and Future Directions of Intrusion Detection and Response,*, February 1999.

[9] J. F. Jou, S. F. Wu, F. Gong, W. R. Cleaveland, and C. Sargor. Architecture design of a scalable intrusion detection system for the emerging network infrastructure. Technical report, MCNC, Dep. of Computer SC. North Carolina State University, April 1997. available from http://www.anr.mcnc.org/JiNao.html.

[10] G. Kiczales. Beyond the black box: Open implementation. *IEEE Software*, January 1996.

[11] G. Kim and E. Spafford. The design and implementation of Tripwire: A filesystem integrity checker. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, 1994.

[12] K. Kim. Realtime object-oriented distributed software engineering and TMO. *International Journal of Software Engineering and Knowledge Engineering*, July 1999. to appear.

[13] J. P. Loyall, D. E. Bakken, R. E. Schantz, J. A. Zinky, D. Karr, R. Vanegas, and K. R. Anderson. QuO aspect languages and their runtime integration. *Proceedings of the Fourth Workshop on Languages, Compilers and Runtime Systems for Scalable Components*, May 1998.

[14] J. P. Loyall, R. E. Schantz, J. A. Zinky, and D. E. Bakken. Specifying and measuring quality of service in distributed object systems. In *Proceedings of The 1st IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 98)*, April 1998.

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

[16] Netscape Corporation. Secured socket layer. Internet URL http://home.netscape.com/security/techbriefs/ssl.html, 1999.

[17] P. G. Neumann and P. A. Porras. Experience with EMERALD to date. In *Proceedings of the 1st Usenix Workshop on Intrusion Detection and Network Monitoring*, April 1999.

[18] OMG. *CORBA/IIOP 2.3, OMG Technical Document 98-12-01*. Object Management Group, Framingham. MA, December 1999.

[19] C. Sabnis, M. Cukier, J. Ren, P. Rubel, W. Sanders, D. Bakken, and D. Karr. Proteus: A flexible infrastructure to implement fault tolerance in AQuA. In *Proceedings of the IFIP International Working Conference on Dependable Computing for Critical Applications*, January 1999.

[20] R. E. Schantz, J. A. Zinky, D. A. Karr, D. E. Bakken, J. Megquier, and J. P. Loyall. An object-level gateway supporting integrated-property quality of service. In *Proceedings of The 2nd IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 99)*, May 1999.

[21] D. Schmidt, D. Levine, and S. Mungee. The design of the TAO realtime Object Request Broker. *Computer Communications*, 21(4), April 1998.

[22] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS -a graph based intrusion detection system for large networks. In *Proceedings of the 19th National Information Systems Security Conference*, September 1996.

[23] S. Staniford-Chen, B. Tung, and D. Schnackenberg. The common intrusion detection framework. In *the Information Survivability Workshop*, October 1998. Position Paper.

[24] D. F. Sterne, G. W. Tally, C. D. McDonell, D. L. Sherman, D. L. Sames, P. X. Pasturel, and E. J. Sebes. Scalable access control for distributed object systems. In *Proceedings of the 8th Usenix Security Symposium*, August 1999.

[25] S. Stolfo, A. Prodromidis, S. Tselepis, W. Lee, D. Fan, and P. Chan. JAM: Java agents for meta learning over distributed databases. In *Proceedings of KDD-97 and AAI 97 Workshop on AI Methods in Fraud and Risk Management*, 1997.

[26] R. Vanegas, J. A. Zinky, J. P. Loyall, D. Karr, R. E. Schantz, and D. E. Bakken. QuO's runtime support for quality of service in distributed objects. *Proceedings of Middleware 98, the IFIP International Conference on Distributed Systems Platform and Open Distributed Processing*, September 1998.

[27] J. Zinky, D. Bakken, L. O'Brien, V. Krishnamurthy, and M. Ahmed. PASS - a service for efficient large scale dissemination of time varying data using CORBA. In *Proceedings of the 19th International Conference on Distributed Computing*, Austin, June 1999.