An Architecture for Security-Oriented Perfective Maintenance of Legacy Software ⋆

D. Cotroneo, A. Mazzeo, L. Romano, * S. Russo

Università degli Studi di Napoli "Federico II"

Via Claudio 21 – 80125 Napoli, Italy

Abstract

This work presents an implementation strategy which exploits separation of concerns and reuse in a multi-tier architecture to improve the security (availability, integrity, and confidentiality) level of an existing application. Functional properties are guaranteed via wrapping of the existing software modules. Security mechanisms are handled by the business logic of the middle-tier: availability and integrity are achieved via replication of the functional modules, confidentiality is obtained via cryptography. The technique is presented with regard to a case study application. We believe our experience can be used as a guideline for software practitioners to solve similar problems. We thus describe the conceptual model behind the architecture, discuss implementation issues, and present technical solutions.

Key words: Security, Perfective Maintenance, Legacy Software, CORBA, Replication

⋆ Work performed with support of Italian Ministry of Research and University (MURST).

* Corresponding author: tel: +39-0817683834 Fax: +39-0817683816; e-mail:

1 Introduction

Security has become a key requirement for the vast majority of current applications. Ideally, a secure system is one which always enables authorized users to access system services, and never allows unauthorized users to do so. Security is thus a composite attribute, which takes into account the following features [2]:

- Availability - A system is available if it is providing the expected service;
- Integrity - Avoids that messages and stored data be tampered with by someone braking into the system;
- Confidentiality - Provides privacy for messages, services, and stored data by hiding information.

As systems are being opened to the Internet, commercial traders, financial institutions, service providers, and consumers are exposed to a variety of potential damages, which are often referred to as electronic risks [3], [4]. These may include direct financial loss resulting from fraud, theft of valuable confidential information, loss of business opportunity through disruption of service, unauthorized use of resources, loss of customer confidence or respect, and costs resulting from uncertainty. In order to mitigate risks and promulgate the deployment of information systems in open networked environments, applications must guarantee high security levels.

However, there is a great deal of existing applications around, which were not designed with the objective of providing high security, since they were supposed to operate in protected environments, which were immune from the

lrom@unina.it

threats of open networked environments. These applications are thus inadequate to operate in modern Internet and intranet scenarios. Nevertheless, they are often a valuable heritage, since they i) fully satisfy system functional requirements, ii) have been thoroughly tested during years long operation, and iii) are strongly coupled with the rest of the enterprise information and production infrastructure. It is foreseeable that in the next years there will be an increasing need for perfective maintenance actions aiming at improving the security level of existing applications. In general, by perfective maintenance it is meant software maintenance performed to improve the performance, maintainability, or other attributes of a computer program [1].

In this work, we address the issue of security-oriented perfective maintenance of an existing software system. We focus on a specific class of existing applications, namely legacy applications [1] . Improving the security of legacy applications gives the opportunity to value the investment in such applications, while reducing the risk of security breaches. By legacy application we mean a software program for which maintenance actions consisting in modifications to the source code are either impossible or prohibitively costly. This may be due to a variety of reasons, including:

- The application is written in a programming language which has become obsoleted, as compared to the rest of the technologies used by the enterprise to develop its business applications. An example legacy application is a COBOL program in an enterprise environment where web oriented technologies are being massively used;

[1] We will thus use the terms "existing application" and "legacy application" interchangeably throughout the paper.

- The application is not well documented. As an example, it has been modified over the years by different people, who are not available anymore and/or who have not adequately reported in the software documentation the modifications they have performed.

We propose an implementation technique which exploits separation of concerns and reuse in a multi-tier architecture. Functional properties are guaranteed by incorporating the existing software modules. Security mechanisms are handled by the business logic of the middle-tier. Availability and integrity are achieved via replication of the functional modules: by using replication, we are able to deliver system services to authorized users, also in the presence of failures and/or intrusions affecting individual legacy modules. Confidentiality is obtained via cryptography.

This approach can be extremely effective in minimizing the development cost of a secure system, for two fundamental reasons:

(1) Code writing activity is minimized, since system services are implemented via reuse of existing software;
(2) System testing activity is reduced, since only security-related functions and interactions between individual components need to be tested.

The middle-tier is implemented as a pool of distributed objects, which cooperate over a CORBA platform. To this end, we had to address a number of issues, and in particular:

- *Provision of a secure transport to middle-tier objects* - In order for middle-tier objects to securely communicate on top of an unsecure network infrastructure, they had to use the facilities provided by the security-enabled

transport of the underlying middleware platform. Instead of explicitly programming calls to security functions in individual objects - which would have resulted in a time-consuming and error-prone activity - we exploited the CORBA Portable Interceptor mechanism to implement a security wrapper, by which we transparently hooked middle-tier objects to the secure transport layer of the middleware platform;

- *Communication between middle-tier objects and legacy server replicas* - This consisted of two distinct problems. One was to integrate multicast support in the system, i.e. the legacy server replicas and the middle-tier had to communicate over a multicast enabled infrastructure. To achieve this goal, we developed a pseudo device, which re-routed requests to an application level multicast facility. The other was to decouple communication between the Replica Manager and the Adjudicator component. To this end, we exploited the CORBA Notification Service;
- *Adjudication* - Since a secure system must be able to accommodate different requirements in terms of security and performance, depending on the specific characteristics of the deployment scenario, we had to provide a variety of alternative adjudication strategies, for the creation of a single response (to be sent to the client) from multiple sources (provided by individual server replicas). More precisely, the system supports three alternative adjudication policies: i) *Pass First*, which penalizes security but achieves the highest performance possible, ii) *Threshold-based*, which provides some protection against faults and intrusions in individual replicas, at the cost of some performance penalty, and iii) *All Agree*, which provides the highest level of protection possibile, but has the worst system response time, since no output is produced until all replicas have responded.

In the paper, we provide a thorough treatment of technical challenges we had to face, and a detailed description of the solutions we have implemented. The implementation relies solely on Commercial-Off-The-Shelf (COTS) technologies, in that commodity PCs are used, running commercial operating systems, which are instrumented with commercial middleware support. The rest of the paper is organized as follows. Section 2 illustrates the objectives of our research. Section 3 illustrates the assumptions we make. Section 4 describes the conceptual architecture of the overall system, including the roles of individual components and the interactions between them. Section 5 deals with the design and implementation of the middle-tier server, which represents the core of the system. Section 6 concludes the paper with some final remarks about results achieved and lessons learned.

2 Objectives of Security-Oriented Maintenance

Before we present the objectives of our work, we need to define some terms we use in the rest of the paper. Definitions are taken from [6] and [2]. A **fault** is the adjudged cause of an **error**, which in turn may lead to a **failure**, i.e. to a deviation from the specified service, where the service specification is an agreed upon description of the expected service. We focus on two specific categories of faults:

- Faults, whose nature is accidental, and which are originated by physical phenomena. We will call these faults **Accidental Faults**, or simply **Faults**;
- Faults, whose nature is deliberate and malicious, and which are human made. We will call these faults **Deliberate Faults**, or **Intrusions**.

A security-oriented perfective maintenance action must lead to a new system, which - in addition to implementing the same functions as the original one - satisfies the following security requirements:

(1) Availability - The service must be **available** to authorized users and **not available** to unauthorized users;
(2) Integrity - Messages delivered to authorized users must not be tampered with, i.e. their contents must not be modified;
(3) Confidentiality - Messages delivered to authorized users must not be disclosed, i.e. their contents must not be revealed to others.

In order to achieve the security objectives stated above, the new system must be able to avoid/tolerate Faults and Intrusions. More precisely, our objective is to avoid that a fault or an intrusion affecting a module running in the back-end propagates through the tiers, and eventually leads to a failure of the overall system, as illustrated in figure 1.

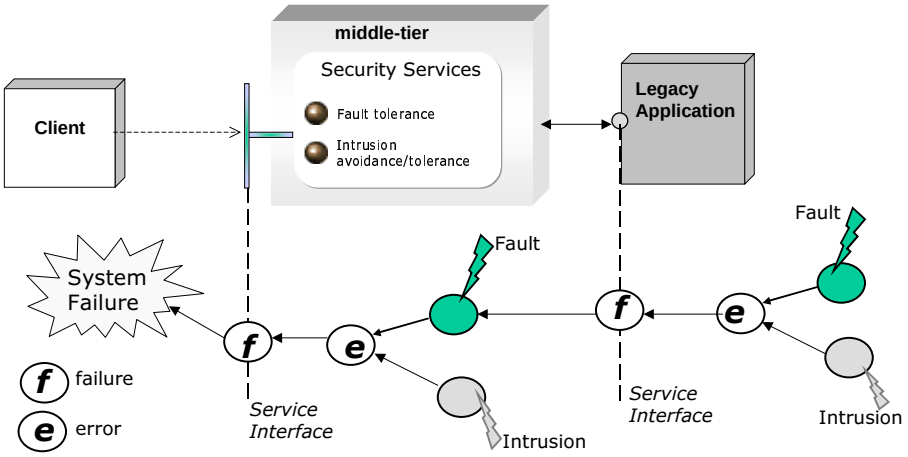

Fig. 1. Propagation of faults and intrusions.

Legacy server *failures* (middle-tier server *faults*) can be of two different types:

(1) **Timing Failure (Timeout Fault)** - That means the legacy code fails

to produce any output within a predefined amount of time. This may be originated by a variety of causes, including:

- Process/Host Performance fault - The legacy application process or host is too busy for delivering the requested service within the predefined timeout;
- Process/Host crash/hung - The legacy application process/host is down or hung, i.e. it does not reply to external stimuli;
- Network failure - The quality of the connection to the machine hosting the legacy application is too poor, or the machine can not be reached at all;
- Unavailability of crucial system resources - This typically occurs when the system is under a Denial Of Service (DOS) attack [10].

(2) **Value Failure (Value Fault)** - That means the legacy application does provide a reply, but this is incorrect, i.e. it contains errors. This may be originated by a variety of causes, including:

- Hardware-induced software errors - These are errors in the application due to faults affecting the underlying hardware [9];
- Communication errors - Such errors occur when data gets corrupted while traversing the network infrastructure. Unless "leaky" communication protocols [11] [2] are adopted, this kind of errors is fairly unlikely to happen;
- Erroneous system state/configuration - This may be the consequence of naturally occurring phenomena (such as process aging) or of intentional

[2] Most network protocols, also the so called unreliable ones, protect data with Cyclic Redundancy Codes (CRC) information. As a consequence, corrupted data is most likely detected (and dropped) than delivered. A leaky protocol is instead a protocol which allows corrupted information to be delivered to the destination.

attacks.

3 Assumptions

As far as the legacy application is concerned, we assume that its state be deterministic. If this is the case, each replica of the application can be considered as a Finite State Machine (FSM) and we are able to enforce consistency of individual replicas. It should be noted that this assumption is weaker than the (often made) stateless server assumption. A stateless server implements a combinatorial machine, i.e. at any given time its outputs only depend on the current values of the inputs, and replica consistency is thus not an issue at all. The technique we present can be extended to legacy applications which do not satisfy the deterministic state assumption (FSM assumption), provided that all sources of non-determinism be dealt with. Common sources of non-determinism are multi-threaded execution in the replicas, and system calls (especially when different operating systems host individual replicas). How to deal with non determinism when managing replica consistency is a widely investigated research topic, but it is beyond the scope of this work. The interested reader can refer to [13], and [14] to gain more insight into the topic.

As far as the middle tier server and the underlying infrastructure are concerned, we do not address issues related to faults and intrusions affecting them, since we concentrate on faults and intrusions in the back-end application. In other words, we assume that the middle tier server and the infrastructure supporting it are immune from faults and intrusions. We want to explicitly note that most research projects in the field of middleware infrastructures and

frameworks also make this assumption. To the best of our knowledge, no research project in the field of dependable distributed environments currently takes explicitly into consideration the problem of the dependability of the supporting infrastructure. This also applies to major research projects, such as AQuA [12], Eternal [8], and OGS [16]. We are only aware of a fault injection study conducted on the Chameleon system [7], where injections are performed both to the protected applications and to the supporting infrastructure.

As far as the network infrastructure is concerned, we assume a Local Area Network (LAN) connects the distributed objects which implement the middle-tier server between them, and to the replicas running in the back end. Such a connection will be considered reliable, i.e. we assume messages are always delivered without errors, and with a limited delay. This is not a very strong assumption for currently available LAN technologies.

4 Overall System Architecture

The solution we propose relies on a three-tier architecture which replicates and secures the legacy application. The system relies solely on Commercial-Off-The-Shelf (COTS) technologies. The overall system architecture is depicted in figure 2.

The figure shows the clients, the middle-tier server, and the group of replicated legacy application instances. Replicas are added/removed as they become available/unavailable. Clients access the services via the service Proxy. Communication between Clients and the Proxy is over a protected channel. The administrator can set system parameter values according to security and

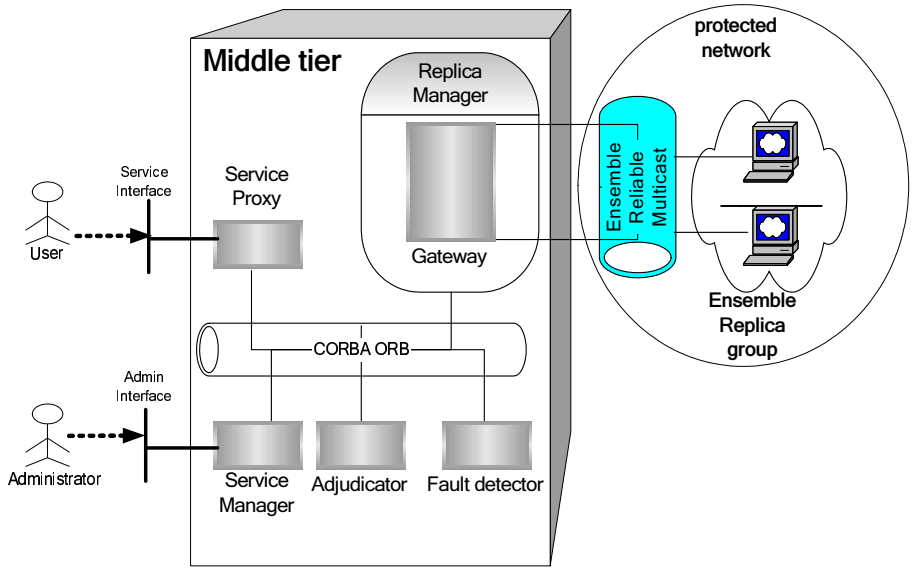

Fig. 2. Conceptual system architecture

performance requirements. The middle-tier server is a CORBA distributed application, consisting of several interacting components, which communicate over a secure CORBA platform. The middle tier adds security to the legacy application, which runs in the back-end. In the following, we describe the roles of individual components of the middle-tier.

4.1 Service Proxy

The Service Proxy encapsulates the services provided by the legacy application and exports them via the Service Interface, thus making such (enhanced) services available to the clients. The creation of a response is not a straightforward task, because all middle-tier objects are involved in the process. In particular, the ServiceProxy is only in charge of allocating the response, and it demands to the Adjudicator object the responsibility of populating it.

4.2 Service Manager

The Service Manager component is in charge of configuring all other components. It provides functions to customize the behavior of individual objects (such as the specific adjudication strategy which must be used for building the reply to be sent to the client), and to set system configuration parameters (such as the key length and value, the number of replicas to be launced, and the number of threads in the thread pools). System configuration is performed via the Admin Interface.

4.3 Fault Detector

As illustrated in figure 1, in a three-tier structure, failures of the back end components represent faults of the middle tier. We will thus use the terms failure and fault to describe the same phenomenon, depending on the specific observation point used, i.e., the legacy system or the middle tier server, respectively. The Fault Detector is in charge of detecting both Value Faults and Timeout Faults. The concepts of Value Faults and Timeout Faults were defined in section 2.

4.4 Replica Manager

The Replica Manager is in charge of enforcing replica consistency, i.e. of ensuring that all replica be in the same state at any time. To this end, it handles the following activities:

- Delivery of messages to all members of the replica group - These include

requests, replies, and group view changes. By group view change it is meant a modification of the system induced by the fact that one or more replicas have failed/have recovered from a failure, and are to be removed from/added to the group of active replicas;

- Recoveries and reconfigurations - Due to a group view change, the system must modify its operation mode. If a replica has failed, the system must switch to a degraded mode. If a replica has recovered from a failure, its state must be restored, in order to allow it to be re-integrated in the system and the operation mode of the overall system must be modified, to accommodate the new replica.

The Gateway is a key sub-component of the Replica Manager, since it wraps the server module of the legacy application, in order to allow the middle-tier to use its functions. To this end, the Gateway must perform the following tasks:

- It must handle multicast communication - Multicast communication is needed to ensure that all replicas are kept in a consistent state. Multicast connectivity was achieved by means of Ensemble [17], an application-level multicast package developed at Cornell University;
- It must handle all synchronization-related issues - In particular, it must provide a loosely synchronous communication channel to middle-tier objects. Decoupling was achieved via proper configuration and use of the CORBA Notification Service [26];
- It must handle all format-related issues - In fact, research has demonstrated that, in order for data comparison procedures to work properly, data interpretation capabilities are needed [18]. The Gateway is thus in charge of purifying the data from application-specific and platform-related depen-

dencies, before data is handed over to the adjudicator.

4.5 Adjudicator

The Adjudicator builds a single reply to be sent to the client from the multiple replies of currently available replicas. In order to do so, it interacts with the Gateway and with the Fault Detector. It determines the reply, based on the specific choices made by the administrator via the Admin Interface provided by the Service Manager. Alternative strategies are implemented, which are described in the following:

- *Pass First* - In this case, the first reply from any of the legacy server replicas is immediately sent back to the client. No comparison is performed between data from different replicas;
- *Threshold-based* - In this configuration, the system uses a threshold mechanism to determine the reply to be sent to clients. Assuming n servers are in the current group of active replicas and a threshold value $T < n$ has been set, a reply is delivered only if there are at least T replicas providing the same result;
- *All Agree* - Assuming n servers are currently in the group of active replicas, a reply is delivered only if all the n replicas provide the same result.

It is worth noting that the above described alternatives are characterized by different trade-offs in terms of security vs. performance. In particular, the *Pass First* strategy strongly penalizes security but achieves the highest performance possible. In fact, on the one side it allows the middle-tier to get data from the fastest (at any given time) back-end source, on the other side in the event

of a value fault (due to application internal faults or to intrusions), the error is propagated to the client. The *Threshold-based* strategy provides some protection against faults and itrusions in individual replicas, at the cost of some performance penalty. In fact, in order for the system to be able to produce an output, it has to wait at least for T responses from the replicas. On the opposite end of the spectrum is the *All Agree* strategy. This strategy provides the highest level of protection possibile, but the worst system response time, since no output is produced until all replicas have responded.

5 Implementation Strategy

We deployed the system over the heterogeneous platform illustrated in figure 3. The figure illustrates system configuration, and allocation of tasks to nodes.

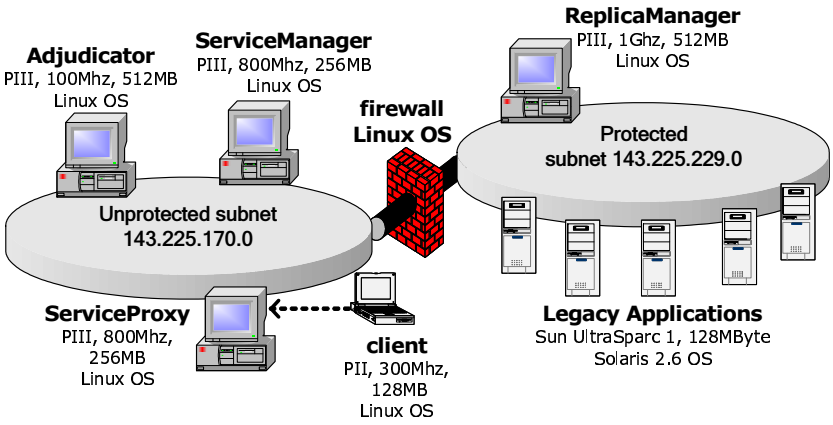

Fig. 3. Experimental testbed.

The platform consisted of two LANs, connected by a firewall. In the following, we will refer to the LAN on the left hand side of the firewall as to the "unprotected network", and to the LAN on the right hand side of the firewall as to the "protected network". Clients and middle-tier objects (except the Replica Manager) are distributed over the unprotected network. The `ServiceProxy`

communicates with the clients over a protected connection. Since the client and the `ServiceProxy` (as well all other objects, with the exception of the `ReplicaManager`, which is written in C++) are written in Java, we used Java libraries for SSL communication [20] to implement such a connection. Middle-tier objects communicate over a secure CORBA infrastructure, namely Orbix 2000 with SSL/TLS [22] transport. As far as the CORBA Notification Service is concerned, we used the classes provided by IONA Orbacus [27]. The firewall is simply a Linux box supporting iptables [5] filtering. The Replica Manager is the only middle-tier object residing on the protected network. It communicates with other middle-tier objects using the SSL/TLS secure transport of the Orbix platform, and with legacy server replicas using Ensemble unsecure multicast support.

In the rest of the section, we provide a thorough treatment of technical challenges we had to face, and a detailed description of the solutions we have implemented. In particular, we focus on the following issues:

(1) *Provision of a secure transport to middle-tier objects* - In order for middle-tier objects to securely communicate on top of an unsecure network infrastructure, they had to be able to use the facilities provided by Orbix 2000 SSL/TLS. Instead of programming calls to security functions explicitly in individual objects - which would have resulted in a time-consuming and error-prone activity - we exploited the CORBA Portable Interceptor mechanism to implement a security wrapper, which transparently hooked middle-tier objects to the SSL/TLS transport. We describe how we did this in section 5.1;

(2) *Communication between middle-tier objects and legacy server replicas* - In order to integrate the multicast support to the server replicas, we

had to connect both the `Replica Manager` and the legacy application to Ensemble multicast support. We describe how we did this in section 5.2;

(3) *Decoupling of communication between middle-tier objects* - In order to decouple the communication between the `Replica Manager` and the `Adjudicator` component, we exploited the CORBA Notification Service [26]. We describe how we configured the QoS support of the event delivery infrastructure of the CORBA Notification Service and how we used such a service to achieve our goals in section 5.3;

(4) *Adjudication of results* - The `Adjudicator` object is in charge of implementing the adjudication strategies described in section 4.5 for the creation of the response to be sent to the client. We provide a detailed description of the most challenging issues and of the solutions we have implemented in section 5.4.

5.1 Transparent Hooking of middle-tier objects to SSL/TLS transport

Our first problem was to provide a secure transport to middle-tier objects. The CORBA security service specification [29] distinguishes between two main categories of secure distributed applications:

- *Security aware applications* - If this model is chosen, components have knowledge of the underlying security mechanisms, and can play an active part in enforcing the security policies;
- *Security unaware applications* - If this other model is chosen, the majority of the components has no knowledge of the underlying security mechanisms (and thus plays no part in the security enforcement process), while a handful of specialized components (usually referred to in the literature as security

wrappers [30]) are in charge of providing security to the entire application.

The main drawback of the security aware approach is that it requires explicit programming of security related procedures in the application components. Developers would make direct use of features provided by some commercial ORB vendors, such as IONA ORBIX 2000 SSL/TLS API [20], with the final effect of wiring the security logic in the application code. On the contrary, the security unaware approach has no impact on the application code implementing business related functions, since security is added to the application via security wrappers. This has two important advantages: i) it makes this approach more suitable for providing security to existing systems, and ii) already implemented security wrappers can be re-used to provide security functions to different applications.

In this work, we take an approach for providing security services to applications, which uses CORBA interceptors [28] to transparently hook middle-tier objects to Orbix 2000 SSL/TLS transport. The approach uses a security wrapper (which is a security aware application) to provide SSL/TLS support to middle-tier objects (which are security unaware applications). We discuss the design of the interception scheme, and present its implementation.

We implemented a `Client Request Interceptor` and a `Server Request Interceptor` interface. These objects are in charge of rerouting requests and replies to the SSL/TLS support. Interceptors are security aware objects, and they are thus in charge of explicitly interacting with the SSL/TLS enabled infrastructure of the ORB. We used IAIK [19] Java libraries to handle X509 certificates [21] used by SSL/TLS. For a detailed description of individual phases of the interception process scheme which we have implemented, the in-

terested reader can refer to [32], where we describe how a challenge-response authentication scheme can be added to an existing client/server application, exploiting mechanisms of the CORBA Portable Interceptor interface. We also provide a performance analysis of the implemented scheme and of the underlying mechanisms. In order to intercept communications between all middle-tier objects, we must then register interceptors for all such objects. By doing so, we achieve the goal of hooking middle-tier objects to the SSL/TLS transport in a transparent way, i.e. middle-tier objects' requests and replies are handled by the SSL/TLS secure communication channel, without middle-tier objects being aware of that.

We want to emphasize that, by doing so, we do not need to explicitly program security functions in every individual object of the middle-tier architecture, which are only responsible of properly implementing the business logic of the application. This clean separation of duties has several important advantages. In particular, it favors system maintenance and evolution. In fact, since security can be implemented without modifying the application code, the proposed technique is an effective means for adding additional security functions and/or to incorporate support for changed security requirements in existing applications.

Although this introduces some performance penalty, due to additional actions in the invocation path, we believe the advantages related to this clean separation of responsibilities largely outweighs such a performance penalty. This is especially true if one keeps in mind that research results have shown that a major fraction of the performance penalty induced by interception is related to the registration and initialization phases of the incerceptors themselves. Since these activities are performed only once, at startup time, they have no

impact on system performance during operation.

5.2 Integration of multicast support

Our second problem was to handle multicast communication between middle-tier objects and the replicas in the current group. This responsibility was assigned to the `Gateway` object, which is a key sub-component of the `Replica Manager`. This component wraps the server module of the case study application, in order to allow the middle-tier to use its functions. Sub-roles were identified and mapped to two sub-components. These are:

- *Request* - This object encapsulates the request to be sent to the legacy system and provides a post interface which multicasts the request to all replicas, which are currently in the group;
- *Reply* - This object is a list (it is actually a multi-object), which contains all replies from individual server replicas;
- *View* - This object encapsulates the Ensemble `View.state` and `View.local` records, which describe the current configuration of the replica group. It is used by the Adjudicator to decide which actions should be taken.

We decided to use Ensemble [17] for multicast communication. We thus had to attach Ensemble to the `Gateway` object (at one end) and to the server replicas (at the other end). The former task was straightforward: we just had to link the Ensemble library to the Gateway code. The latter task was quite more complex. The legacy application (a software module for the execution of cryptographic procedures [3]) was written in C and run on a Sun Workstation,

[3] The application consisted of a client and a server module, which communicated over a TCP socket channel. The client sends the file to be signed to the server. The

equipped with Solaris 2.6 OS. It came with a TCP/IP socket based interface. We had to intercept TCP calls, and redirect them to `Ensemble`.

To achieve this goal, we developed a virtual device driver within the Solaris Kernel of the node which hosted the legacy server. The overall communication scheme is depicted in figure 4.

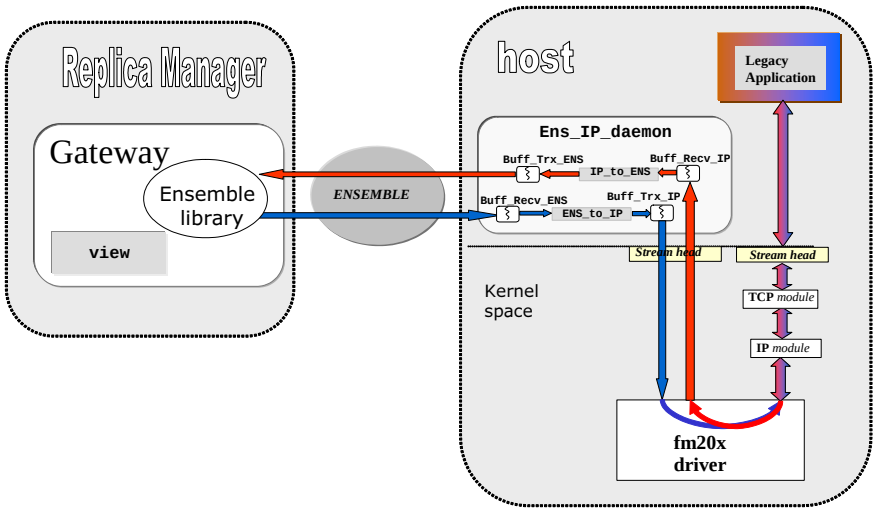

Fig. 4. Communication between the gateway and the group of legacy applications

Solaris 2.6, which derives from the Unix System V family, provides a powerful I/O mechanism, the STREAMS framework [25], consisting of a set of system calls, kernel resources, and kernel routines, which provide an effective environment for kernel services and drivers requiring modularity. The fundamental STREAMS unit is the Stream. A Stream is a full-duplex bi-directional data-transfer path between a process in the user space and a STREAMS driver in the kernel space. It is composed of a Stream head, zero or more modules, and

server computes a digest of the file and signs it with its private key. The algorithm used to compute the message digest is SHA1 [23], and the algorithm used for the digital signature is RSA [24]. For performance reasons, the private key is kept in RAM. Only the signature is returned to the client.

a driver. Data on a Stream is passed in the form of messages. Each Stream's module has its own pair of queues, one for reading and one for writing. Queues are the basic elements by which the Stream head, modules, and drivers are connected. We implemented a loopback pseudo-device driver [31], which is able to route messages between two distinct Streams. More precisely, the driver implements a sort of full-duplex pipe between two Streams: one is attached to an Ensemble-enabled daemon process (namely, `Ens_IP_daemon`), the other is attached to the legacy application. The `Ens_IP_daemon` receives and transmits messages using the Ensemble communication library. The legacy application is connected to a virtual IP address which is mapped to the pseudo device-driver. That is to say, the driver acts as a virtual network interface card for the legacy application. The daemon process `Ens_IP_daemon` is structured as a multithreaded process. Two threads, namely `Buff_Tx_ENS` and `Buff_Rx_ENS`, are in charge of handling Ensemble communication with the `Gateway` component and two other threads, namely `Buff_Tx_IP` and `Buff_Rx_IP`, are in charge of handling the communication with the pseudo device-driver. The `Buff_Rx_ENS` thread receives multicast messages sent by the `Gateway` and the `Buff_Tx_ENS` transmits packets to the driver. They interact by means of a circular queue, namely `ENS_to_IP`, according to the producer-consumer paradigm. The `Buff_Tx_ENS` thread is in charge of transmitting packets to the `Gateway` by using a UDP unicast communication, and the `Buff_Rx_ENS` thread is in charge of receiving packets from the driver. They communicate over a circular queue, namely `IP_to_ENS`, according to the producer-consumer paradigm.

We tested the performance of our driver. Tests were executed over a 100 Mbit/s fast-ethernet switched LAN. The `Gateway` sub-component was on a

commodity PC (PIII 800Mhz with 256 Mbyte of RAM) running Linux. The legacy application was on a Sun workstation (Ultra 1 with 128 Mbyte of RAM) running Solaris 2.6. Figure 5 shows the throughput, as a function of the packet size, between the `Gateway` and the legacy application in the following contexts: i) the gateway was connected to a single instance of the legacy application replica by means of the standard socket library (plain IP); ii) the gateway was connected to a single instance of the legacy application replica by means of the Ensemble communication library, and the pseudo-device driver `fm20x`.

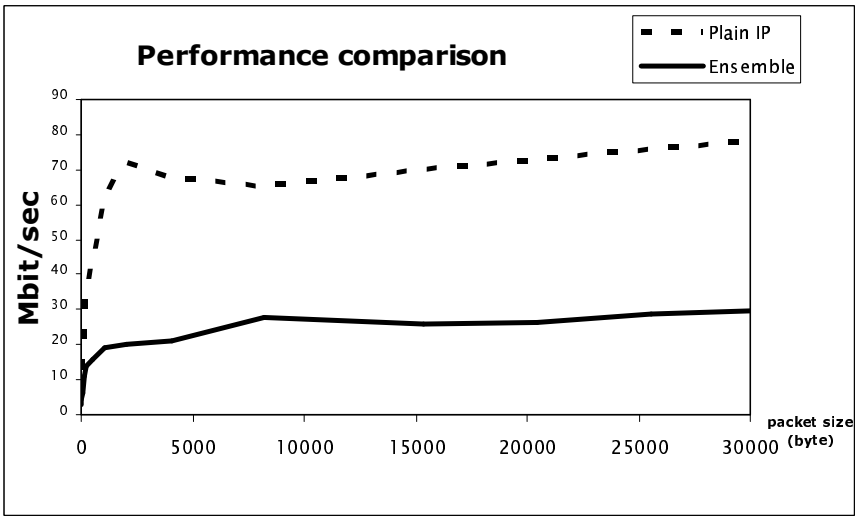

Fig. 5. Performance Comparison

As the figure shows, our communication scheme has an average throughput of 25 Mbit/s against 80 Mbit/s of socket-based communication. We believe this performance is fairly enough, if one considers that we introduced an additional tier in the communication chain, and an additional stack in the protocol.

The third problem we had to face was how to provide a loosely-coupled mechanism for communication between middle-tier objects. This required accurate design of middle-tier objects (namely the `Replica Manager`, the `Adjudicator`, and the `Service Proxy`), as well as proper configuration of the CORBA Notification Service. A simplified view of the architecture of the middle-tier, including the `Replica Manager`, the `Adjudicator`, and the `Service Proxy`) components, is depicted in figure 6.

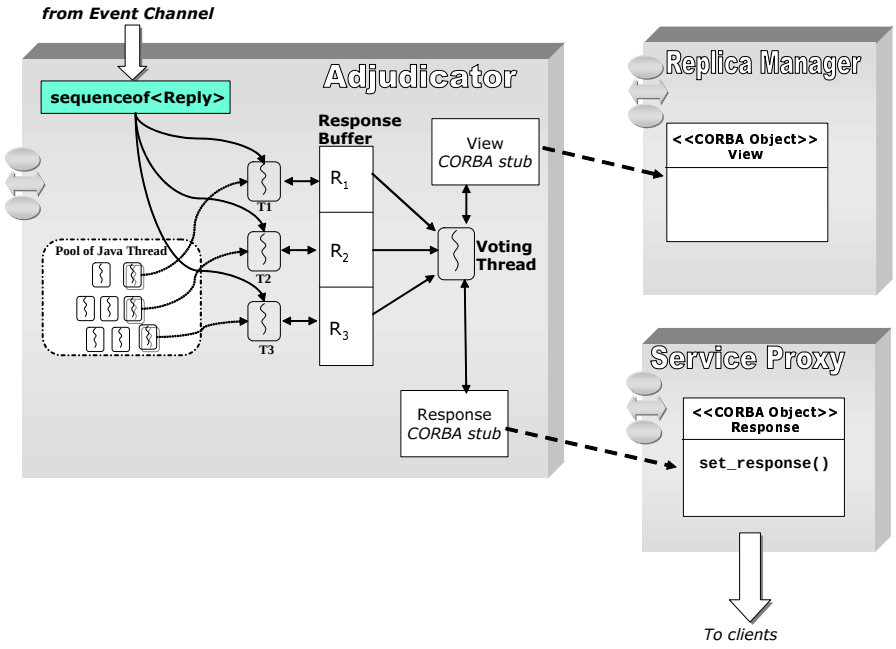

Fig. 6. Replica Manager, Adjudicator, Service Proxy, and Event Channel

Server responses are encapsulated in a reply multi-object, and sent to the Adjudicator component via the ORB. Information about the current state of the system is made available to the Adjudicator via a CORBA stub of the `View` object.

In order to decouple the communication between the `Replica Manager` (`RM`)

and the `Adjudicator` component, the CORBA Notification service is exploited. The RM acts as an event supplier and it adopts a push paradigm, i.e. as soon as the reply is available, it creates an event and sends it to the CORBA Event channel. The Adjudicator acts as an event consumer and it adopts a pull model, i.e. as soon as finished to process the previous reply it retrieves the next one form the CORBA Event channel. Since it is inefficient to transfer events one at a time, the CORBA Notification Service includes support for sequences of structured events. Suppliers may transfer multiple events to a channel in a single CORBA method invocation. Likewise, consumers may receive multiple structured events in a single CORBA method call [26]. We thus used a typed event, consisting of a sequence of replies, to transfer multiple replies in one shot. The length of the sequence depends on the specific values of configuration parameters (these are set by the administrator). If the *Pass First* strategy is selected, the typed event is a sequence of one reply, i.e. a sequence of length 1. In this case, as soon as the `RM` has received the first reply, this is sent to the `Adjudicator`. If the *Threshold-based* strategy is configured, the length of the sequence varies over time, as described in the following. The first sequence has a length of T, where T is the threshold value of the adjudication process. This means that the `RM` waits for the first T replies, and then it sends them to the `Adjudicator` as a single structured event. From this point on, the `RM` sends one reply at time until it receives a `STOP` message from the `Adjudicator`. The `Adjudicator` sends a `STOP` message as soon as it has detected that T replicas have provided the same result. In fact, according to the *Threshold-based* strategy, a reply is sent to clients if at least T replicas provide the same result. Figure 7 illustrates the behavior of the `RM` and of the `Adjudicator` objects with a sequence diagram, where $T = 3$ (a result is produced if at least 3 server replicas agree on results) and $n > 4$

(more than 4 server replicas are running). In the example scenario, the first three replies (R_1, R_2,R_3) are different ($R_1 = R_2$ but $R_1 \neq R_3$) and the fourth reply (R_4) is equal to the first one (R_1).

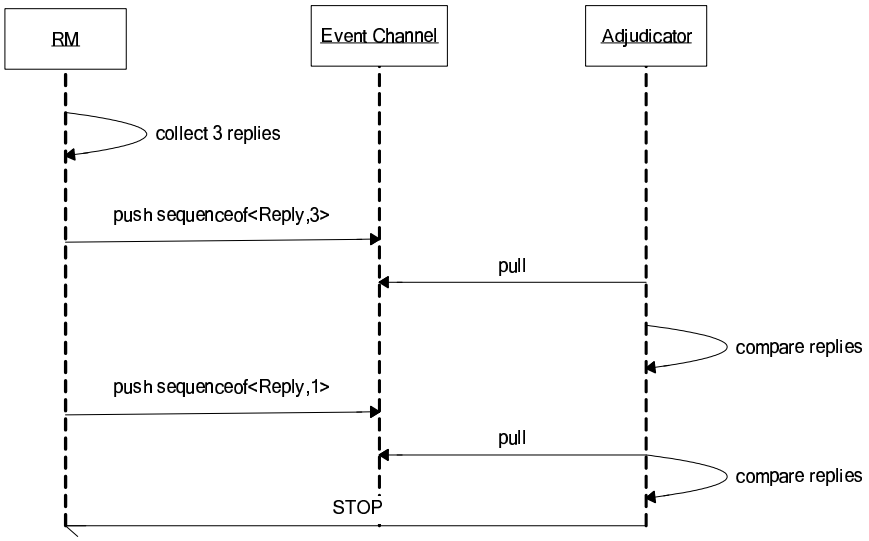

Fig. 7. Communication RM - Adjudicator with Threshold-based strategy

Due to the asynchronous behavior of middle-tier objects, additional replies might be delivered to the Event Channel also after the STOP message has been produced. Such replies would automatically be discarded by the Event Channel. In the following, we provide details about how we configured the Notification Service in order to achieve this behaviour. We also illustrate how we set QoS parameters for such a service. The interested reader can refer to [26] for further details about the configuration of an Event Channel for a CORBA Notification Service. In our system, Event Channel configuration parameter values were as follows:

- Reliability - We set `EventReliability = ConnectionReliability = Persistent`. This means the Event Channel store both connection status and event information persistently. After a component failure (RM or Adjudicator), connections are automatically restored, and events are retrieved from storage

and requeued for delivery;

- Event Sequence - In order to handle the *typed event*, in the form of a sequence, two parameters were set: `MaximumBatchSize` and `PacingInterval`. The former indicates the length of the sequence, i.e. when the sequence reaches `Maximum Batch Size`, even if the time has not reached the `Pacing Interval`, the sequence is sent. The latter is the amount of time for events to be collected into a sequence before delivery;
- Timeout - This property, which indicates the timeout for a consumer waiting for an event, must be set according to the characteristics of the deployment platform and of the legacy application. A value of five seconds was chosen for our prototype.

5.4 Adjudication of results

Another problem we had to face was how to build a single reply to be sent to the client from multiple replies provided by the replicas. This responsibility was assigned to the `Adjudicator` component. More precisely, the `Adjudicator` implements two of the functions described in the conceptual system architecture (in sections 4.5 and 4.3, respectively), namely the `Adjudicator` itself and the `Fault Detector`. These functions were implemented in the same component, because they are tightly coupled to each other. The `Adjudicator` has the capability of detecting Value Faults and of building a (hopefully) correct reply to be sent to the client.

The structure of the `Adjudicator` component was already illustrated in Figure 6, where its interactions with the `Replica Manager` and with the Event Channel were also shown. Here, we limit our attention to the adjudication pro-

cess. As already mentioned, the `Adjudicator` obtains from the CORBA Event Channel a sequence of n responses. The state of the system is known from the Ensemble `View Object`. The voting process is applied to the n responses.

Since it is expensive to create a new thread for each incoming response, a pool of Java threads is created at initialization time (the administrator sets the number of threads which make up the pool [4]). A Java thread is then assigned on the fly to each response.

Figure 6 illustrates the voting process of the `Adjudicator` component in the case of three responses, with three replicas currently active. Three threads from the pool are assigned to processing of the responses. This entails reading data from the `Reply` object and copying it to the `Response Buffer`. As soon as this is done, the component is ready to pull another `Reply` object. The `Voting Thread` is in charge of implementing the adjudication strategy, according to the settings performed by the administrator via the `Service Manager` interface.

The creation of a response involves several middle-tier objects. Upon completion of the voting process, the response can be created. The `ServiceProxy` is in charge of only allocating the response, demanding to the `Adjudicator` object the responsibility of populating it. The voting thread sets the Response via the ORB, calling the `setresponse()` method on the `Reponse` CORBA stub, which acts as a stub for the `Response` CORBA object. A sort of pipeline is so set-up between the `Replica Manager`, which pushes Reply Objects, and the `Service Proxy`, which delivers the `Response` to the client.

[4] This should be done in accordance with the characteristics of the hardware platform. As an example, on systems where memory is scarce, it is better to limit the number of threads in the pool.

6 Conclusions

This work presented a security-oriented maintenance approach, which provides a cost-effective means of leveraging the security level of an existing application. The proposed technique exploits separation of concerns and reuse in a multi-tier architecture. Functional properties are guaranteed by incorporating the existing software modules. Security (namely availability, integrity, and confidentiality) is achieved by means of the business logic of the middle-tier component. Availability and integrity are achieved via replication of the functional modules: by using replication, we are able to deliver system services to authorized users, also in the presence of failures and/or intrusions affecting individual legacy modules. Confidentiality is obtained via cryptography.

The security of the existing application was actually improved, since the overall system is able to tolerate failures and intrusions affecting individual instances of the back-end servers. In particular:

- Availability is improved, in that the overall system fails only when all replicas of the original system have failed;
- Integrity is improved, since messages delivered to the client are successfully modified only if i) a number of them is modified, and ii) this is done in a consistent way (i.e. the contents of individual messages after modification are identical);
- Confidentiality is improved, because communication takes place over protected channels.

This approach can be extremely effective in minimizing the development cost of a secure system, for two fundamental reasons:

(1) Code writing activity is minimized, since system functions are implemented via reuse of existing software;

(2) System testing activity is reduced, since only security-related functions and interactions between individual components need to be tested.

The proposed strategy has been implemented with regard to a case study application, consisting of a client/server program written in C, and deployed over a COTS based heterogeneous platform (commodity PCs, running commercial operating systems, and instrumented with commercial middleware support). The legacy software is integrated in the new system without changing it at all. We provided a thorough treatment of technical challenges we had to face, and a detailed description of the solutions we have implemented. These are briefly summarized in the following:

- *Provision of a secure transport to Middle Tier objects* - In order for middle-tier objects to securely communicate on top of an unsecure network infrastructure, they had to use the facilities provided by the security enhanced transport of the underlying middleware. Instead of explicitly programming calls to security functions in individual objects - which would have resulted in a time-consuming and error-prone activity - we exploited the CORBA Portable Interceptor mechanism to implement a security wrapper, which transparently hooked middle-tier objects to the SSL/TLS transport. Objects of the middle-tier architecture are thus only responsible for properly implementing the business logic of the application. This clean separation of duties has several important advantages. In particular, it favors further system maintenance and evolution. Although this came at a cost in terms of performance penalty, due to additional actions in the invocation path, we believe the advantages related to this clean separation of responsibilities

largely outweighs such a performance penalty. This is especially true if one keeps in mind that research results have shown that a major fraction of the performance penalty induced by interception is related to the registration and initialization phases of the incerceptors, which are executed only once, at startup time;

- *Communication between middle-tier objects and legacy server replicas* - In order to enforce replica consistency, we had to integrate multicast support into the system. We took a driver based approach (i.e., we developed a virtual device driver for the host operating system), which allowed us to intercept messages from individual legacy server replicas and to re-route them to the multicast infrastructure. We provided quantitative results about the performance of our scheme, as compared to plain TCP/IP routing. Results indicated that our communication scheme has an average throughput of 25 Mbit/s against 80 Mbit/s of plain TCP/IP delivery. This is fairly good, if one takes into account that we introduced an additional tier, and an additional protocol stack in the communication chain;
- *Decoupling of communication between middle-tier objects* - In order to decouple the communication between two key components of the middle-tier, namely the `Replica Manager` and the `Adjudicator`, we exploited advanced features of the CORBA Notification Service. Such a Service provides a QoS enabled event delivery infrastructure. We provided details about how we configured the Notification Service to achieve our goals;
- *Adjudication of results* - We provided three alternative adjudication strategies, which are characterized by different trade-offs in terms of security vs. performance. In particular, the *Pass First* strategy strongly penalizes security, but achieves the highest performance possible. In fact, on the one side it allows the middle-tier to get data from the fastest (at any given time)

back-end source, on the other side in the event of a value fault (due to application internal faults or to intrusions), an error is propagated to the client. The *Threshold-based* strategy provides some protection against faults and intrusions in individual replicas, at the cost of some performance penalty. In fact, in order for the system to be able to produce an output, it has to wait at least for T responses (where T is the value chosen for the threshold) from the replicas. On the opposite end of the spectrum is the *All Agree* strategy. This strategy provides the highest level of protection possibile, but has the worst system response time, since no output is produced until all replicas have responded.

References

[1] Institute of Electrical and Electronics Engineers, "IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries". New York, NY: 1990.

[2] Y. Deswarte, K. Kanoun, and J. C. Laprie, "Diversity against Accidental and Deliberate faults", in Computer Security, Dependability, and Assurance, P. Amman, B.H. Barnes, S. Jajodia, E.H. Sibley, eds, IEEE Computer Society Press, 1999

[3] W. Ford, M.S. Baum, "Secure Electronic Commerce", Prentice Hall Inc., Upper Saddle River (1997)

[4] D. Atkins, et al., "Internet Security Professional Reference". 2nd edn. New Riders Publishing, Indianapolis, 1997

[5] `http://www.linuxguruz.org/iptables/`

[6] J. C. Laprie, "Dependable Computing and Fault Tolerance: Concepts and Terminology", in *Proc. of 15th International Symposium on Fault Tolerant Computing*,IEEE Computer Society,, pp. 2-11, Ann Arbor, MI, 1985.

[7] Z.T. Kalbarczyk, S. Bagchi, K. Whisnant, and R.K. Iyer, "Chameleon: A software Infrastructure for Adaptive Fault Tolerance", in *IEEE Transactions on Parallel and Distributed Systems*, vol.10, no.6, June 1999.

[8] L.E. Moser, P.M. Melliar-Smith, P. Narasimhan, L. Tewksbury and V. Kalogeraki, "The Eternal System: An Architecture for Enterprise Applications", in *Proc. of International Enterprise Distributed Object Computing Conference* ,University of Mannheim, Germany (September 1999), pp. 214-222

[9] K.K. Goswami, R.K. Iyer, "Simulation of Software Behavior Under Hardware Faults", in *Proc. of the 23rd Annual International Symposium on Fault-Tolerant Computing*, Toulouse, France, June 1993.

[10] John D. Howard, An Analysis of Security Incidents on the Internet 1989-1995, Apr. 1997, Pittsburgh, Pennsylvania, USA. `http://www.cert.org/research/JHThesis/Start.html`

[11] R. Han, D. Messerschmitt, A progressively reliable transport protocol for interactive wireless multimedia, in Multimedia Systems 7: pp. 141–156, 1999

[12] M. Cukier et al., "AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects", in Proc. of the 17th IEEE Symposium on Reliable Distributed Systems (SRDS'98), West Lafayette, Indiana, USA, October 23, 1998, pp. 245–253.

[13] R. Jiminez-Peris, M. Patino-Martinez, S. Arevalo, "Deterministic scheduling for transactional multithreaded replicas"', in Proc. of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS-2000), pp. 164 -173.

[14] P. Narasimhan, L. E. Moser, P.M. Melliar-Smith, "Replica Consistency of Objects in Partitionable Distributed Systems", in *Distributed Systems Engineering*, vol.4, no.3, September 1997, pp. 139–150.

[15] J.C. Fabre and T. Pèrennou, "A Metaobject Architecture for Fault-Tolerant Distributed Systems: The FRIENDS Approach",in *IEEE Transactions on Computers*, vol. 47, no. 1, January 1998.

[16] P. Felber, R. Guerraoui, A. Schiper, "The Implementation of a Object Group Service", in *Theory and Practice of Object Systems (TAPOS)*,Wiley&Sons, Vol. 4, No. 2, 1998.

[17] Ken Birman, Robert Constable, Mark Hayden, Christopher Kreitz, Ohad Rodeh, Robbert van Renesse, Werner Vogels,The Horus and Ensemble Projects: Accomplishments and Limitations, in Proceedings of the DARPA Information Survivability Conference & Exposition (DISCEX '00), January 25-27 2000 in Hilton Head, South Carolina

[18] D.E. Bakken, Z. Zhan, C.C. Jones, D.A. Karr, Middleware Support for Voting and Data Fusion, in: The 2001 International Conference on Dependable Systems and Networks, IEEE-CS, 2001, pp. 453–462.

[19] Institute for Applied Information Processing and Communications, March 2001 http://jcewww.iaik.tu-graz.ac.at/jce/jce.htm.

[20] IONA Technologies PLC, "Orbix 2000 SSL/TLS Programmer's Guide". December 2000 available on http://www.iona.com.

[21] R. Housley, W. Ford, W. Polk, D. Solo, Internet X.509 Public Key Infrastructure Certificate and CRL Profile, RFC 2459, January 1999.

[22] IONA Technologies PLC, "Orbix 2000 Programmer's Guide Java Edition". December 2000 available on http://www.iona.com.

[23] National Institute of Standards and Technology: Secure Hash Standard, FIPS PUB 180-1, Federal Information Processing Standards Pubblication, 1995 (available on-line at http://www.itl.nist.gov/fipspubs/fips180-1.htm)

[24] R. Rivest, The MD5 Message-Digest Algorithm, RFC 1321, MIT LCS & RSA Data Security Inc., April 1992.

[25] Uresh Vahalia, UNIX Internals, The new frontiers, Prentice Hall International, 1996.

[26] Object Management Group. 2000. "Notification Service Specification". `ftp://ftp.omg.org/pub/docs/formal/00-06-20.pdf`. Framingham, MA: Object Management Group.

[27] ORBacus Notify,`http://www.iona.com/products/orbacus_home.htm`.

[28] Object Management Group (OMG), "CORBA Interceptors Published Draft with CORBA 2.4+ Core" Chapters Document Number ptc/01-03-04 available on `http://www.omg.org`.

[29] Object Management Group (OMG), "CORBA Security Service Specification version 1.7" March 2001 available on `http://www.omg.org`.

[30] R. Zahavi, "Enterprise Application Integration with CORBA", Ed. Willey & Son, Inc, 1999;

[31] Driver Developer Site AnswerBook, Solaris 2.6 Writing Device Driver, SunSoft Inc, Mountain View CA., 1997 (`http://docs.sun.com`).

[32] D. Cotroneo, A. Mazzeo, L.Romano, and S. Russo, "Using CORBA Interceptors to Implement a Security wrapper", in Proc. of International Conference on Advances in Infrastructure for e-Business, e-Education,e-Science, and e-Medicine on the Internet 2002 (SSGRR 2002s), l'Aquila, Italy.