

Security in a Distributed Processing Environment

by

Elizabeth Mary Joyce

B.Sc. (Hons)

A thesis submitted to the University of Plymouth

In partial fulfilment for the degree of

DOCTOR OF PHILOSOPHY

Department of Communication and Electronic Engineering

Faculty of Technology

In collaboration with

IONA Technologies, plc.

& Orange

December 2001

90 0523479 4



UNIVERSITY OF PLYMOUTH	
Item No.	90 0523479 4
Date	18 SEP 2002
Class No.	THESIS 004.35 J04
Cont. No.	X704469781
PLYMOUTH LIBRARY	

LIBRARY STORE

REFERENCE ONLY

Security in a Distributed Processing Environment

Elizabeth Joyce

B.Sc (Hons)

Abstract

Distribution plays a key role in telecommunication and computing systems today. It has become a necessity as a result of deregulation and anti-trust legislation, which has forced businesses to move from centralised, monolithic systems to distributed systems with the separation of applications and provisioning technologies, such as the service and transportation layers in the Internet. The need for reliability and recovery requires systems to use replication and secondary backup systems such as those used in e-commerce.

There are consequences to distribution. It results in systems being implemented in heterogeneous environment; it requires systems to be scalable; it results in some loss of control and so this contributes to the increased security issues that result from distribution. Each of these issues has to be dealt with. A distributed processing environment (DPE) is middleware that allows heterogeneous environments to operate in a homogeneous manner. Scalability can be addressed by using object-oriented technology to distribute functionality. Security is more difficult to address because it requires the creation of a distributed trusted environment.

The problem with security in a DPE currently is that it is treated as an adjunct service, i.e. and after-thought that is the last thing added to the system. As a result, it is not pervasive and therefore is unable to fully support the other DPE services. DPE security needs to provide the five basic security services, authentication, access control, integrity, confidentiality and non-repudiation, in a distributed environment, while ensuring simple and usable administration.

The research, detailed in this thesis, starts by highlighting the inadequacies of the existing DPE and its services. It argues that a new management structure was introduced that provides greater flexibility and configurability, while promoting mechanism and service independence. A new secure interoperability framework was introduced which provides the ability to negotiate common mechanism and service level configurations. New facilities were added to the non-repudiation and audit services.

The research has shown that all services should be security-aware, and therefore would be able to interact with the Enhanced Security Service in order to provide a more secure environment within a DPE. As a proof of concept, the Trader service was selected. Its security limitations were examined, new security behaviour policies proposed and it was then implemented as a Security-aware Trader, which could counteract the existing security limitations.

Acknowledgements

I would like to thank the following:

- Prof. Peter Sanders, my Director of Studies, who provided me with the opportunity to undertake this research and for all his support and advice during my time with the Group;
- Prof. Paul Reynolds, Supervisor, for his advice and insights;
- Dr. Steven Furnell, Supervisor, whose unending help and collaboration I deeply appreciate;
- Everyone in the Network Research Group;
- Declan O'Sullivan, my mentor, and everyone in IONA Technologies PLC, for all their help and collaboration;
- Dr. Joseph Morrissey, for his support and patience when listening to endless hours of rhetorical questions;
- And finally, to my parents and family, who have always been there for me, and provide a constant source of support and encouragement.

Declaration

At no time during the registration for the degree of Doctor of Philosophy has the author been registered for any other university award.

Relevant scientific seminars and conferences were regularly attended at which work was presented; external institutions were visited for consultation purposes, and the papers prepared for publication.

The work presented in this thesis is solely that of the author.

Signed.....*Elizabeth Joyce*.....
Date.....*31/8/02*.....

Glossary of Abbreviations

AA	Audit Agent
ACA	Access Control Agent
ACL	Access Control List
ACM	Access Control Matrix
AKB	Audit Knowledge Base
API	Application Programming Interface
AR	Audit Responder
ASA	Audit Sampling Agent
as-UAP	Access session related User Application
AuthA	Authentication Agent
CA	Certification Authority
CCM	CORBA Component Model
CCS	Comprehensive CORBASec
CISS	Comprehensive Integrated Security System
CL	Capability Lists
CORBA	Common Object Request Broker Architecture
CSI	Common Secure Interoperability
CSM	Communications Session Manager
DBMS	DataBase Management System
DCE	Distributed Computing Environment
DES	Data Encryption Standard
DMA	Domain Mapping Agent

DPE	Distributed Processing Environment
EMERALD	Event Monitoring Enabling Responses to Anomalous Live Detection
GIOP	General Inter-ORB Protocol
GSS-API	Generic Security Service Application Protocol Interface
IA	Initial Agent
IDL	Interface Definition Language
IDS	Intrusion Detection System
IIOP	Internet Inter-ORB Protocol
IN	Intelligent Networks
ISE	Integrated Service Engineering
ISO	International Standards Organisation
KB	Knowledge Base
KTN	Kernel Transport Network
MAC	Message Authentication Code
MD-5	Message Digest 5
MFC	Microsoft Foundation Classes
NCCE	Native Computing and Communications Environment
NIDES	Next-generation Intrusion Detection Expert System
NRAdj	Non-Repudiation Adjudicator
NREA	Non-Repudiation Evidence Agent
NRS	Non-Repudiation Store
ODBC	Open DataBase Connectivity
ODL	Object Definition Language
ODP	Open Distributed Processing
OMG	Object Management Group

OO	Object-Oriented
ORB	Object Request Broker
OSI	Open Systems Interconnection
OTS	Object Transaction Service
PA	Provider Agent
PAC	Privilege Attribute Certificate
PCM	Profile/Context Manager
PCM	Policy/Context Manager
PoC	Proof of Concept
PS	Persistence Service
QoP	Quality of Protection
QoPA	Quality of Protection Agent
RACF	Resource Access Control Facility
RM-ODP	Reference Model for Open Distributed Processing
RSA	Rivest, Shamir, and Adleman
SecA	Security Agent
SECIOP	Secure Inter-Orb Protocol
SF	Service Factory
SHA	Secure Hash Algorithm
SIA	Secure Interoperability Agent
SOAP	Simple Object Access control Protocol
SSC	Security Services Component
SSL	Secure Socket layer
SSM	Service Session Manager
ss-UAP	Service session related User Application

TCB	Trusted Computing Base
TCSM	Terminal Communications Session Manager
TINA	Telecommunications Information Networking Architecture
TMN	Telecommunications Management Networks
TTP	Trusted Third Party
UA	User Agent
URL	Uniform Resource Locator
USC	User Sponsor Code
USM	User Session Manager
WAP	Wireless Application Protocol

TABLE OF CONTENTS

GLOSSARY OF ABBREVIATIONS.....	IV
1. INTRODUCTION.....	1
1.1 SERVICE ENGINEERING AND SECURITY	1
1.2 AIMS AND OBJECTIVES OF THE RESEARCH	3
1.3 THESIS STRUCTURE.....	4
2. INTEGRATED SERVICE ENGINEERING AND DISTRIBUTED PROCESSING ENVIRONMENTS.....	7
2.1 INTRODUCTION.....	7
2.2 INFLUENTIAL TECHNOLOGIES IN ISE	8
2.2.1 <i>Open Distributed Processing</i>	8
2.2.2 <i>Object-Orientation</i>	10
2.3 TELECOMMUNICATIONS INFORMATION NETWORKING ARCHITECTURE.....	12
2.3.1 <i>The Overall Architecture</i>	12
2.3.2 <i>The Service Architecture</i>	15
2.3.3 <i>The Computing Architecture and the DPE</i>	17
2.3.4 <i>The Management Architecture</i>	19
2.4 COMMON OBJECT REQUEST BROKER ARCHITECTURE	21
2.4.1 <i>CORBA Interface Definition Language (IDL)</i>	22
2.4.2 <i>CORBA Object Request Broker (ORB)</i>	23
2.4.3 <i>CORBA services</i>	24
2.5 THE TRADING SERVICE	26
2.5.1 <i>The Trader Data Structures</i>	27
2.5.2 <i>Attributes</i>	28
2.5.3 <i>Interfaces</i>	28
2.5.4 <i>Linked Traders</i>	30
2.5.5 <i>Uses of an Unsecured Trader</i>	32
2.6 THE SECURITY SERVICE	33
2.7 SUMMARY	39
3. LOGICAL SECURITY IN DISTRIBUTED SYSTEMS.....	40
3.1 INTRODUCTION.....	40
3.2 SECURITY PRINCIPLES	41
3.2.1 <i>Access Control Service</i>	41
3.2.2 <i>Authentication Service</i>	43
3.2.3 <i>Confidentiality Service</i>	45
3.2.4 <i>Integrity Service</i>	47
3.2.5 <i>Non-Repudiation and Auditing Service</i>	48
3.2.6 <i>Security Management</i>	51
3.3 OTHER PRINCIPLES RELEVANT TO DPE SECURITY	52
3.3.1 <i>Security Domains and Trust Models</i>	52
3.3.2 <i>Distributed Trusted Computing Base</i>	54
3.3.3 <i>Interoperability</i>	55

3.3.4	<i>Mechanism Independence and the Separation of Mechanism & Service Management</i>	60
3.4	SUMMARY	60
4.	REQUIREMENTS FOR A NEW FRAMEWORK FOR DPE SECURITY	62
4.1	INTRODUCTION.....	62
4.2	REQUIREMENTS FOR DPE SECURITY.....	62
4.2.1	<i>Distributed Object Complications</i>	63
4.2.2	<i>Review of Currently defined GDOS Security Requirements</i>	64
4.2.3	<i>Analysing the DPE Security Problem Domain</i>	71
4.3	FORMULATING A DPE SECURITY FRAMEWORK	74
4.4	A NEW SECURITY FRAMEWORK FOR DPES.....	80
4.4.1	<i>DPE Security Service Overview</i>	80
4.4.2	<i>Realisation and Deployment Issues</i>	84
4.4.3	<i>DPE Security Management Overview</i>	88
4.5	DPE SECURED SERVICE EXAMPLE.....	96
4.5.1	<i>Logging in to the Provider</i>	97
4.5.2	<i>Starting a New Service Session</i>	101
4.6	SUMMARY	107
5.	SECURE INTEROPERABILITY IN A DPE	108
5.1	INTRODUCTION.....	108
5.2	DPE SECURE INTEROPERABILITY REQUIREMENTS	109
5.3	DPE SECURE INTEROPERABILITY – THE ISSUES	112
5.3.1	<i>Conflicting Security Mechanisms</i>	112
5.3.2	<i>Conflicting Security Policies</i>	114
5.3.3	<i>Conflicting Security Protocols</i>	116
5.3.4	<i>Different Trust Domains</i>	116
5.4	A NEW SECURE INTEROPERABILITY FRAMEWORK.....	120
5.4.1	<i>New Policy Configuration Structure</i>	120
5.4.2	<i>New Secure Interoperability Protocol</i>	124
5.4.3	<i>New Secure Interoperability Service Objects</i>	128
5.4.4	<i>Secure Interoperability Example</i>	133
5.5	SUMMARY	140
6.	SECURITY-AWARE DPE SERVICES	142
6.1	INTRODUCTION.....	142
6.2	SECURITY ISSUES FOR SUPPORTING SERVICES IN A DPE.....	143
6.3	SECURITY ISSUES RELATED TO TRADING & TRADERS	146
6.3.1	<i>Authentication</i>	146
6.3.2	<i>Access Control</i>	146
6.3.3	<i>Integrity and Confidentiality</i>	149
6.3.4	<i>Non-Repudiation</i>	150
6.4	CURRENT LIMITATIONS.....	151
6.5	NEW FACILITIES REQUIRED.....	155
6.5.1	<i>Security-Aware Trader Attributes</i>	155
6.5.2	<i>Security-Aware Trader Data Structures</i>	156

6.5.3	<i>Security-Aware Trader Interfaces</i>	158
6.5.4	<i>Security-Aware Trader and the new Framework for DPE Security</i> ..	164
6.5.5	<i>New Facility Summary</i>	166
6.6	OTHER SECURITY-AWARE SERVICES IN A DPE.....	167
6.7	SUMMARY	169
7.	VERIFICATION OF THE NEW FRAMEWORK	171
7.1	INTRODUCTION.....	171
7.2	MAPPING TO CORBASEC	171
7.2.1	<i>CORBASEC vs. DPE Requirements</i>	172
7.2.2	<i>Mapping to the new Comprehensive CORBASEC</i>	176
7.2.3	<i>Management and Mechanism-Independence</i>	177
7.2.4	<i>Authentication & Authorisation Enhancements</i>	180
7.2.5	<i>Integrity & Confidentiality Enhancements</i>	187
7.2.6	<i>Non-Repudiation & Audit Enhancements</i>	191
7.2.7	<i>Secure Interoperability</i>	201
7.2.8	<i>Security-Aware Trader</i>	203
7.3	SUMMARY	204
8.	PROOF OF CONCEPT	206
8.1	INTRODUCTION.....	206
8.2	THE PROOF OF CONCEPT PROTOTYPE.....	207
8.2.1	<i>Implementation of the Prototype</i>	209
8.2.2	<i>A Practical Demonstration Scenario</i>	220
8.2.3	<i>Requirements Matrix</i>	225
8.3	VERIFICATION	228
8.3.1	<i>Performance Modelling</i>	228
8.3.2	<i>Standards Verification</i>	240
8.4	SUMMARY	249
9.	CONCLUSIONS	251
9.1	ACHIEVEMENTS OF THE RESEARCH	251
9.2	LIMITATIONS OF THE RESEARCH	252
9.3	SUGGESTIONS FOR FUTURE WORK	253
9.4	SUMMARY OF RESEARCH CONCLUSIONS.....	255
10.	REFERENCES	257
	APPENDIX A - IDL for Comprehensive CORBASEC	271
	APPENDIX B – IDL for Security-Aware Trader Service	321
	APPENDIX C – IDL for Generic Security Service API	338
	APPENDIX D – Cryptlib and Prototype Information	346
	APPENDIX E – Papers	352
	APPENDIX F – Letters	379

TABLE OF FIGURES

FIGURE 2-1 TINA OVERALL ARCHITECTURE	13
FIGURE 2-2 STRUCTURE OF TINA SYSTEM	14
FIGURE 2-3 TINA SESSIONS	16
FIGURE 2-4 DPE ARCHITECTURE.....	18
FIGURE 2-5 TINA MANAGEMENT ARCHITECTURE	20
FIGURE 2-6 OMG CORBA ARCHITECTURE	22
FIGURE 2-7 CORBA ORB STRUCTURE.....	24
FIGURE 2-8 TRADER INTERACTIONS	26
FIGURE 2-9 TRADER.....	31
FIGURE 2-10 CORBA SECURITY SERVICE.....	34
FIGURE 2-11 CORBA SECURITY OBJECTS	36
FIGURE 3-1 X.509 CERTIFICATION AUTHORITY HIERARCHY STRUCTURE	45
FIGURE 3-2 MONITORING AGENT STRUCTURE.....	49
FIGURE 3-3 INTEROPERABILITY BRIDGING SOLUTIONS	57
FIGURE 4-1 TINA SERVICE EXAMPLE	76
FIGURE 4-2 A NEW SECURITY FRAMEWORK FOR DPEs (OPERATIONAL)	80
FIGURE 4-3 EXAMPLE OF SECURITY SERVICE OBJECT DEPLOYMENT	85
FIGURE 4-4 ADMINISTRATIVE POLICY CLASS	89
FIGURE 4-5 SECURITY SERVICE OBJECTS - MANAGEMENT	91
FIGURE 4-6 NEW SECURE LOGIN EXAMPLES	98
FIGURE 4-7 NEW SECURE SERVICE EXAMPLE.....	104
FIGURE 5-1 SECURE INTEROPERABILITY PROTOCOL MESSAGE SEQUENCE	128
FIGURE 5-2 NEW SECURE INTEROPERABILITY LOGIN EXAMPLE	135
FIGURE 6-1 TRADER ACCESS CONTROL	147
FIGURE 6-2 TRADER SERVICE OFFER ACCESS CONTROL	148
FIGURE 6-3 PROTECTING STORED DATA	149
FIGURE 6-4 SERVICE OFFER ACCESS CONTROL WITH REGISTRY SECURITY PROPERTY	153
FIGURE 6-5 SECURITY-AWARE TRADER'S ADMIN INTERFACE	160
FIGURE 6-6 SECURITY-AWARE TRADER'S LOOKUP INTERFACE	162
FIGURE 6-7 SECURITY-AWARE TRADER	167
FIGURE 7-1 COMPREHENSIVE CORBASEC OBJECTS.....	177
FIGURE 7-2 CCS DOMAINMANAGER	180
FIGURE 7-3 CCS AUTHENTICATION.....	184
FIGURE 7-4 CCS INTEGRITY AND CONFIDENTIALITY.....	190
FIGURE 7-5 CCS NON-REPUDIATION.....	196
FIGURE 7-6 CCS AUDIT	201
FIGURE 7-7 CSI MESSAGE TYPES	202
FIGURE 8-1 SUMMARY OF ISSUES IN RESEARCH.....	209
FIGURE 8-2 OBJECT IMPLEMENTATION	212
FIGURE 8-3 CCS OBJECTS IMPLEMENTED.....	213
FIGURE 8-4 SECURITY-AWARE TRADER INTERFACES IMPLEMENTED	214
FIGURE 8-5 INTERCEPTOR INITIATED CALLS.....	216
FIGURE 8-6 DIRECT CALL ON OBJECTS	217

FIGURE 8-7 AUTHORISED PATHS THROUGH THE DEMO WITH NEW SECURITY SERVICE	223
FIGURE 8-8 ADMINISTRATION SELECTION SCREEN	223
FIGURE 8-9 SECURITY SERVICE ADMINISTRATION SCREENS	224
FIGURE 8-10 AUTHORISED PATHS THROUGH TRADER DEMO WITH NEW SECURITY SERVICE	225
FIGURE 8-11 TRADER SECURITY ADMINISTRATION & QUERY SCREENS	225
FIGURE 8-12 AUTHENTICATION EVENT SEQUENCE CHART	230
FIGURE 8-13 ACCESS CONTROL EVENT SEQUENCE CHART	230
FIGURE 8-14 QOP EVENT SEQUENCE CHART	231
FIGURE 8-15 NON-REPUDIATION EVENT SEQUENCE CHART	232
FIGURE 8-16 AUDIT EVENT SEQUENCE CHART	233
FIGURE 8-17 SECURE INVOCATION EVENT SEQUENCE CHART	234
FIGURE 8-18 OPERATIONAL OBJECT INVOCATION COMPARISON	235
FIGURE 8-19 NUMBER OF ADMINISTRATION OBJECT METHODS	238

TABLE OF TABLES

TABLE 2-1 CORBASERVICES	25
TABLE 4-1 GDOS SECURITY REQUIREMENTS	70
TABLE 4-2 SUMMARY OF TINA VS. DPE SECURITY REQUIREMENTS	79
TABLE 5-1 REQUIREMENTS FOR DPE SECURITY	110
TABLE 5-2 ADDRESSING SECURE INTEROPERABILITY SCENARIOS	119
TABLE 5-3 POLICY CONFIGURATIONS	123
TABLE 5-4 SECUREINTEROPERABILITYPOLICY STRUCTURE	129
TABLE 5-5 SECUREINTEROPERABILITYPOLICY STRUCTURE	132
TABLE 5-6 USER/PROVIDER SECUREINTEROPERABILITYPOLICIES	134
TABLE 5-7 ATTRIBUTE AND ROLE MAPPINGS	134
TABLE 6-1 TRADER SECURITY POLICIES	155
TABLE 6-2 SECURITY-AWARE TRADER'S SERVICE TYPE REPOSITORY EXAMPLE	157
TABLE 6-3 SECURITY-AWARE TRADER'S REGISTRY ENTRY EXAMPLE	158
TABLE 7-1 DPE SECURITY REQUIREMENTS AVAILABLE IN CORBASEC	173
TABLE 7-2 ADMINISTRATION OBJECTS	178
TABLE 7-3 AUTHENTICATION & AUTHORISATION SECURITY SERVICE OBJECT MAPPINGS TO CCS	184
TABLE 7-4 QOP SECURITY SERVICE OBJECT MAPPINGS TO CCS	189
TABLE 7-5 CCS NON-REPUDIATION MAPPINGS	194
TABLE 7-6 SECURE INTEROPERABILITY SERVICE OBJECT MAPPINGS TO CCS	203
TABLE 8-1 OPERATIONAL OBJECT INVOCATION COMPARISON	235
TABLE 8-2 ADMINISTRATION OBJECTS	236
TABLE 8-3 COMPARISON OF THE NUMBERS OF ADMINISTRATION OBJECT MODELS	237
TABLE 8-4 SECURITY-AWARE TRADER'S SECURITY OBJECT INVOCATIONS	239

1. Introduction

1.1 *Service Engineering and Security*

Computers are pervasive throughout the telecommunications industry. They are utilised by the core infrastructure, e.g. in switches, by the software applications operating and controlling the infrastructure, e.g. Intelligent Networks combine these technologies to create a means of separating switching and logic functions in order to build a more flexible distributed architecture for service provisioning. The Internet provides another illustration of how distributed computer systems are combined with an underlying telecommunication network.

Object-Oriented technologies are also playing a key role in integrating heterogeneous systems across the globe. E-commerce companies use it to wrap legacy applications and make them available to an Internet audience. Telecommunication companies use distributed object systems to build their Telecommunication Management Networks (TMN), and so manage their vast telephone networks. Such systems need to be supported. This is where the concept of Integrated Service Engineering (ISE) emerges. ISE supports the development, deployment and provisioning of services. It is accomplished through the use of a Service Machine, a key component of which is the Distributed Processing Environment (DPE). The DPE provides an object bus and a set of supporting services, which allow distributed objects to be created, activated, operated and destroyed in a stable and consistent environment. Future modifications and technology innovations may make it even more difficult to distinguish between computer and telecommunication technologies. Therefore ISE, which acts as the

standard providing all these services. needs to cope with the new demands of this environment.

On May 4, 2000, the “ILOVEYOU” worm, also known as the “Love Bug”, bombarded email systems around the world [1]. Users received an email asking them to check the attached “Love Letter”. The attachment was a script that contained the payload. If the attachment was opened the computer was infected. The “Love Bug” changed registry settings so that it would be run every time the computer was rebooted and sent copies of itself to everyone listed in the user’s address book. It also destroyed multimedia files, such as JPEGs and MP3s. It is estimated that over two-thirds of the Fortune 500 Companies were affected at a cost of \$6.7 billion [2, 3]. Although the “Love Bug” was a computer worm, it required the underlying telecommunication network to allow the worm infect on a global scale.

Security has always been an issue. It has been used by governments and private individuals to protect resources they deemed valuable and therefore at risk. Cryptography, the science of hiding information from unwanted eavesdroppers, has a long history [4]. While it was realised that security was required in telecommunications and computing when they were two distinct technologies, distributed systems suffer from a new set of security problems. The system itself is distributed and therefore is not necessarily under the complete control of the users. For example, if you are sending an email, it may pass over several insecure networks before reaching its destination. The distribution also results in increased access to the system, i.e. it provides more points of vulnerability for attack. While security on this new media was not originally a primary concern, viruses such as the “Love Bug” have heightened security awareness. Businesses, governments and individuals are now

realising that they are at risk and must protect themselves and their assets in this technology arena.

Computer telecommunications are subject to serious threats. These threats can happen within any part of society – civilian or government, and can have far reaching even global consequences. ISE is a key component, as it facilitates the provisioning of services in this distributed environment. By its very nature, this environment is more vulnerable and security is seen as the single most important design criteria in many systems today [5]. Therefore ISE must deal with security and all the problems it presents.

1.2 Aims and Objectives of the Research

The aim of this research is to investigate and facilitate the security of DPEs. The research recognises the importance of security in distributed systems in e-commerce and telecommunications environments. The study has five objectives.

1. **Understand the DPE and its security requirements:** The study needs to understand DPEs, define the security requirements of distributed systems and identify any requirements that are particular to the ISE environment. Through analysis of the State of the Art in ISE it should be possible to identify those areas of the requirements that need to be addressed using current and novel security techniques.
2. **Define a framework for DPE security:** The research needs to define a new security framework for DPEs. This framework needs to address all the of the requirements defined in objective 1.

3. **Assess how DPE security is maintained across a heterogeneous environment:** The framework needs to ensure that it preserves security in a fully distributed heterogeneous environment. It must be able to work on and across multiple hardware platforms. It needs to be able to interoperate across multiple security domains, where different security policies and mechanisms are in operation.
4. **Assess the impact of DPE security across all services:** DPEs also provide a set of distributed services to support distributed objects. The research will assess how secure these services are and whether the new Security Framework can adequately protect them.
5. **Assess practical implementation and verification of DPE security framework:** The research will be verified by mapping it to a DPE specification and then implementing the Security Framework. verification will be based on this implementation.

1.3 Thesis Structure

The thesis has been structured so that most of the background information (mainly the state of the art survey work) is confined to the initial chapters.

Chapter 1 – Introduction – This provides an introduction to the research project objectives and how they were accomplished.

Chapter 2 – Integrated Service Engineering and Distributed Processing Environments – This chapter discusses the general principles applied in ISE Service Machines and their key component, the DPE. Current DPE architectures are

described, including a detailed description of one service, the Trading service, which is used in a later chapter to identify service-related issues.

Chapter 3 – Logical Security in Distributed Systems – The general principles of security are discussed, along with security issues that are specific to distributed systems.

Chapter 4 – Requirements for a new Framework for DPE Security – The requirements of DPE security are analysed and, as a result, a new set of DPE security requirements are defined. The current problems in DPE security are then identified, and this directed the recognition of the need for a new security framework, which is then presented to address these issues.

Chapter 5 – Secure Interoperability in a DPE – The Secure Interoperability Service is defined in this chapter. Although it is a key component of the new Security Framework, the substantial work involved in designing this service requires a separate chapter to fully consider the new features.

Chapter 6 – Security-Aware DPE Services – This chapter investigates how the DPE security service interacts with other DPE services to see if there are any security issues. The Trading service was selected for a detailed analysis of the topic. On finding numerous security problems, a new Security-Aware Trader is then proposed and defined to overcome the existing vulnerabilities.

Chapter 7 – Verification of the New Framework – The research provides a proof of concept by mapping the new Security Framework to a particular DPE specification, namely the Object Management Group's (OMG) Common Object Request Broker

Architecture (CORBA). This chapter describes how this was achieved, the issues that were discovered and how they were addressed.

Chapter 8 – DPE Security Prototype – This chapter describes the implementation of a prototype of the proof of concept defined above, in order to prove that it is viable in practice. Although an implementation proves that the research can be constructed, other verification work is required to ensure that it is feasible in a real-world scenario. This chapter provides performance-modelling data and evaluates the future trends of DPEs, indicating where this research can play a part.

Chapter 9 – Conclusion – The final chapter assesses the research and whether the objectives were successfully met. It defines future work in the DPE security arena that should be considered.

A number of appendices are also included, which provide a range of supporting materials, including published papers.

2. Integrated Service Engineering and Distributed Processing Environments

2.1 Introduction

Integrated Service Engineering (ISE) considers the problem of service development, deployment and provision in today's distributed heterogeneous telecommunications environment. A service machine is the technology, both hardware and software, used in provisioning and deploying these services. A key component of the ISE service machine is the Distributed Processing Environment (DPE), which helps support the lifecycle of these objects and allows them to inter-operate across heterogeneous operating systems, networks, languages, applications, tools, and multi-vendor hardware [6].

ISE initially began in the realms of the telecommunications world, but with the emergence of computing technologies such as integrated circuits in the 1960's, the telecommunication providers began to realise they could harness the technology to enhance their own networks and services. The main influences to this work were Intelligent Networks (IN) and Open Distributed Processing (ODP). By using both of these technologies, telecommunication providers could increase their services and fully utilise the existing infrastructure resources. Another influential computing technology was object-orientation. This was seen as another very useful technology in the telecommunications environment. With deregulation impending, the network

Chapter 2: Integrated Service Engineering and Distributed Processing Environments

providers would be forced into interoperability and would have to be able to provide new services quickly and efficiently if they were to remain competitive.

ISE did not remain solely in the telecommunication sphere. The Internet, which utilises existing telecommunications networks, is a strong user of ISE standards. It too is a distributed system that requires flexible implementation independent provisioning of services, even though the services are of a different nature to the ISE originators (e.g. e-commerce). Many businesses are interested in ISE because it promotes heterogeneous interoperability, and so allows them to take advantage of Internet technologies to access their legacy systems. This chapter will now look at ISE DPE and its supporting principles.

2.2 Influential Technologies in ISE

The areas that most significantly influenced ISE were ODP and object-orientation, the relevant principles of which are examined in the following sub-sections.

2.2.1 Open Distributed Processing

All distributed processing, be it object-oriented or not, is based on the work of the International Standards Organisation (ISO). The following is a definition of a distributed processing “ideal”:

“Within a permissible domain of interest, anyone should be able to access and use any resource at any location and at any time, with only the desired knowledge of the underlying infrastructure, and with a response time acceptable for the required purpose” [7].

Chapter 2: Integrated Service Engineering and Distributed Processing Environments

The ISO works on standardisation in Open Distributed Processing (ODP). It has developed a framework (or reference model) called the Basic Reference Model of ODP (RM-ODP) [8]. It specifies an architecture that integrates support for distribution, interoperability and portability. Fundamental to the RM-ODP is the notion that distributed processing systems can be studied and described from several viewpoints. Each viewpoint represents a different abstraction of a distributed system [9]. The viewpoints are as follows [10]:

- **Enterprise:** directed to the needs of system users, it provides a view of how the information system is placed and used within an enterprise;
- **Information:** directed to the needs of information managers, engineers and analysts, it provides an information model with a view covering information sources and sinks, and the flows between them;
- **Computational:** directed to the needs of application designers, it provides a view on how information processing facilities, functionally or logically, perform the information processing tasks;
- **Engineering:** directed to the needs of system and communication designers, it provides a view of the distributed mechanisms and the various transparencies needed to support distribution;
- **Technology:** directed to the needs of programmers, system maintainers and system managers, it provides a view of the components and links that are used to build a distributed system.

Chapter 2: Integrated Service Engineering and Distributed Processing Environments

Distributed systems are capable of offering substantial benefits to their users. The key characteristics have been identified as follows [11]:

- **Resource sharing**, which may relate to items of data, software components (this includes distributed objects) or hardware components.
- **Openness** is the requirement for the availability of well-defined interfaces to resource managers.
- **Concurrency** brings the benefit of higher performance.
- **Scalability** has been a dominant concern in distributed systems. The replication of data and the distribution of load between servers are the key techniques that are used to address it.
- **Fault tolerance** can be addressed more efficiently in distributed systems than in more centralised system architectures, e.g. hardware redundancy and recovery from hardware and software failures.
- **Transparency** addresses the need of users and application programmers to perceive a collection of networked computers as an integrated system hiding the distributed nature of the resources used to perform the user's task.

2.2.2 Object-Oriented

Another key area for ISE, which has influences in both IN and ODP, is Object-Oriented (OO). OO is the organisation of software as a collection of discrete objects that incorporate data structure and behaviour. OO supporters believe that it promotes future reuse and reduces errors and maintenance [12]. The distributed

Chapter 2: Integrated Service Engineering and Distributed Processing Environments

processing community has also adopted it because it hides implementation details – an important issue in a distributed environment.

The key OO principles are as follows [11]:

- **Object:** a piece of code that owns Attributes (data values) and provides services through Methods (also called functions or operations).
- **Classes:** a collection of like objects make up a class (sometimes called a type). A class acts as a template that describes the behaviour of a set of objects. Therefore objects are actually run-time instances of a class.
- **Encapsulation:** hides the internal implementation details of an object from other objects. An object can publish a public interface that defines how other objects can interact with it, while still keeping the implementation private.
- **Polymorphism:** allows the same method to do different things. Depending on the type of object, the method will produce a different effect/action.
- **Inheritance:** allows a new child class to be created from an existing class. The subclass or derived class inherits the methods and data structure of its parent class, and can then add its own methods and data structures, without affecting the parent. This promotes savings in code and simplifies the overall understanding required within a system.

2.3 Telecommunications Information Networking Architecture

The Telecommunications Information Networking Architecture Consortium (TINA-C) is a consortium of about 40 communications companies, computer and network equipment vendors. TINA-C defined the de facto standard Telecommunication Information Networking Architecture (TINA), based on Bellcore's original INA, which hoped to guarantee interoperability between information networks designed using the architecture by defining a set of principles and concepts for the specification, design, implementation, deployment, execution, and operation of software for telecommunication systems. Telecom systems are complex; TINA breaks them down into manageable units through logical/functional partitions and separations [13].

TINA has a business model [14], which describes the stakeholders and how they interact in the TINA environment. Consumers buy services from Retailers. However, the service is actually provided by Third Party Service Providers, while connectivity streams are supplied by Connectivity Providers. Brokers act like a telephone directory, and allow stakeholders to obtain references to other providers.

The following sub-sections will outline the TINA overall architecture, and its relevant constituents.

2.3.1 The Overall Architecture

The overall architecture was defined as follows (and is depicted in figure 2-1 below [15]):

Chapter 2: Integrated Service Engineering and Distributed Processing Environments

- **Service Architecture:** design, specification, implementation and management of services;
- **Network Architecture:** design, specification, implementation and management of the transport network;
- **Management Architecture:** design, specification and implementation of the software systems to manage services and resources;
- **Computing Architecture:** design, build and distribute software and the supporting software environment.

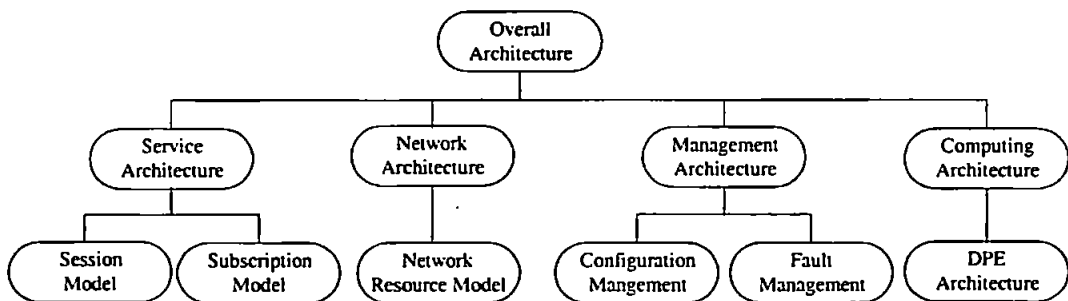


Figure 2-1 TINA Overall Architecture

The 'Basic Separation Architecture' [16] is one of the key principles in TINA-C. It states that there are computing separations between different layers of software. The architecture is made up of a collection of interconnected computing nodes (see figure 2-2 below). The lowest level of a node is the hardware. Above this the Native Computing and Communications Environment (NCCE) is found. This is made up of the operating systems for the local hardware. The NCCE provides a type of

Chapter 2: Integrated Service Engineering and Distributed Processing Environments

transparency, as the Distributed Processing Environment (DPE) is unaware of the hardware and operating systems used. The DPE is sub-divided into the DPE Bottom and the DPE surface. The DPE Bottom offers services such as trading which are available on every node, while the DPE surface offers other services to all nodes but they will only be resident on certain nodes. The complete DPE handles distributed processing and provides transparency between the nodes and the telecommunication applications, which exist on the highest level.

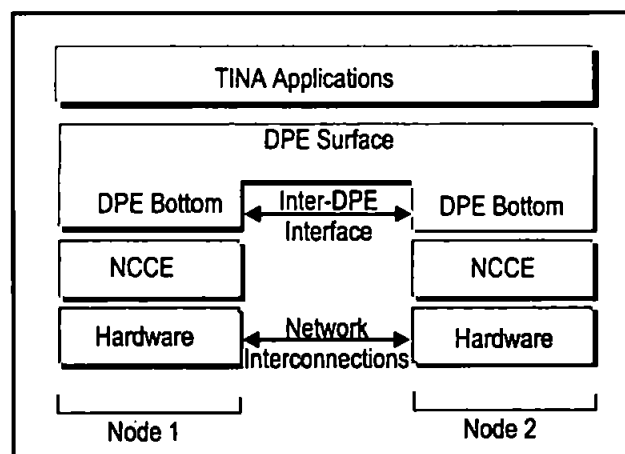


Figure 2-2 Structure of TINA system

TINA was structured in this way to provide true independence (i.e. technology independence and portability) as it states that non-TINA DPEs can be part of the system. This also implies that federation with the non-TINA systems should be possible.

Chapter 2: Integrated Service Engineering and Distributed Processing Environments

Within the TINA application layer of this architecture there is further layering, based on the Telecommunication Management Network (TMN) [17, 18] layers. They are defined as follows:

- **Element Layer:** populated by objects that represent atomic units of physical or logical resources, defined for allocation control, usage and management purposes;
- **Resource Layer:** populated by objects that maintain, view and manipulate collections of elements and their relationships (it provides the service layer with an abstracted view of the elements);
- **Service Layer:** populated by objects involved in provision of services to stakeholders; objects can be service-specific or service-independent.

From this Overall Architecture, the Networking Architecture is considered outside the scope of this research and so will not be presented in any further detail. The Service, Computing and Management Architectures all have some relevance to the work and are now considered in more detail.

2.3.2 The Service Architecture

The traditional concept of a call in telecommunications is substituted by the more flexible concept of a session. A session represents the information used by all processes involved in the provision of a service [19]. For example, in a videoconference the information about connections, charging and user profiles may change during the conference as participants join and leave. The session helps keep

such information coherent throughout the conference. Sessions are not just for complex services, and can represent something as simple as a web-search. The session can be further refined into access, service usage and communications separations, see figure 2-3 [15] below.

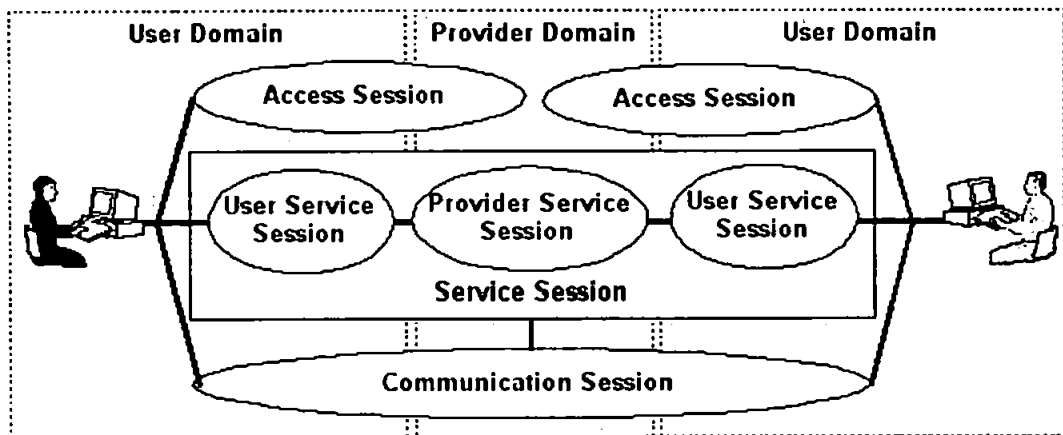


Figure 2-3 TINA Sessions

Before being able to participate in a session, each user must establish an *access session* with the provider; this is comparable to a login session on a multi-user computer. The access session corresponds to the establishment of the terms and conditions of the session. It allows the user to start, combine and participate in several sessions, if authorised to do so. The *service session* corresponds to the provision of the service itself and ensures overall coherence of control and management. It is divided into the User Service Session, which manages the state of each user's activity and resource attributes (e.g. charging context), and the Provider Service Session, which contains the service logic and offers the functions allowing the user to join a session, or be invited to session. The service session contains only one provider but

can have multiple users. The *communication session* provides an abstract view of the actual transport network connections.

TINA also uses the concept of domains [20]. One type of domain is an administrative domain where all the objects in the TINA system are under the ownership of a single stakeholder. A simple example of this is illustrated in figure 2-3 above, where the user and provider domains are depicted.

2.3.3 The Computing Architecture and the DPE

The Computing Architecture adopted the basic concepts of RM-ODP. It uses viewpoints to model complex systems (see section 2.2.1). One such viewpoint is Engineering, which describes the framework for deploying applications and describing the DPE.

The DPE Architecture consists of the DPE Kernel, the Kernel Transport Network and the DPE services, as illustrated in figure 2-4 below [14]. The DPE Kernel provides support to object life-cycle control, i.e. creation/deletion of objects at run time, and inter-object communication, which provides mechanisms to support the invocation of operations provided by operational interfaces of objects. The Kernel provides the basic, technology-independent, functions that represent the capability of most computing systems (i.e. the ability to run programs and the ability of programs to communicate with each other). The DPE Kernel is assumed to be present on all nodes that contain a DPE.

The Kernel Transport Network (kTN) facilitates communications between remote objects, i.e. DPE kernels on different nodes. The kTN provides a technology

independent view of the communication facilities provided by the NCCEs of the DPE nodes. It is a virtual network that is logically different from the transport network.

TINA differentiates between the DPE Kernel and DPE Services. The DPE Kernel provides a basic set of capabilities that are expected on all nodes, while DPE services are considered more advanced capabilities that may not be present on all nodes. The DPE services provide operational interfaces to support the runtime execution and communication of objects.

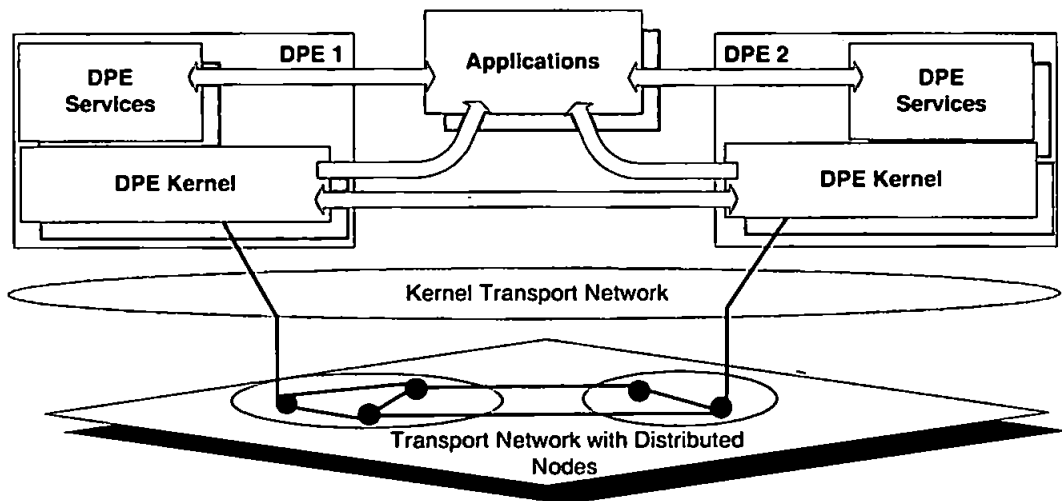


Figure 2-4 DPE Architecture

A subset of the DPE services are listed below [21]:

- **Trading:** provides binding between objects that use a service and objects that provide the service;
- **Notification:** enables objects to receive notifications without being aware of the set of recipient objects;

Chapter 2: Integrated Service Engineering and Distributed Processing Environments

- **Transaction:** consists of three main management functions – transaction, concurrency control and deadlock management;
- **Security:** authentication, authorisation and security controlling;
- **Object Lifecycle:** object creation, deletion, activation, deactivation and move.

The Computing architecture also defines the TINA Object Definition Language (ODL) [22]. TINA-ODL is used to define objects and their interfaces, and supports streams or asynchronous messaging.

2.3.4 The Management Architecture

The TINA Management Architecture (depicted in figure 2-5 below [14]) is a set of concepts and principles used to build and manage systems that will manage TINA systems. The architecture can be divided into two forms of management, Computing and Telecommunications. However, before looking at these, some generic management principles within TINA will be stated. Firstly, management can be functionally separated using the Open System Interconnection's (OSI) system management FCAPS, i.e. Fault, Configuration, Accounting, Performance, and Security [23]. Secondly, management systems are modelled so that management operations and relations can be defined. Managed entities are represented as objects and provide operational interfaces to allow managing objects to manipulate them.

Computing Management involves the management of computers (NCCE), DPE and of the software that runs on the DPE. Software management (i.e. deployment, installation and operation of software computing nodes) and Infrastructure

Chapter 2: Integrated Service Engineering and Distributed Processing Environments

management (i.e. how to manage NCCEs, DPEs, and kTN) are the main concerns of this type of management.

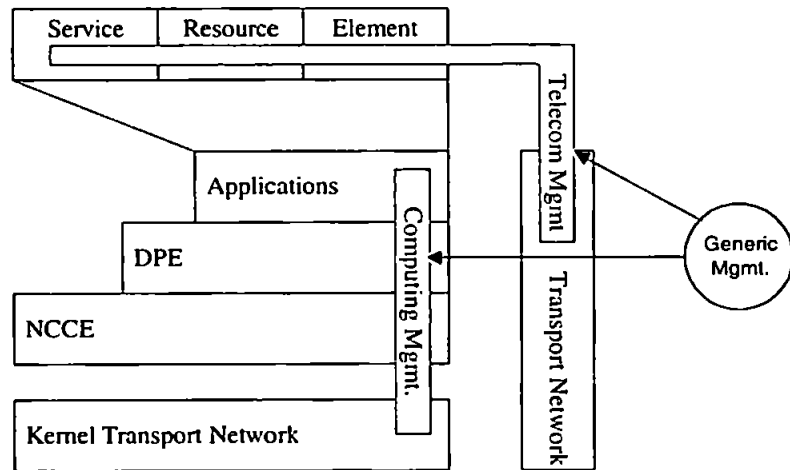


Figure 2-5 TINA Management Architecture

Telecommunication Management involves the management of the transport network and the management of the applications that use and control this network and the management services. Therefore telecommunication management deals with both the service and network architectures.

TINA has been adopted by the Object Management Group (OMG) as the basis for its Common Object Request Broker Architecture, which has many commercial implementations available.

2.4 Common Object Request Broker Architecture

The TINA architecture has been adopted, in particular by the OMG. It defines a middleware standard - Common Object Request Broker Architecture (CORBA), which adheres to both of the previously mentioned Open Distributed Processing (ODP) standards (see section 2.2.1) and object-orientation (OO) standards (see section 2.2.2).

CORBA (see figure 2-6 below [24]) currently consists of an ORB and 15 CORBAServices (see Section 2.4.3 below for CORBAServices overview) [24]. Its function is to allow objects, which are implemented across a heterogeneous and distributed platform, to communicate. The ORB is an object bus, which allows objects to transparently make/receive requests to/from other objects, whether they are local or remote. The CORBAServices are a collection of system-level services that compliment the ORB by providing a robust environment and extending a distributed object's behaviour, i.e. all the basic services an object will need during its lifecycle such as security and persistence. CORBA was designed to allow intelligent objects to discover each other and inter-operate on an object bus. In addition, CORBAfacilities are specified. They are classed as either horizontal or vertical. Horizontal facilities apply to all application domains and there are currently only four defined - printing facility, secure time service, internationalisation service and mobile agents facility. Vertical or Domain facilities relate to particular application fields; they are defined as collections of IDL-defined frameworks that provide services, which applications can use directly. There are currently nine domains working on defining industry

Chapter 2: Integrated Service Engineering and Distributed Processing Environments

appropriate IDL, e.g. Healthcare, Financial, Insurance, Telecommunications, Utilities, Electronic Commerce, Manufacturing, Transportation and Life Science Research. .

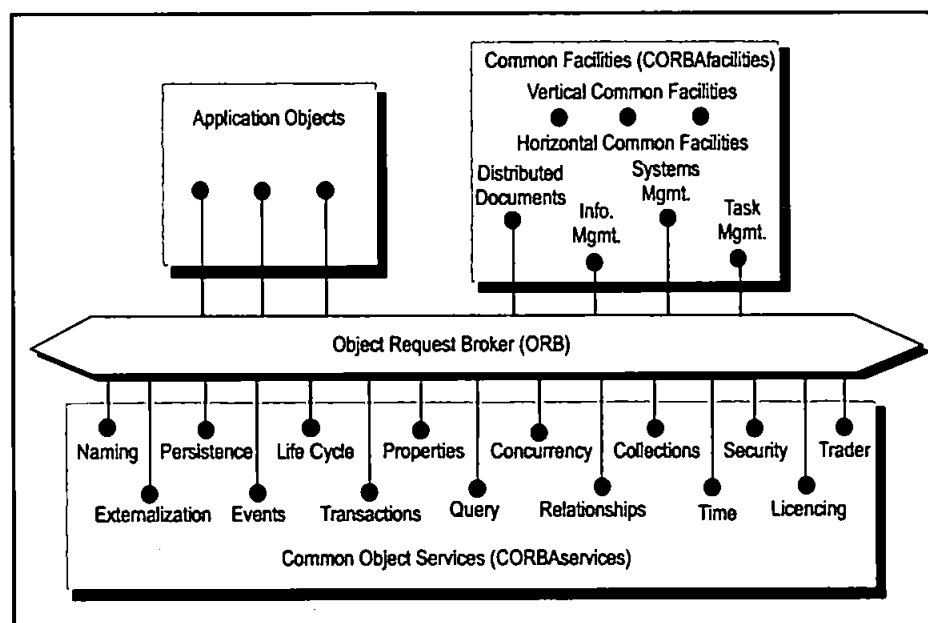


Figure 2-6 OMG CORBA Architecture

2.4.1 CORBA Interface Definition Language (IDL)

Distributed objects are accessed through their interfaces. So, in order to provide flexibility, interfaces are defined not in code but in an Interface Definition Language (IDL). This means that the interface is now accessible across different languages, tools, and operating systems. The IDL defines the operations a distributed object can perform, the parameters required and any exceptions that may be generated in the process.

Although IDL appears to be a subset of the C++ language, it is not a programming language. It is used to specify the contract that exists between the client and server.

Some additional keywords have been added to deal with distribution issues. It is currently mapped to several languages, e.g. C, C++ [25], Java [26], Ada, Smalltalk and COBOL. Programmers are able to deal with CORBA objects using their native language constructs. Since the IDL provides implementation-independent access to objects in the ORB, client and server objects that are written in different languages are able to inter-operate. Therefore IDL provides the basis for interoperability and transparency.

2.4.2 CORBA Object Request Broker (ORB)

The ORB is the middleware that allows clients and servers to communicate. It allows clients to transparently invoke a server method, while the client is unaware of where the server is located or how it is implemented. Figure 2-7 [24] below illustrates the CORBA ORB structure.

On the client side, the ORB intercepts a client call and then finds an object to implement the request. It passes the parameters, invokes the service and then returns the results. The client IDL stubs provide static interfaces to objects, by defining how clients invoke corresponding services on servers. The stub acts as a local proxy for a remote server object. The server operations are defined in IDL and the stubs are generated by an IDL compiler, and include any marshalling¹ code required.

¹ Marshalling is the conversion from one data representation type to another in communication software and is a key component in distributed applications.

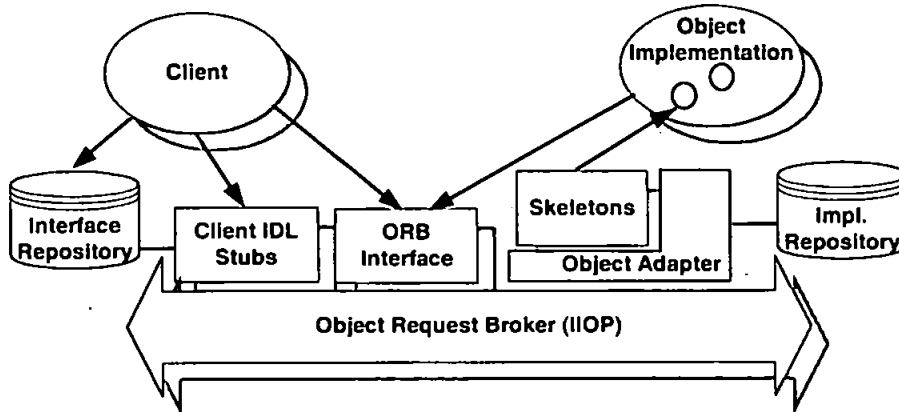


Figure 2-7 CORBA ORB Structure

On the server-side, the ORB locates the server object adapter, gives it the parameters, and then gives control to the object implementation via the server IDL skeleton. The server IDL skeletons are generated by an IDL-compiler. They provide static interfaces to each exported server.

The Object Adapter accepts requests for services on behalf of the server's objects. It provides a run-time environment for instantiating server objects, passing requests and assigning object references to server objects.

The Implementation Repository (also known as the Server Repository) holds information on the classes that servers support and their corresponding runtime objects and object references.

2.4.3 CORBAServices

The CORBAServices are a set of system level services that are used to extend the ORB functionality. Currently 15 such services are defined, as listed in Table 2-1

Chapter 2: Integrated Service Engineering and Distributed Processing Environments

below. Every service can be accessed by every client (if security allows) and basic CORBA services are used by both applications and CORBA facilities [27].

No.	CORBA Service	Function
1.	Life Cycle Service	Creates, copies, moves and deletes objects on the ORB.
2.	Persistence Service (PS)	Stores components persistently on a variety of storage servers.
3.	Naming Service	Locates components by name (i.e. provides clients with an object reference to a server).
4.	Event Service	Register/Unregister interest in specific events ² - basic publish and subscribe messaging service.
5.	Concurrency Control Service	Lock manager working for threads or transactions.
6.	Object Transaction Service (OTS)	Two-phase commit co-ordination among recoverable components using flat or nested transactions.
7.	Relationship Service	Creates dynamic links between objects, and mechanisms for traversing the links that group objects together.
8.	Externalisation Service	Stream-like mechanism used to get data into and out of objects.
9.	Query Service	Query operations for objects (superset of SQL).
10.	Licensing Service	Meters the use of objects for licensing purposes.
11.	Properties Services	Associates properties (named values) with objects.
12.	Time Service	Synchronises time in a distributed environment.
13.	Security Service	Framework for distributed object security.
14.	Trading Service ³	Advertises object services; similar to the naming service, it is used by clients to find server object references.
15.	Collection Service	Manipulates objects in a group as opposed to manipulating them individually (e.g. queues, stacks, lists, etc.).

Table 2-1 CORBA Services

² An **Event** is an occurrence within an object specified to be of interest to one or more objects, e.g. when security administrator objects register interest in when the security alarm object is set to "alarm-raised".

³ The Trading Service is selected as the example service for the research and will be studied in more detail.

2.5 The Trading Service

The Trader is often described as the DPE's Yellow Pages. If a client is looking for a service, but does not have a name of the service provider, the client can go the Trader and ask for the names of all the service providers of the required service. Traders have an important role to play in future Internet and telecommunication networks. The interest in security in web-based [28] and other distributed systems [29, 30] means that Traders will have to incorporate security if they are to be included in this future.

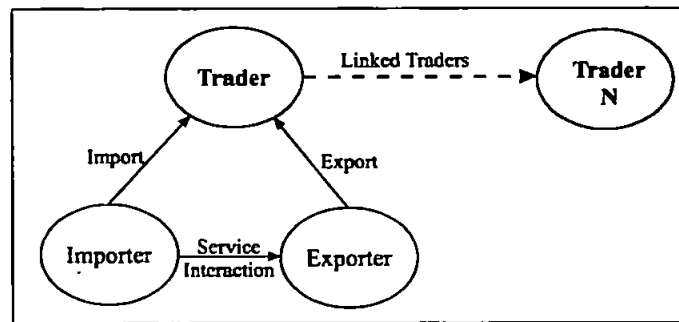


Figure 2-8 Trader interactions

Trading is the process of matching a service request, against a list of supported services provided by potential servers, as illustrated in figure 2-8 above [31]. The basic function of the Trading services involves an exporter (i.e. a server) advertising its available services, by notifying the Trader. The Trader keeps a Registry of such advertisements. An importer (i.e. a client) makes a request on the Trader for a particular service, specifying any conditions that need to be met. The Trader checks its Registry to find a matching service type, with corresponding conditions. The Trader then notifies the importer of the exporter and the service.

2.5.1 The Trader Data Structures

The Trader uses two data structures the Repository and the Registry. The Repository (or Service Type Repository) holds details of service types. This generally consists of the interfaces to a service and a set of properties that would describe the service. For example, if the service were a data store, the service description would hold details such as the type of data store, e.g. file server or database, the location of the store, amount of space available, whether it supports backup or replication, etc. A property can also specify its mode. The property mode attributes have the following connotations:

- **mandatory** - an instance of this service type *must* provide an appropriate value for this property when exporting its service offer.
- **readonly** - if an instance of this service type provides an appropriate value for this property when exporting its service offer, the value for this property may not be changed.

If a property is defined without any mode, it is defined as being “optional” (i.e., an offer of that service type is not required to provide a value for that property name, but if it does, it must be of the type specified in the service type), and the property value subsequently may be modified. The “mandatory” mode indicates that a value must be provided, but that subsequently it may be modified. The “readonly” mode indicates that the property is optional, but that once given a value, subsequently it may not be modified. Specifying both modes indicates that a value must be provided and that subsequently it may not be modified.

Chapter 2: Integrated Service Engineering and Distributed Processing Environments

These service details are static details, but a Trader can also hold dynamic properties. Dynamic properties are not held in the Trader and have to be obtained at run-time via a dynamic property evaluator interface with the specified service. For example a dynamic property of the datastore could be the “space available”, which the Trader would obtain at runtime [32].

The second data structure is the Registry. It holds instances of the service types described in the Repository, i.e. it holds the details of actual datastores, e.g. “Departmental FileServer”, Floor 2, 3 gigabytes available, supports SQL. So by specifying a service type and a list of properties, a client can ask a Trader to provide a list of all the datastores that are support SQL and have over 1 gigabyte of space available.

2.5.2 Attributes

Each Trader also has Attributes. These define a Trader’s characteristics, i.e. policies for functionality supported and policies for scoping the extent of a search. Attributes are initially specified when a Trader is created and can be modified or interrogated via an administration interface.

2.5.3 Interfaces

Importers, Exporters and the Traders are all part of the Trading Community, i.e. all objects that interact to import/export services [31]. Interaction between members of the community is via a set of defined interfaces. Interfaces are also defined to other Trader components.

Chapter 2: Integrated Service Engineering and Distributed Processing Environments

The interfaces described below are the TINA specification. This specification was originally produced by the OMG for the CORBA Trading Service [33], which was itself based on the ISO's ODP Trader specification [31]. The interface names are defined in uppercase bold, and the operation names are in italics.

- The **LOOKUP** interface is used by importers to discover and import services, via the *Query* operation.
- The **OFFERITERATOR** interface is used to return a set of service offers from the *Query* operation by enabling the service offers to be extracted by successive operations on the interface.
- The **REGISTER** interface is used by exporters to advertise their services. They can advertise the services using the *Export* operation; the *Withdraw* method removes a service offer from the Trader; *Describe* returns the information about an offered service that is held by the Trader; *Modify* is used to change the description of a service as held within a service offer.
- The **DYNAMICPROPERTYEVAL** interface is provided by an exporter who wishes to provide the value of dynamic properties at runtime, e.g. when exporting a datastore interface, a dynamic property could be "space available" which can only be derived at runtime. The exporter provides a reference to the interface so that the Trader can invoke the *evalDP* operation to obtain a property value.
- The **LINK** interface allows a Trader to use the services of another Linked Trader. Links can be added, removed, listed and modified via the interface.

Chapter 2: Integrated Service Engineering and Distributed Processing Environments

- The **PROXY** interface allows a Trader to determine at runtime the object reference of a service offer, because although the Trader has the offer name and type it does not have an object reference.
- The **SERVICETYPEREPOSITORY** interface allows service types to be created and managed in the Repository. It provides operation to allow the user to add, remove, list and modify service types in the repository.
- The **ADMIN** interface allows the administrator to configure the system and set various parameters. There are four methods. The *Attributes* and *Set* operations allow administrator to set and return the values of the current trader attributes. *List_offers* allows the administrator to perform housekeeping by obtaining a handle on each of the offers (excluding proxy offers) within a Trader. *List_proxies* returns a set of offer identifiers for proxy offers held by a Trader.

2.5.4 Linked Traders

Traders from different domains can create links or federations and so pool their service offers. If a Trader cannot find a matching service, it will then pass the request onto another Linked (or Federated) Trader. The linked Trader can then check its Registry to see if it can match the original request. So when a Trader links to other Traders, it makes the offer spaces of those other traders implicitly available to its own clients, i.e. linked trading allows an importer access to multiple Trading domains.

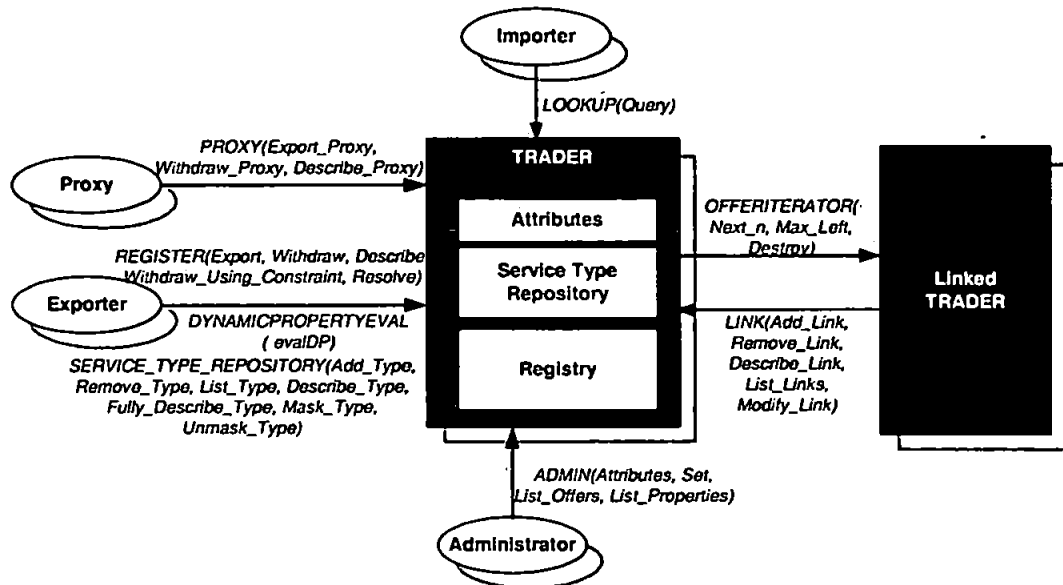


Figure 2-9 Trader

A Trader has to be explicitly linked to another. However, these other Traders may be linked to yet more Traders, and so the initial Trader can reach a large number of other Traders. This can also cause a problem by providing too much choice. In order to narrow the search parameters on service offers, Traders provide Policies, Constraints and Preferences. Policies are used to provide information that affects a Trader's behaviour at runtime, e.g. allow the client to specify the scope of a search, how the search is to be performed or how many trader links can be traversed. Constraints allow the client to specify search criteria, by using a well-formed expression conforming to a constraint language. For example, a client could use SQL as a constraint language. Preferences allow the client to specify the order in which offers are returned. Figure 2-9 illustrates an example of a basic Trader structure described.

2.5.5 Uses of an Unsecured Trader

Apart from the core function of the Trader providing references to server objects, it has also been suggested by Resnick [34] that the Trader could be used to standardise World Wide Web (WWW) facilities. There are a number of search engines, web crawlers and white pages such as Yahoo, HotBot, and Alta Vista. However, these facilities, especially the search engines, lack a programmatic interface and differ not just in implementation but also in how they are accessed, how predicates are formed and how Uniform Resource Locators (URLs) are registered. Therefore a synergy between the Trader and the Internet facilities would offer a solution. Search engines would benefit from a standardised programmatic API, which is important when the search engines are not just interested in web pages, but also in intelligent objects that export functional interfaces, and the clients seeking them are not people using GUI interfaces but client objects using APIs. The search engines offer highly scalable data stores, with fast search algorithms and accumulated stores of server objects that have already been categorised. This opens up a whole new opportunity for offering services, of any kind provided by intelligent objects, to both users and client objects in a distributed environment.

It is also important to remember that ODP and Trading is not just for Internet use. It is designed to work on any heterogeneous distributed object environment. Therefore some other possible uses of the Trader have been suggested by the Distributed Systems Technology Centre (DSTC) research group in University of Canberra, Australia [35]:

Chapter 2: Integrated Service Engineering and Distributed Processing Environments

- real-time trading, e.g. dynamic configuration of services within telecommunications switches (combining bandwidth from local and trunk carriers to provide an end-to-end service);
- large scale trading, e.g. using trading to access network elements from network management applications for a national telephone system.

2.6 The Security Service

While the security principles and the current DPE security model will be discussed in the following chapters, a brief description of the CORBA security service, which will be used later in the research, will be presented in this section. The CORBA Security Service (CORBASec) [36] provides a framework for distributed object security. There are two levels of security. **Level1** provides protection for applications that are “unaware” of security, by transparently calling security functions on object invocation. **Level2** security provides more facilities and allows applications themselves to control the security provided, i.e. security-aware applications.

CORBASec currently supports certain levels of authentication, access control, confidentiality, integrity and non-repudiation. Another feature of CORBA security is the use of credential delegation between objects. It allows credentials to be propagated along an object request chain.

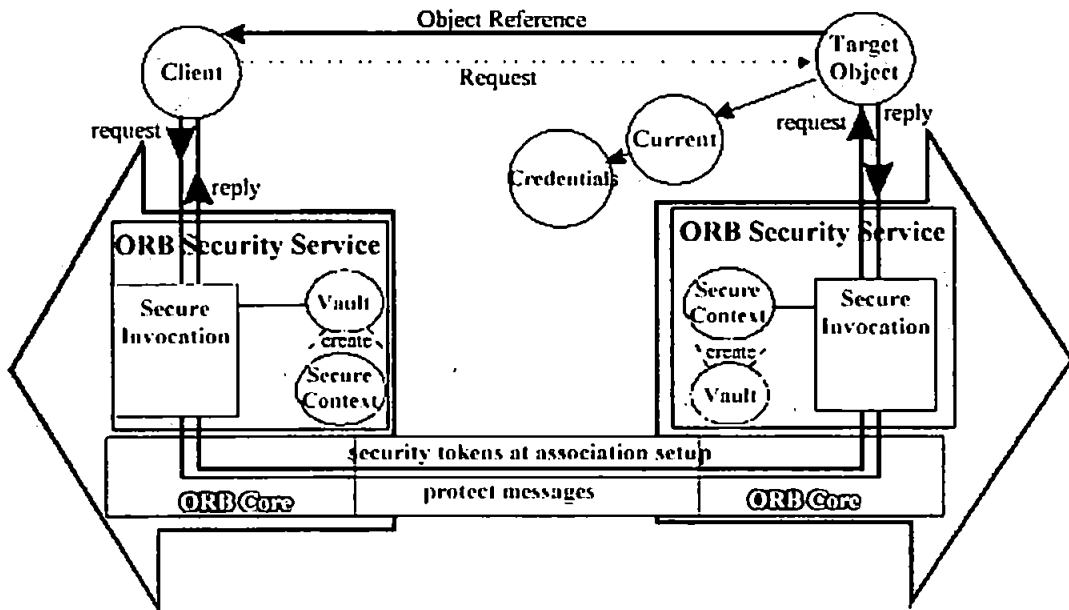


Figure 2-10 CORBA Security Service

Security is implemented by a number of objects, as shown in figure 2-10 above. Apart from the specific security interfaces, CORBA makes use of two objects, **Current** and **Credentials**. Current, a pseudo-object initially used by the transaction service to propagate transaction context, has been adopted by security to propagate the security context. It does so by holding a reference to Credentials. Once a user is authenticated, a Credentials object is created. It holds information such as roles, privileges and an authenticated ID.

In order to provide “out-of-the-box” interoperability across multi-vendor ORBs, CORBA now defines different Common Secure IIOP (CSI) profiles [37]:

Chapter 2: Integrated Service Engineering and Distributed Processing Environments

- **CSI Level 0** security provides identity-based policies without delegation. Therefore only the identity (no other attributes) of the initiating principal is transmitted from the client to the target, and it cannot be delegated.
- **CSI Level 1** security provides identity-based policies with unrestricted delegation. As in CSI Level 0 only the identity is transmitted from the client to the target. However, the identity can be delegated to other objects, using simple unrestricted delegation.
- **CSI Level 2** security provides identity and privilege-based policies with controlled delegation. Therefore, all attributes can be passed from client to target, including access, and audit identities and any privilege attributes such as role or group. These attributes can be delegated, but are subject to any restrictions placed on the delegation process by the initiating principal.

CSI Level 0 is addressed by SSLIOP, an implementation of IIOP over a Secure Socket Layer (SSL) [38] connection. The full-scale security version of IIOP, SECIOP, is used by the other mechanisms⁴. Both protocols lie between the network transport layer (TCP/IP) and the GIOP protocol layer, and so are considered mutually exclusive.

Figure 2-11 below summarises the objects that are specified in the CORBA Security Service specification [36]. These objects are categorised into their security service functionality.

⁴ CSI version 2 is addressing the use of SSLIOP to cover Level 1 and 2, by introducing Privilege Attribute Certificates, so that SSL can provide access control.

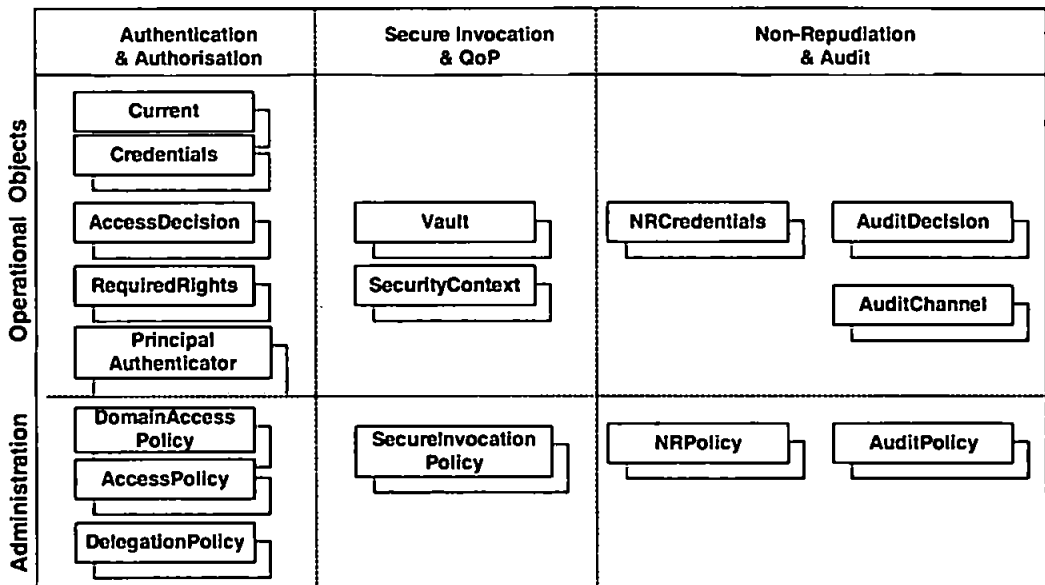


Figure 2-11 CORBA Security Objects

The object functionality is defined as follows:

Operational Objects:

- **Current:** represents service state specific information associated with the current execution context and is available to both clients and servers.
- **Credentials:** represents a particular principal's credential information. It includes information such as that principal's privilege and identity attributes, such as an audit id. It also includes some security-sensitive data required when this principal is involved in peer-entity authentication. However, such data is not visible to applications. It is referenced by the Current object.
- **PrincipalAuthenticator:** responsible for authenticating principals and creating Credentials containing their privilege attributes.

Chapter 2: Integrated Service Engineering and Distributed Processing Environments

- **AccessDecision:** responsible for determining whether the specified Credentials allow an operation to be performed on a target object. It uses access control attributes for the target object to determine whether the principal's privileges, obtained from the SecurityContext (see below) are sufficient to meet the access criteria for the requested operation.
- **RequiredRights:** specifies which rights are required to use which operations of an interface, and is generally used by AccessDecision.
- **Vault:** facilitates creating Credentials objects and establishing security contexts between clients and targets when they are in different trust domains.
- **SecurityContext:** hold security information about the client-target security association and are used to protect messages, and is generally created by the Vault object.
- **NRCredentials:** hold the identity and attributes of a principal, which are specifically used for non-repudiation operations. The attributes include whatever is needed for identifying the user when generating and checking evidence, e.g., it might include the principal's key (or provide access to it) when required to sign evidence. NRCredentials is available via the Current object.
- **AuditDecision:** used to obtain information about what needs to be audited for the specified object/interface in this environment.
- **AuditChannel:** used to write audit records.

Administrative Objects:

- **DomainAccessPolicy:** implements the access policy, by granting/revoking a set of named “subjects” (e.g., users) with a specified set of “rights” (e.g., get, set, manage, use) to perform operations on the “objects” in the domain.
- **AccessPolicy:** defines what subjects are available in a domain, and what rights they can be granted, for particular operations.
- **DelegationPolicy:** controls which credentials are used when an intermediate object in a chain invokes another object.
- **SecureInvocationPolicy:** specifies secure invocation policies for security associations, including controlling the delegation of client’s credentials, and message protection.
- **NRPolicy:** holds the non-repudiation policy information, such as the evidence types required.
- **AuditPolicy:** identifies which operations (if any) on an object will be audited.

The full CORBASec specification [36] contains more comprehensive details on these objects and the security service.

2.7 Summary

This chapter has studied the concepts of ISE and Distributed Processing Environments. All DPEs have a defined set of requirements such as independence, openness, transparency, scalability and object-orientation in order to provide a flexible environment that can adequately support distributed objects. These requirements will have to be taken into consideration when defining any security framework within this environment.

The main focus of the ISE study was TINA, the telecommunications architecture, which has widely influenced the telecommunication and distributed research environment. The TINA DPE architecture consists of the DPE Kernel, kTN and DPE services. It is the DPE security service and the management security function that is of interest to this research and they will be examined in more detail in the following chapter.

TINA is also a practical standard that has been adopted by the OMG for its ORB technology. CORBA, along with its Trading and Security Services was described, and will be used later in the research. The next chapter will now look at security in general and the issues and principles that arise within the context of a DPE.

3. Logical Security in Distributed Systems

3.1 Introduction

According to Price Waterhouse Coopers [39], it is estimated that in 2000 hackers will cost businesses around the world almost 1.6 trillion US dollars and that 40,000 person years⁵ of productivity due to computer downtime. However, the survey is believed to underestimate the total cost because it only refers to companies with over 1000 employees and so does not take small to medium sized enterprises into account. This highlights the extent of the problem on a global scale, and figures indicate that the problem is getting worse. The latest CERT/CC statistics show that the number of security incidents is increasing. In 1988 only 6 incidents were reported, while in 2000 21,756 were reported [40]. With such a high cost, security cannot be ignored; the situation has to be addressed.

Security refers to procedural, logical, and physical measures that are aimed at preventing, detecting or limiting any system misuse, be it accidental or deliberate. Procedural measures refer to administration and policies such as changing passwords regularly or selecting trustworthy staff. Physical measures are those taken to ensure security by tangible means such as locking doors. Logical measures are those such as authentication and access control. It is the logical measures that are examined in this chapter.

⁵ One person year is defined as one person working a 24-hour day, 365 days a year.

General security concepts, which have well accepted, standardised specifications, will be explained. Other general security principles, relevant to the research will also be presented; they will prove useful in guiding the definition of a DPE security framework in the up-coming chapters.

3.2 Security Principles

Security for distributed systems uses a set of overlapping concepts or services, as specified by the International Standards Organisation (ISO) [41] - authentication, access control, confidentiality, integrity, and non-repudiation. Security management is also considered. By applying these concepts, a system can be made more secure. The ISO security services relate to distributed environments and to the Open System Interconnection (OSI) Reference Model [42], and so the concepts also apply to a DPE. The following sections will look at each of the services and what they provide.

3.2.1 Access Control Service

The ISO states that “security is used to minimise the vulnerabilities of assets and resources” [41]. An asset is anything of value in a global computing system and a vulnerability is any weakness that could be exploited to violate a system and its information. Therefore, one obvious way to minimise threats is to limit the users who can have access to assets/resources [43]. This means that all data, programs and services need to be protected, but not just from users but also from illegal access by other programs and services. It should be noted that threats occur from two basic areas, external and internal. Generally, external threats can be minimised by denying

access to the network, e.g. permitting external access only through a firewall [44]. Internal threats are more difficult to handle, or even to recognise. However, internal threats are a significant problem as numerous studies have shown that they have typically accounted for about 80-85% of security breaches [45], although the CSI/FBI 2000 survey shows, external incidents are increasing because of the Internet [46].

Threats can be deliberate or accidental. They can occur as the result of:

- destruction of assets;
- corruption or illegal modification of assets/resources;
- illegal or unauthorised disclosure of data;
- interruption or denial of services.

Therefore control of access needs to be addressed at several levels:

- access into the network / DPE;
- access to an asset or resource;
- type of access to an asset or resource.

The Access Control Security Service protects resources from unauthorised use. It can be used on various assets, e.g., communications packages, stored data, or components.

The service can be broken down into several core components [47]:

- **Subjects & Objects:** the entities to which access control is applied to or utilised by.

- **Access Operations:** Access operations specify the type of access that is permissible. It requires the definition of Access Rights [48] and Access Attributes [49].
- **Access Control Structures:** The basic access control structure is an Access Control Matrix (ACM). Two derivatives of the ACM are the Access Control List (ACL), where access rights are stored with the object, and the Capabilities List (CL) where access rights are stored, using an un-forgeable token, with the subject.
- **Intermediary controls:** Administration needs to be as simple and effective as possible, therefore intermediary controls are introduced. Privileges collect the right to execute a certain set of operations under a particular activity, e.g. system administration. Privileges are often specified in a predefined set of Roles, where subjects derive their access rights from the role they are performing.

3.2.2 Authentication Service

One type of threat is known as a masquerade; that is when an entity successfully pretends to be some other legal entity and thereby gains illegal access to a resource. Therefore before granting access to a user or resource, the security service should be able to guarantee that the user/resource is actually who/what it claims to be [50, 51]; this is the responsibility of the authentication service.

In the case of connection-oriented environments (i.e. CORBA), peer entity and peer-to-peer authentication apply. Peer entity authentication provides corroboration of the

identity of a principal within the context of a communication relationship (only one entity is identifying itself, either the client or the server), while peer-to-peer authentication (also known as mutual authentication) involves both client and server entities authenticating each other. The process involves the exchange of authentication information. The information exchanged will depend on the authentication technology used. It is generally based one of the following:

- secret knowledge - e.g., passwords;
- cryptographic techniques - e.g., digital signatures [52, 53];
- characteristic - e.g., biometrics [54, 55];
- possessions - e.g., smartcards [56].

A key authentication concept, which should be mentioned at this point, is the Trusted Third Party (TTP) [57]. In the case of public keys, it is actually impossible to be sure that a particular user's public key is not a forgery unless a digital certificate is used [58]. The certificate contains the user's public key and an endorsement that the key is real, made by a TTP's digital signature. The issue of trust is now shifted to the TTP – so if an entity trusts the TTP, he can trust that the user's public key he received is real, and not a forgery. Such TTP's are called Certification Authorities (CA). The X.509 Authentication Framework [59] specifies a framework for certificates and a hierarchical structure for CAs, as illustrated in figure 3-1 below.

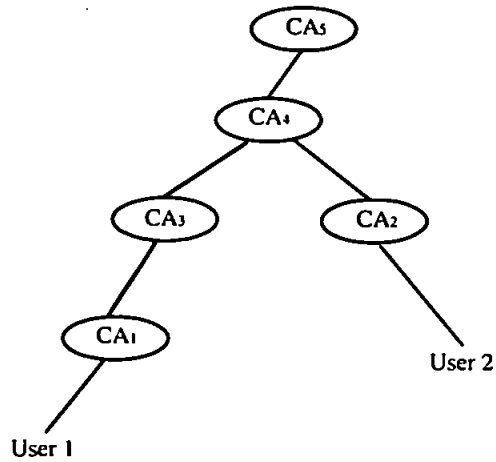


Figure 3-1 X.509 Certification Authority Hierarchy Structure

In the above figure, User1 and User2 do not currently trust each other because they are in different domains, using different CAs, CA₁ and CA₂ respectively. However, they do have a common CA when the hierarchy structure is used, CA₄, and because CA₄ has issued CA₂, that means that User1 can trust User2's certificate. Similarly, User2 will trust User1's certificate, because working through the hierarchy, CA₁ is issued by CA₃, which in turn is issued by the trusted CA₄.

3.2.3 Confidentiality Service

Confidentiality on a network means being able to guarantee the privacy and secrecy of an asset, such as a data file containing personnel details. Confidentiality can be applied to data, whether it is in storage or in transit, and may be applied to only selected fields, instead of a whole message/record, in the interests of enhancing performance while still providing adequate protection. There are two basic cryptographic approaches:

- **Symmetric:** where the encryption and decryption keys are the same, and therefore keeping the key secret is imperative. An example of this is the Data Encryption Standard (DES) [60];
- **Asymmetric:** where the encryption and decryption keys are different. In this case the encryption key (or public key), is available to everyone so that they can encrypt plaintext and then send the ciphertext to user A. However, only user A will know the decryption key (or private key) and, therefore, only he can decipher data sent to him. An example of this is the Rivest Shamir Adleman (RSA) algorithm [61, 62].

The existence of certain regulatory requirements, in relation to cryptography, complicates access to and use of cryptographic mechanisms in certain countries [63]. There are two main issues – key length and cryptography use. The cryptographic algorithm and key length define the strength of encryption. Some countries have export laws that limit the key length of a given algorithm, e.g. US, France, Russia. The other issue relates to the use of cryptography, i.e. whether it is used for authentication and integrity purposes versus its use for confidentiality. When used for confidentiality, the export laws are usually more stringent. However, in the case of the US, new regulations were defined in January 2000 that considerably relaxed the tight restrictions that were previously in place [64].

3.2.4 Integrity Service

Integrity of resources ensures that attempts to modify data can be detected no matter what corruption attempts have been made on them [65]. In a comprehensive security survey database, maintained by Cohen, the attack section lists 95 possible classes of attack that can be used in networked systems, it includes everything from computer viruses to input overflows. Of these 95 classes, 66 are used to corrupt information [66]. Therefore any integrity services must guard against any threats involving illegal asset/resource modification. Integrity is applied to both data and system resources. Data integrity ensures that the data has not been accidentally or maliciously altered or destroyed [67]. System integrity ensures that all resources in a network are available to users and that the system remains in a state consistent with strictly defined security rules and regulations [68].

Cryptography can be utilised in the integrity service. An important cryptographic mechanism is the one-way hash function. It takes a variable-length string (called a pre-image) and converts it to a fixed-length output string (called a hash value). It works in one direction, i.e. it is easy to compute a hash value from pre-image, but it is difficult to generate a pre-image that hashes to a particular value. Another desirable attribute of a one-way hash function is to ensure that it is also collision-free, i.e. it is hard to generate two pre-images with the same hash value. Therefore a one-way hash function can be used as a fingerprint to a particular pre-image, e.g. Secure Hash Algorithm (SHA) [69] and Message Digest 5 (MD-5) [70]. A one-way hash function with the addition of a secret key is known as a message authentication code (MAC) [71]. The hash value is a function of both the pre-image and the key. Therefore only

someone with the identical key can verify the hash value. This is useful for providing authenticity.

3.2.5 Non-Repudiation and Auditing Service

Repudiation is the denial of an action by an entity. For example, a user may deny sending or receiving a message. Non-repudiation forces an entity to *be accountable for* its participation in some action [72]. The ISO defines the types of evidence required in a Non-Repudiation service [73]. There are several proofs, some of which are described below:

- **Proof of Origin:** provides the recipient with unforgeable proof that the message originated from the originator.
- **Proof of Receipt:** provides the originator with unforgeable proof that the message was received by the original recipient.
- **Proof of Submission:** provides the originator with unforgeable proof that the message was submitted for delivery to the original recipient.
- **Proof of Delivery:** provides the originator with unforgeable proof that the message was delivered to the recipient.

Non-repudiation is made up of a set of supporting facilities that are required to provide a full service; it includes evidence generation and verification, evidence storage and transmission, and an adjudicator to settle any disputes using the evidence produced. A notary is also required.

Auditing is an intrinsic part of Non-Repudiation. It records security relevant events⁶ in an audit trail (log) for later analysis. This analysis can be used to help identify unauthorised activity within the system. The audit service can also be used to generate alarms to indicate that a more immediate response is required. ISO defines that the security audit function needs to provide trail analysis, archiving and examining, and alarm handling.

A specialisation of auditing is Intrusion Detection. It covers the monitoring of network activity and the analysis of data for potential vulnerabilities and attacks, historical or current. It is an important component in system security. There is a lot of research emphasis on this subject, the most prominent are Next-generation Intrusion Detection Expert System (NIDES) [74] and more recently Event Monitoring Enabling Responses to Anomalous Live Disturbances (EMERALD) [75]. There are also many commercial products available [76].

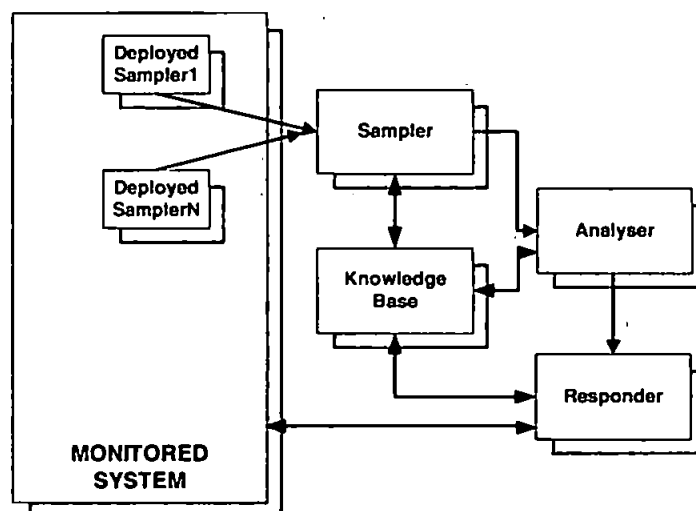


Figure 3-2 Monitoring Agent Structure

⁶ A security relevant event is any action within the system, which has been marked as being of interest to the security service, e.g. user authentication, object creation, or database access.

In Intrusion Detection Systems (IDS), there are three basic processes involved monitoring, analysis and response. These functions are represented in the generic IDS solution, the Monitoring Agent [77] illustrated in figure 3-2 above. It is comprised of the following components:

- **Sampler:** pulls information from the system and produces audit logs;
- **Knowledge Base (KB):** keeps the sum of all knowledge that the monitoring system requires to operate. The KB will hold the following types of information:
 - security policy to ensure that the monitoring is in accordance with the overall system policy;
 - security log to record recent events that have taken place in the system;
 - profiles of the activities of every user and resource in the system;
 - sampling information to generate the security logs;
 - analysis information that is required by the analysis techniques;
 - response information that is required by the responder.
- **Analyser:** takes information from the sampler and compares it with the data stored in the KB and from an analysed conclusion, it determines whether a security violation has taken place.
- **Responder:** takes the analyser output and information from the KB, and it decides and implements the action to be taken.

3.2.6 Security Management

The ISO [41] defines a security policy as a set of criteria for provision of security services. It defines what is and what is not permitted in the area of security during general operation of a secured system. It must be implemented by taking the appropriate security measures. However, security measures will not be effective unless the user understands what needs to be protected and can determine what mechanisms are used, i.e. what the policy is. Security needs a complete and usable administration system that will allow administrators to maintain and operate security on a day-to-day basis.

Administration occurs within a domain. A security domain is defined by the ISO as a set of resources where a specific security policy should be applied. Security management must control and support security within its own domain and possibly allow for inter-domain security interaction. OSI defines three categories of security management; security system management, security services management, and security mechanism management.

- **Security System Management** is responsible for applying security management to the whole system. Firstly, it ensures that the security policy is implemented. Secondly, it must manage interactions with other management functions and with the other security management systems (described below).
- **Security Service Management** deals with all events in relation to security services. It will decide which mechanisms will implement a service. It will negotiate for these mechanisms and then invoke them.

- **Security Mechanism Management** ensures that mechanisms can operate by providing all the necessary resources. For example, in the case of key management, it will generate suitable keys, determine which entities should receive a copy of the key and then distribute keys securely.

3.3 Other Principles relevant to DPE Security

While the security principles described above apply to any type of system requiring protection, the following sub-sections outline some principles that are particularly pertinent to a DPE.

3.3.1 Security Domains and Trust Models

Within any security system, trust is involved somewhere, e.g. the receiver a certificate has to trust the Certificate Authority's process of registration and certification; a system administrator has to trust that users will not give their userids and passwords to unscrupulous hackers; users have to trust system administrators not to abuse their privileges and access private data; Internet shoppers have to trust on-lines businesses to protect their data, especially their credit card numbers. When trust breaks down the consequences can be devastating. If the trust is misplaced, a system can be compromised. A hacker can use a password to break into a system and steal, modify or damage data or available services. If, as happened to several on-line companies, credit card information is compromised [78] then companies can go out of business because the consumer has no confidence in the company's ability to protect their data.

Trust is not only integral to a security model, it is necessary to the successful operation of the security system.

Three possible trust scenarios can function in an interoperability model:

- **No trust:** The issue of having no trust existing between disparate security domains means that mutual suspicion exists. No attempt will be made to establish trust and the domains will continue to treat each other with suspicion. This type of behaviour can be implemented through the use of 'guest' privileges, which allow an untrusted entity very restricted and controlled access to a system.
- **Pre-existing trust:** This scenario refers to the fact that two disparate security domains trust each other due to a previously negotiated trust between them. This type of trust is generally achieved by security administrators from the domains agreeing the terms and conditions of secure interoperability. This can include defining recognised userids that would operate in both domains and defining security mapping between the access control privileges of each domain, i.e. administrative co-operation between the two domains is necessary. For example, in domains that use roles and access privileges to a file system, "UserAB" is defined in domain A and domain B. In domain A, "UserAB" is a member of the "manager" group and has "read" and "write" access to all files on the files server. In domain B, "UserAB" is a member of the "technician" group and has "update" access to files in the technician's directory.

- **Trust needs to be established:** In the pre-existing trust scenario above, the trust was defined between specific domains and required administrative interaction in both domains. There is one other scenario. This is when trust can be established between two domains that have no prior knowledge of each other. An example of this is the use of Secure Socket Layer (SSL) [79] with certificates and Certificate Authorities. “UserAB” is a member of domain and has a certificate issued by Certificate Authority 1. “UserAB” tries to access a server in domain B. Domain B has no knowledge of domain A, but it does recognise Certificate Authority 1. Therefore it can authenticate UserAB’s certificate and allow access to the server. In this case, domain A and B have no prior knowledge of each other but they do have a common trusted third party, Certificate Authority 1, which is used to establish trust.

3.3.2 Distributed Trusted Computing Base

Security trustworthiness is the ability of a system to protect resources from exposure to misuse from malicious or accidental means. However, this is more complex in a DPE than in a centralised system such as IBM’s Resource Access Control Facility (RACF) [80]. Trust in a centralised system is usually static because servers are generally trusted and remain trusted through their entire life. Trust is also confined to a single security facility, such as RACF in an OS/390 environment. This is not the case in a DPE. The security model can exist over multiple distributed platforms with various security mechanisms, such as Sesame [81], Kerberos [82] or SSL, and the trust model is not static over the lifetime of an object, because an object can be both

client and server and so can be both trusted and untrusted depending on the role it is playing at a given time.

A Trusted Computing Base (TCB) is the totality of protection mechanisms within a computer system, including hardware, firmware and software, the combination of which is responsible for enforcing a security policy. The ability of a TCB to enforce correctly a unified security policy depends on the correctness of the mechanisms within the TCB, the protection of those mechanisms to ensure their correctness and the correct input of parameters related to the security policy [83]. In a DPE the notion of a distributed TCB has to be adopted, because the mechanisms, data and program logic used by the TCB could also be distributed. Therefore, a distributed TCB can be seen collection of objects and mechanisms that must be trusted so that a secure end-to-end connection can be made between a client and server. This implies that the distributed TCB may need to include parts of the Native Computing and Communications Environment (communication network, operating system and any security mechanisms resident therein), the DPE kernel, DPE services (including the security service itself) and possibly some related TINA applications (such as management applications).

3.3.3 Interoperability

Interoperability relates to the problem of allowing an interaction to occur between two disparate domains. There are several approaches that can be used to deal with the issue. The various merits and applicability within a DPE environment of these approaches will be discussed in the following sections.

3.3.3.1 Bridges

A bridge provides a point of connection between two disparate domains. It can provide translation between the domains, so that interoperability is possible. The bridge can exist at any level. One example of a bridge is the Wireless Application Protocol (WAP) Gateway [84]. WAP is the de-facto standard for the presentation and delivery of wireless information and telephony services on mobile phones and other wireless terminals. The WAP specification uses standard Web proxy technology to connect the wireless domain to the Web.

There are two basic types of bridge, immediate and mediated. Immediate bridges are full bridging solutions between two domains. They map specifically from one domain to another. Immediate bridges provide a fast and efficient solution but are inflexible because they only provide a mapping between two specific domains. Therefore, if there are n domains, which require bridges to interoperate, then the number of bridges required is:

- $(n^2 - n) / 2$

In the example below there are 4 terminals, each in its own domain. Therefore the number of bridges required in the immediate bridging solution is 6.

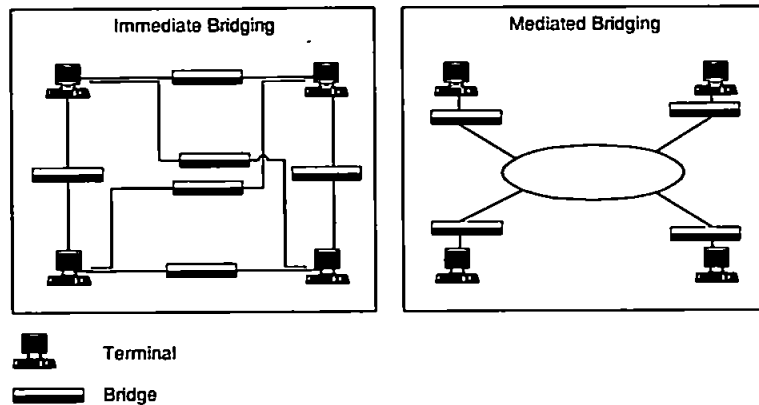


Figure 3-3 Interoperability Bridging Solutions

Mediated bridges provide the translation to some common domain. In this case, only n bridges are required between n domains, and therefore the number of domains can be easily extended without increasing the number of bridges exponentially. In the example illustrated in figure 3-3 above only 4 bridges are required for the 4 domains. However, the use of mediate bridges is not as efficient as immediate bridges for small number of domains, because it requires two bridges between any two domains, i.e. every message is translated twice.

These bridges deal with bridging technology domains. The issue of bridging security domains presents other problems. When considering a mechanism level security bridge, one obvious problem comes to the forefront – any message that is subject to encryption or an integrity check has done so using a specific security mechanism (algorithm). Therefore, to translate to another mechanism, the bridge is required to decrypt and then re-encrypt the message. This would add a considerable performance overhead, to a service that may already be stricken by performance degradation from

the initial encryption, or integrity checksum. If a mediated bridge is used then this performance hit will be doubled because two bridges will exist and translate the message. If a mediated bridge is used then the issue of trust and deployment exist. For example, the bridge will need to have access to both types of mechanism and could possibly reside in one of the two domains. This scenario would imply that the bridge is not necessary as the domain already would have access to the mechanisms. However, the bridge could also reside outside both domains with a TTP, but then trust must be established between each of the domains and the TTP, as the TTP will have access to the original message at some point during the translation process. As such it offers another point of vulnerability over which the two domains have no control.

3.3.3.2 Standard Mechanisms

The use of specific bridging technology is only one solution. Another solution is to use common technologies between the client and server. There are two requirements:

- Common protocol that will undertake the initial negotiation between client and server;
- Common set of security mechanisms available in the client and server installations.

The common protocol will begin the negotiation between the client and server. It will allow the client and server to select a common security mechanism(s) that meet the requirements for the secure association between them. Once the negotiation has been completed, the agreed mechanism(s) can be used to provide the secure context

between client and server. The most common example of this is the SSL protocol, which uses the SSL Handshake protocol to define what mechanisms will be used by the client and server to provide a secure context. The secure communication is then provided by the SSL Record protocol. The assumption here is that the client and server are able to agree a common set of mechanisms that meet the minimum security requirements of their respective security policies, e.g. both client and server policy specify that they require encryption, and an appropriate mechanism is available to secure the communication.

3.3.3.3 Generic Tokens

A generic token is standardised, non-specific data that can be used to provide data between two entities. Such tokens are commonly used as a means of communicating data for many different purposes. For example, a generic token is the format that NIDES (see Section 3.2.5) uses to distribute and work with audit data. Another commonly used example is the Generic Security Service Application Protocol Interface (GSS-API) [85]. It provides portability between distributed security architectures by using simple interfaces to security services and generic tokens, which can then be implemented and utilised by a range of underlying mechanisms and protocols. The generic tokens are successful because they use opaque data, mechanism identifiers for a standard set of mechanism, and standardised status codes. This means that GSS-API is extensible and can easily adopt new mechanisms. Therefore, any generic token should use these devices to ensure that it remains portable across multiple mechanisms and environments.

3.3.4 Mechanism Independence and the Separation of Mechanism & Service Management

Mechanism independence is the notion that a service's functionality is not dependent upon the mechanisms that implement it, in other words different mechanisms can be used by the service to provide the same service functionality. This results in two benefits. Firstly, since the user is not concerned with what mechanism is used to implement a service (e.g. a service object), it means that the service should be portable across different implementation platforms and also assists with the OO objective of encapsulation. Secondly, a result of mechanism independence is the separation of mechanism from service management. This facilitates flexibility and allows the introduction of new mechanisms without compromising the service functionality. This feature also assumes that the protocol is designed to accommodate generic tokens/data types that the appropriate mechanism can then utilise, i.e. the protocol or any object interfaces are not mechanism dependent. Architectures such as the Comprehensive Integrated Security System (CISS) [86] actively promotes independence by utilising a layered architecture. GSS-API also promotes independence by utilising generic interface definitions that are not dependent on any underlying mechanism.

3.4 Summary

This chapter has looked at logical security for distributed systems. Firstly, it examined general security principles, which apply to all systems requiring protection. They include all the security services – authentication, access control, integrity,

Chapter 3: Logical Security in Distributed Systems

confidentiality and non-repudiation - which need to be applied in a DPE security solution.

The discussion proceeded to cover other principles that will prove important in defining DPE security by addressing issues encountered when operating in a distributed, heterogeneous, multi-domain environment. The discussion proposes some methodologies for addressing these problems and they should also be considered in a DPE security solution.

The next chapter will provide an analysis of the security requirements for DPEs, and how they can be achieved, which will then assist in the definition of a new DPE security framework.

4. Requirements for a new Framework for DPE Security

4.1 Introduction

This chapter reviews the currently defined security requirements for DPEs and identifies that they do not fully cover the problem space to support the design of an appropriate security framework. The problem domain is then analysed with an industrial partner ⁷ and goes on to provide a complete set of the DPE security requirements.

These requirements are then applied to the TINA security model, which is found to be wanting by comparison with the requirements. The chapter concludes with a new security framework that address the requirements identified in the new DPE problem domain.

4.2 Requirements for DPE Security

Chapter 3 outlined the general security services (see section 3.2) and also identified some principles such as trust models and distributed TCB (see section 3.3) that would prove useful in defining a DPE security model. However, they do not provide a complete set of DPE security requirements, necessary to define a framework.

⁷ Research was done in collaboration with Orange (Prof. Paul Reynolds), and reviewed by an industry expert.

This section considers the particular requirements of a generic distributed object system (GDOS), which have been collated from current literature. Such systems are the genesis of a distributed processing environment. By their nature GDOS are less secure than client/server because the act of distribution transparency means that the traditional operating system cannot be trusted to protect the server resources and data in transit. Therefore servers in a distributed system have to find new ways to protect themselves without adding an unacceptable overhead effecting performance and availability.

4.2.1 Distributed Object Complications

A distributed object system is considered less secure than traditional client/server systems [24, 36] for the following reasons:

- **A distributed object can have the roles of both client and server:** In traditional client/server systems, servers can always be trusted. However this is not true for distributed systems, as roles are not clearly defined, e.g. a single entity can act as both a client and a server and so the trust model is more complex.
- **Distributed object interactions are not transparent:** Because of encapsulation⁸ [11], a client is not fully aware of the interactions that take place when it invokes an object and so they are more difficult to control.

⁸ **Encapsulation** consists of separating the external aspects of an object, which are accessible to other objects, from the internal implementation details of the object, which are hidden from other objects.

- **Distributed objects are polymorphic:** Objects can be replaced without any interruption to the system, as long as the interface remains the same. This provides a perfect opportunity for Trojan Horses⁹ to infiltrate a system.
- **Distributed objects can scale without limit:** There are no theoretical limits on the number of servers or clients in a system, therefore a security system will have to scale and be able to cope with the large number of resources and users that could possibly be involved.
- **Distributed objects are dynamic:** Objects are created, operate and can then be destroyed. Such dynamics have to be performed securely. The system is not static and the security system needs to be flexible enough to securely accommodate this.

4.2.2 Review of Currently defined GDOS Security Requirements

The following is a summary of an analysis into the security requirements as currently stated by TINA [87] and the OMG [88]. Each requirement, and its implications upon the functionality of the security model, is described and then summarised in Table 4-1.

REQUIREMENT #1: The security system must support Identification and Authentication: Within a secure object system, it is imperative to identify and authenticate an entity. This process should support any authentication mechanism and should result in a unique set of certified credentials for the entity.

⁹ A **Trojan Horse** impersonates a legitimate entity to illegally obtain data or perform some other malicious activity.

Chapter 4: Requirements for a new Framework for DPE Security

There are two possible scenarios, intra and inter-DPE authentication. In the first, the entity seeks authentication within the local system, and is identified and authenticated locally. Therefore, the validation can be trusted locally. In the latter, the entity may have been identified and authenticated in another system within the distributed environment. Thus the security service has to support the validation of such an entity. This is achieved by either a Trusted Third Party (TTP) (see sections 3.2.2 and 3.3.1) from which the local security system can obtain a proof of identity or a proof that the identification and authentication is trustworthy, i.e. was executed within a trusted system or, by the use of a single sign-on facility, which would lower the number of user logons required (a facility which cryptographically can have a high overhead due to the authentication process).

Many different types of entities need to be authenticated, not just users. Services and objects will also request access to other services, data and system resources. All such requests have to be validated. Therefore, authentication will apply to entities at all levels of the system - users, services, and objects.

REQUIREMENT #2: The security system must support Access Control and Authorisation. Once an entity has been authenticated, it also requires privilege information that will define what objects (operations) it can access. This requires a set of privilege attributes be assigned to the entity.

There are multiple schemas that can be used for access control and the security model should be able to use them, including the use of roles/groups to reduce the administrative overhead (see section 3.2.1). Again two forms of operation are

envisaged, one within a local system, and another via a TTP to allow access across different domains.

REQUIREMENT #3: The security system must support Propagation of Attributes. Due to the nature of distributed object systems, objects need to be able to delegate their privileges/attributes to other objects. However, they also need to be able to apply constraints specifying when and where these delegated privileges can be used, otherwise the privileges could be used at anytime and in any domain by a rogue entity.

With different domains, different security policies may present some difficulty and require a trust relationship to be established between two domains. The attributes from one domain should be mapped to authorised attributes in the other domain to provide validation for access control and auditing.

REQUIREMENT #4: The security system must support Secure Communications. Communication, both for operational and system data, needs to be secured (see sections 3.2.3 and 3.2.4), but flexible.

The user should be able to select the Quality of Protection (QoP) required (e.g. cryptographic strength) and should also be able to select how much of the message needs to be protected. Again, the system should have the ability to support different encryption and integrity mechanisms.

REQUIREMENT #5: The security system must support Secure Stored Data. Objects are the definition of both behaviour and data, thus any data within an object or utilised by an object needs to be protected.

Chapter 4: Requirements for a new Framework for DPE Security

While the security service may not be able to secure the data, it should have the ability to indicate that the data is considered sensitive and so should be stored securely, thereby allowing external mechanisms to secure it, i.e. the Quality of Protection required.

REQUIREMENT #6: The security system must support Security Audit. Auditing of security relevant events is essential. The system should be able to identify events based on their classification and assign the audit information to an audit trail and/or an alarm process.

The audit records also need to be protected from any modification, both in transit or when stored in the audit trail. Tools are required to analyse trails, and access to a generic toolset (i.e. sampler, knowledge base, analyser and responder refer to section 3.2.5) and the audit trail records should be available to facilitate intrusion detection.

REQUIREMENT #7: The security system must support Non-Repudiation. To ensure accountability, non-repudiation facilities are required (see section 3.2.5). Non-repudiation will include the generation, verification, transmission and storage of evidence. Such a service also requires access to an adjudicator for dispute settlements.

REQUIREMENT #8: The security system must support Security Management. Security management needs to support the distribution system, service and mechanisms (see section 3.2.6). An administrative interface to handle each function is required; these interfaces should be comprehensive and easy to use, as usability will ensure that security is properly applied.

REQUIREMENT #9: The security system must support Interoperability. DPEs provide for an open and distributed environment, and allow interactions between different administrative domains.

Thus security policies and administration within a local domain need to be preserved, but this has to co-exist with the preservation of inter-domain security. Interoperability is required at both the invocation and security service level. Thus a secure invocation initiated in one domain to be completed in another requires the security services to inter-operate in order to facilitate this, and may require security attributes, i.e. credentials and privileges, to be mapped from one domain to another because they support different schemas. It will also require some negotiation to allow common security mechanisms and protocols to be agreed between different domains. Another option is the use of a gateway to translate between attributes/mechanisms/protocols (see section 3.3.3).

REQUIREMENT #10: The security system must support system Scalability. The architecture should accommodate and allow the evolution of networks, services and management capabilities from small to large (global) scale in terms of its ability to handle the number of users, nodes, and administrative domains required.

The security service itself must be scalable in order to cope with an "carrier class" large-scale systems, and so should support the use of roles/groups to reduce administrative costs and also allow the use of multiple inter-working security domains.

REQUIREMENT #11: The security system must support Integration with existing environments. There is already a huge investment in technology by the

telecommunication and data processing industry. Therefore, if this model is to be successful it must integrate with the existing technologies. This requires a flexible structure that will allow the security model to deal with multiple options for service and mechanism implementations and allow the flexibility to manage each of these different types and their data formats (see section 3.3.4 on mechanism independence and separation of mechanism and security management). This requirement also covers the need to meet the differing regulatory requirements that exist in different countries (e.g. rules regarding the use of cryptography).

REQUIREMENT #12: The security system must support system recovery. The recovery system should establish consistent security states after security failures by taking various actions. This involves the maintenance of the rules used to react to real or suspected security violations, the remote reporting of apparent violations of system security, and security administrator interactions [41]. Within the system a Knowledge Base (KB) could hold the maintenance rules, while the "Sampler" would be used to collect system data that is then processed by the analyser. The "Responder" defines the system response that should be taken in accordance with the rules stored in the KB (see section 3.2.5. for IDS).

The following table summarises the identified requirements, and lists the functionality that is required by a DPE security framework to facilitate the requirements.

Chapter 4: Requirements for a new Framework for DPE Security

No.	Security Requirement	Functionality required
1.	Identification and Authentication	Identify entities and generate identity attributes Use multiple authentication mechanisms
2.	Authorization & Access control	Generate privilege attributes Use multiple authorization mechanisms Use role/groups
3.	Propagation of security attributes	Specify when propagation is required Specify constraints on propagation
4.	Secure communications	Ability to select Quality of Protection Ability to select amount of message to be protected
5.	Secure stored data	Ability to specify that data needs to be secured Ability to specify the Quality of Protection
6.	Secure Auditing	Audit security relevant events Produce audit records Issue alarm Protect audit information in transit or in trail Should be extended to facilitate intrusion detection
7.	Non-repudiation	Generation/Verification of evidence Storage of evidence Secure transport of evidence Adjudication facility
8.	Management: - System - Service - Mechanism	Administrative interfaces required to handle management of each of these management functions
9.	Interoperability	Interoperability at all levels- - Invocation - Security Service, Mechanism and Protocol Mapping of attributes between domains
10.	Scalability	Security service must be scalable itself as well as working in scalable environment Use of domains Use of groups etc in administration
11.	Integration with existing environments	Flexible structure to allow the model to integrate with other technology environments/security models Facilitates regulatory requirements
12.	System Recovery	Knowledge base system

Table 4-1 GDOS Security Requirements

Whilst the twelve requirements identified above originated from a DPE environment (i.e. TINA and OMG), they do not specifically address the complete DPE problem

domain, and can be applied to any distributed object system. It is necessary to consider a new definition of the problem domain for DPEs, and focus on requirements in the secure DPE space.

4.2.3 Analysing the DPE Security Problem Domain

Figure 2-3 (see section 2.3.1) illustrates the framework of TINA system, of which the unifying component is the DPE. In order to have a comprehensive set of requirements it is necessary to view the DPE as part of the overall architecture and the layers around the DPE need to be included in the analysis to ensure that any relevant security functionality that extends beyond the DPE layer boundary is included.

The procedure for identifying the requirements for DPE security was to define the domains of security within the TINA system structure and secondly to identify what domains are relevant to the DPE security model, i.e. identify the scope of DPE security. The results of this analysis are grouped around the definition of three sub-domains of the TINA framework.

4.2.3.1 Transport Sub-Domain

The transport sub-domain covers both the NCCE and Hardware layers and relates to the security of the hardware and operating systems utilised by a TINA implementation.

REQUIREMENT #T1: The security system must support the procedures, both physical and logical, for preventing any intrusion or modification of networking or computing resources, i.e. it must support intrusion detection.

REQUIREMENT #T2: The security system must support such actions as ensuring correct installation and adequate protection of hardware and software, addition of software patches, and control of communications ports through firewall technology. [N.B. this requirement is considered outside the scope of the DPE security model, with the exception of the management of mechanisms, in particular security mechanisms, both hardware and software based, that may be utilised by the DPE security. The reason that this management function is considered an exception is because of the importance of mechanism-independence to DPEs and the fact that the kTN (providing a logical transport network for the DPE, using the NCCE resource) will be part of the NCCE.]

4.2.3.2 The Middleware Sub-Domain

The middleware sub-domain covers both the DPE Kernel and DPE Security Service concerned with the protection of TINA service objects, i.e. the computational objects used to create services. This security will need to operate at three separate levels: *operational, control and administrative.*

REQUIREMENT #M1: The security system must support functions such as ensuring only authorised access to objects, based on authenticated identities, and also the protection of any inter-object communications.

REQUIREMENT #M2: The security system must support the *operational* concerns of a TINA service, e.g. the video conferencing session, is secured. This includes the maintenance of integrity and confidentiality of any data streams and ensuring only authorised subscribers use the service.

REQUIREMENT #M3: The security system must support the *control* concerns of the TINA service by ensuring that information controlling the service configuration, e.g. the Quality of Service of a video stream, is protected. This involves securing the control data and ensuring audit services are available to track any control changes.

REQUIREMENT #M4: The security system must support the *administrative* concerns of the DPE, which includes the assurance that the security data relative to the service and subscriber (e.g. user and service profiles) is available and, that it can be administered securely.

REQUIREMENT #M5: The security system must support the DPE kernel. The DPE kernel is resident on every node. Although its internal security needs to be provided by local implementation means, inter-DPE security will have to be supported, as individual objects of logical DPEs may physically reside in different domains.

4.2.3.3 Application Sub-Domain

This sub-domain entails the protection of the applications built in the top layer of the system structure.

REQUIREMENT #A1: The security system must support adequate security by authenticating participants, only allowing authorised access to services, securing any inter-participant communications and providing adequate audit and non-repudiation facilities as required.

REQUIREMENT #A2: The security system must be provided in accordance with a security policy. It should be simple to administer and hide distribution issues, such as location, from the user.

REQUIREMENT #A3: The security system must operate at two levels because there is no guarantee that security is available in every domain.

Firstly, *security-inactive* applications (i.e. those that do not employ any security facilities themselves, but they are still subject to any DPE security that is specified by the active security policy within that domain) require that although a security provider operating in a DPE may not have any security policy, the DPE will automatically provide security. It may be assumed that this type of security is always available in a secure DPE.

Secondly, *security-active* applications i.e. those applications that are consciously implementing security themselves, are required to be able to access the DPE security service. Security-active applications are still subject to the DPE security policy, as with security-inactive applications, but they are also managing and implementing their own application security by utilising the DPE security service directly.

4.3 Formulating a DPE Security Framework

Prior to the proposal of a new security framework for a DPE it is necessary to visit the existing security framework and juxtapose this with the previously developed security requirements. This analysis will highlight differences and opportunities that will form the basis of the new framework proposals. (N.B. This is supplemented by using a service example of a videoconference).

The current TINA security model is focussed on the access session (see section 2.3.2), which defines the terms and conditions for its operation. Sessions and other

Chapter 4: Requirements for a new Framework for DPE Security

information objects are mapped onto service objects¹⁰, which present an interface to the client, through which they operate; the client is not concerned with the internals of the object (i.e. its implementation is of no consequence to the client). Figure 4-1 [15] and the example below illustrate a simple case of two users using a service in a provider domain.

Access session related objects provide a framework for offering secure and personalised access to services and for supporting mobility. The **Initial Agent (IA)** is the initial contact point for the **Provider Agent (PA)** wishing to interact with the provider, and is used to gain an access session with the **User Agent (UA)**. The **Provider Agent** and **User Agent** objects interact within a secure and trusted relationship between the user and the provider (an access session). They support authorisation, authentication and customisation of the user's service access and provide a secure mechanism for starting and joining sessions. In terms of the access session, the user domains take access user roles; the provider domain takes an access provider role. The access session related **User Application (as-UAP)** provides the user interface for the user to interact with the provider. It interacts with the **Provider Agent** to perform user requests, e.g. to establish an access session, and use services.

¹⁰ Precisely these are known in TINA syntax as Components. Components reside in a computational space and are not deployable yet maintain the characteristics of objects. For ease of understanding the term object has been used throughout to mean both computational components and technology objects.

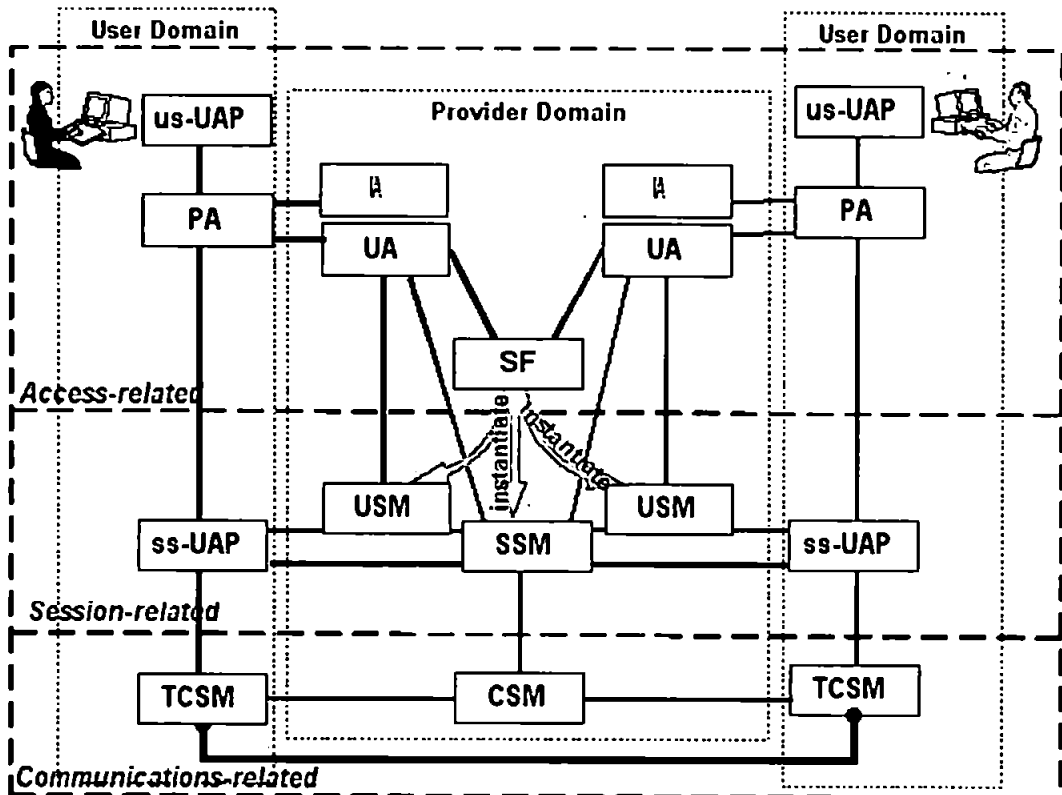


Figure 4-1 TINA Service Example

Service session related objects provide a framework for defining services, which can be accessed and managed across multiple domains. In the provider domain, **Service Session Managers (SSMs)** and **User Service Session Managers (USMs)** are instantiated by **Service Factories (SFs)** based on requests from User Agents. A Service Session Manager and User Service Session Manager provide session control capabilities — a Service Session Manager supports those shared among the users, and a User Service Session Manager supports those dedicated to a user. The **service session related User Application (ss-UAP)** in the user domain allows a user to interact with a service session and acts as an end point for session control.

The communication session related objects provide end-to-end connectivity. Figure 4-1 (based on session example from 18) shows a **Communications Session Manager (CSM)**, using a **Terminal CSM (TCSM)** to establish a stream binding between two stream interfaces on the users' User Applications.

Although recognising the need for security, the TINA architecture does not provide a security framework. It provides some notion of authentication and authorisation functionality in the access session, but there is no detailed or formal specification on the topics. For example, although a user profile is a recognised information object in the TINA information model, there is no specification of what data needs to be held to represent an authenticated user in the system, e.g. the user's security attributes, so that they can be propagated through the distributed system. The issue of inter-domain authentication and authorisation is not addressed, e.g. the use of TTPs and attribute propagation. The other security facilities of integrity, confidentiality, audit and non-repudiation are not addressed by any of the service objects. Indeed, there are no service objects specified for security, the security function is just listed as part of the existing service objects such as the Initial Agent, Provider Agent and User Agent, and there are no interface definitions to assist system developers when building these objects. Indeed, it is this type of ambiguity that has led to confusion and proprietary solutions, which has hampered interoperability between product vendors, and service providers [89].

Security management is mentioned in the management architecture as part of the FCAPS framework. In relation to security, it requires that FCAPS functions be considered in the service architecture [15] under the description of management

Chapter 4: Requirements for a new Framework for DPE Security

contexts. FCAPS defines the rules that govern particular management functional areas during a session, for example the accounting management context might contain information such as the tariff structure, which would calculate charge/charging-rate for a service. TINA also mentions management policies, i.e. a set of rules governing a particular management function in the domain that is associated with the policy. However, its specifications do not address what security contexts or policy configurations are required.

Security is also limited to the access session. It is not mentioned within the service session. This is an unrealistic expectation, because it assumes that security is only considered when the service is initially accessed by a user, and does not account for the times when security requirements may change during the life of the service, e.g. if a video conference session has several participants, and during the session the conference leader, and therefore controller, leaves and hands his responsibility over to another participant. In this case the security service would need to verify that the new leader is authorised to take this responsibility and update his profile.

Scalability is not seen as a problem because the TINA system is designed to be scalable through its use of service objects. It is accepted that this has not been proven and the scalability issue has only been addressed by means of modelling techniques.

Secure interoperability has not been approached. The TINA architecture states that it should be able to inter-operate with non-TINA systems. However, since it does not address a full TINA security model, it is impossible to evaluate how it can integrate with other existing security models.

Chapter 4: Requirements for a new Framework for DPE Security

Security within the DPE itself has not been addressed. There is no mention of how the DPE will be secured or how interaction with DPE services will be accomplished securely. The issue of a distributed TCB is not mentioned, and there is no specification of how applications will interact with DPE security.

The DPE's failure to meet the specified security requirements is summarised in table 4-2 below. Simply stated, TINA does not provide a security framework to protect ISE services.

No.	Security Requirement	Addressed in TINA
1	Identification and Authentication	Yes – but limited
2	Authorization & Access control	Yes – but limited
3	Propagation of security attributes	No
4	Secure communications	No
5	Secure stored data	No
6	Secure Auditing	No
7	Non-repudiation	No
8	Security Management	No
9	Interoperability	No
10	Scalability	Yes – use of objects
11	Integration with existing environments	No
12	System Recovery	No
T1	Intrusion Detection	No
T2	Management of hardware/software protection	No
M1	Secure, authenticated inter-object communications	Yes – but limited
M2	Secure operation of TINA services	Yes – but limited
M3	Secure control of TINA services	Yes – but limited
M4	Secure administration of TINA services	No
M5	Inter-DPE security	No
A1	Secure participant interaction with applications	No
A2	Secure, usable administration of applications	No
A3	DPE security for security in-active and security active applications	No

Table 4-2 Summary of TINA vs. DPE security requirements

4.4 A New Security Framework for DPEs

The following defines a new security framework for DPEs using syntax and semantics in line with TINA specifications. The operational service objects and their interactions within the service architecture are described first. Issues relating to implementation and deployment are also addressed. The management framework is then defined, along with the information structures required by the framework.

4.4.1 DPE Security Service Overview

Twelve new operational-level security service objects, illustrated in figure 4-2 below and functionally described thereafter, have been identified.

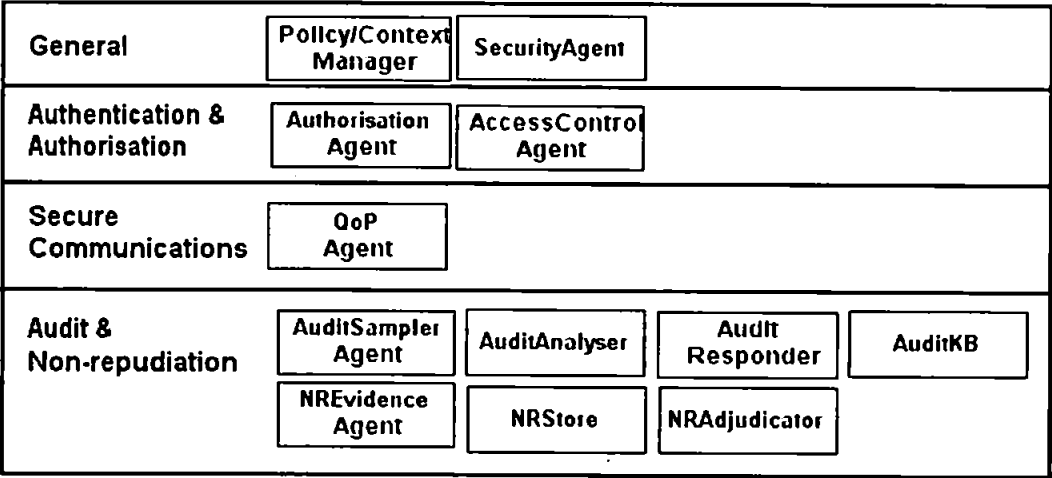


Figure 4-2 A New Security Framework for DPEs (Operational)

4.4.1.1 General Layer

The general layer has two security objects

- **Profile/Context Manager (PCM):** The PCM retrieves any security management data required by the operational level security objects. It will

access security policy information, facilitating security service object actions. For example, the Profile/Context Manager can retrieve authentication data to validate a user logging into the system. It is used to control user interaction with the security management data and objects.

- **Security Agent (SecA):** The Security Agent acts as an initial point of contact with the security system, and therefore plays a role in the access session. It is not required in the service session because the objects used here are considered part of the TCB and the user will not have direct interaction with these service objects. Security Agent controls the user interaction by preventing a user directly interacting with security service objects.

4.4.1.2 Authentication and Authorisation Layer

The Authentication and Authorisation Layer has two security objects:

- **Authorisation Agent (AuthA):** Encapsulates the authentication process for the TINA system. Authorisation Agent authenticates the user with the authentication data presented by the user, via the Provider Agent and Initial Agent, and validates it against the policy data retrieved by the Profile/Context Manager. Authorisation Agent is TINA service independent and security mechanism independent. It is also responsible for initiating the creation of the user's security context, and instantiating it with the appropriate identity and privilege security attributes.
- **Access Control Agent (ACA):** When a user makes a separate request, the security access control is handled by the Access Control Agent. It is based in

the provider domain, because access is a server-side issue in ISE. It takes the user's security context information and uses it to compare to the access control policy (via the Profile/Context Manager), to decide whether the user is authorised to make the service request. If authorised, the user is allowed to proceed with the access session. The Access Control Agent can also be used by the service-session objects, when they are requesting a new service that may be restricted.

4.4.1.3 Secure Communications Layer

The Secure Communications Layer has one security object:

- **Quality of Protection Agent (QoPA):** The Quality of Protection Agent is responsible for providing secured communications between service objects. It is able to compare the QoP policies, via the Profile/Context Manager, of the communicating parties and then decide what QoP will be implemented. The Quality of Protection Agent is also utilised by the audit and non-repudiation facilities.

4.4.1.4 Audit and Non-Repudiation Layer

The Audit and Non-Repudiation Layer has seven objects:

- **Audit Sampling Agent (ASA):** The Audit Sampling Agent is deployed throughout the TINA system and is responsible for collecting any audit data. It is responsible for deciding whether an event is security relevant. If it is, then the appropriate data is retrieved and forwarded to the Audit Analyser.

- **Audit Analyser (AA):** The Audit Analyser analyses the information sent by the Audit Sampling Agent to decide if the event is anomalous. It indicates the analysis result (i.e. whether a system violation has occurred or whether suspicion levels should be raised) and produces an analysis token; the latter is used to provide a full justification of the analysis results, if required.
- **Audit Knowledge Base (AKB):** The Audit Knowledge Base stores all data related to the auditing process, which includes audit sampling records (i.e. the audit trail), audit analyser tokens, audit responder actions, and any profiling and analysis data used to identify any anomalous behaviour.
- **Audit Responder (AR):** The analysis result and token are then sent to the Audit Responder, which decides what to do. The responder has two basic types of response – saving the data to a specific audit log or producing some alarm. The alarm may be sending an email or screen message to an administrator, or it may involve a partial or complete system shutdown/lockout. These responses can be constructed using service objects.
- **Non-Repudiation Evidence Agent (NREA):** Whenever the NR security policy defines that evidence is required, e.g. proof of receipt, the Non-Repudiation Evidence Agent will be able to generate/verify evidence token for the appropriate service object.
- **Non-Repudiation Store (NRS):** The Non-Repudiation Evidence Agent interacts with the Non-Repudiation Store also, to store evidence tokens when required.

- **Non-Repudiation Adjudicator (NRAdj):** The Non-Repudiation Adjudicator represents a notary that can make judgements on any disputes. A TTP will be used to verify evidence and then prove/disprove claims made by clients or servers. It provides the following capabilities:
 - specify the tokens and identities of the disputing objects;
 - return a decision and the supporting token (i.e. the token that validates the decision).

The adjudication process has two phases – the first is an on-line adjudication. The on-line adjudication allows the adjudicator (without any human intervention) to validate the evidence tokens, i.e. make sure they have valid signatures and that the times are correct. If one-evidence token is found to be invalid, then the process will be able to settle the dispute by deciding in favour of the valid token holder. However, if both tokens are valid, then one of three options is possible. If the Non-Repudiation Adjudicator is implemented as an expert system, then it may still be able to settle the dispute based on some existing rules it contains. If not, it can either signal for human intervention and request assistance in the adjudication process or it can return a judgement of “*undecided*”.

4.4.2 Realisation and Deployment Issues

There are several issues within the Computational model of the new DPE security framework that should be addressed before proceeding to the service example, as they

will help clarify the service example presented in the following section. Figure 4-3 below is used to help illustrate these issues (based on service example in 18).

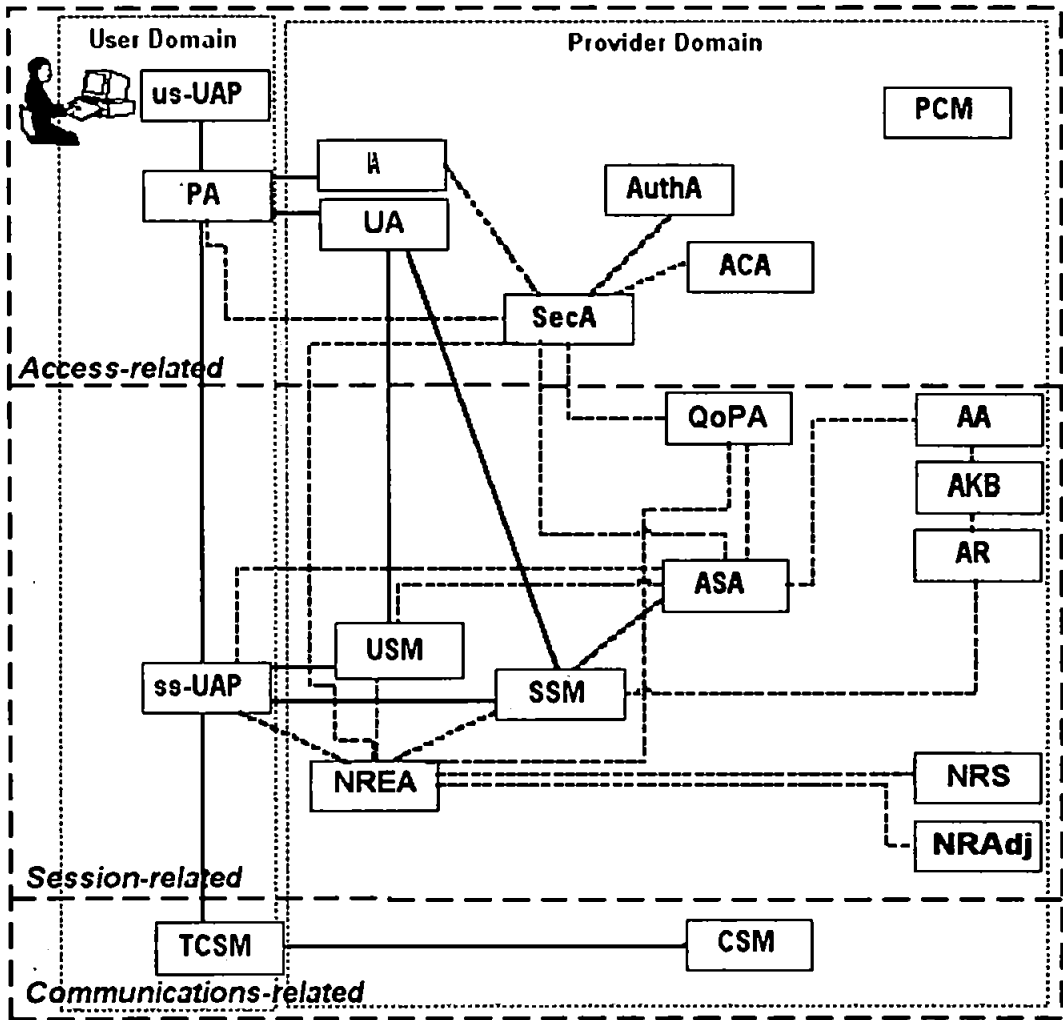


Figure 4-3 Example of Security Service Object Deployment

4.4.2.1 Absence of data storage objects

The TINA service example (see section 4.3) has a notable absence of any data storage objects. Such specification is often left to the information model. However, in the

security framework it was deemed necessary to identify the Audit Knowledge Base and Non-Repudiation Store within the model for the following reasons:

- The data stored in the Audit Knowledge Base and Non-Repudiation Store is very important to the overall framework and its security. The audit information will help identify intrusions and the non-repudiation evidence tokens are required for adjudication.
- The Audit Knowledge Base stores a variety of information, and therefore will present a variety of interfaces to the Audit Analyser, Audit Responder, and Audit Sampling Agent.
- The data in both repositories needs to secure data, both in transit and in storage. Therefore as service objects both the Audit Knowledge Base and Non-Repudiation Store will be able to utilise security mechanisms to secure the data internally, and they will also have security policies that will allow them to negotiate security contexts with any clients, e.g. the Audit Analyser and Audit Knowledge Base will use the Quality of Protection Agent to create a secure communication channel between them so that audit records can be transported securely over the network.

4.4.2.2 DPE service object placement in relation to the DPE Node

It was previously stated that DPE services, such as security, are not necessarily present on all DPE nodes. However, it is recommended that any DPE node involved in security would have most of the security service objects available locally, to reduce the overhead of accessing remote service objects. The exceptions to this rule are the

Audit Knowledge Base, Audit Responder, Audit Analyser, Non-Repudiation Store and Non-Repudiation Adjudicator. The reason for this is that all the other agents are involved in almost every service – authorisation, access control and secure communications are usually requirements in a protected system. The audit and non-repudiation functions are not always required, because they generally incur a high overhead. However, if required the Audit Sampling Agent and Non-Repudiation Evidence Agent will be frequently utilised by the service objects and so should be available locally.

4.4.2.3 Availability of security service objects in session model

The security service objects are not restricted to any single session type; their availability will be dependent on the security service requirement within each session. The access session is generally concerned with authentication and access control, therefore the Security Agent, Authorisation Agent, Access Control Agent, and Profile/Context Manager are available. Secure communications can be required in both the access session (to secure the authentication process) and service session (to secure the TINA service), and, therefore, the Quality of Protection Agent is available to both. The CSM is considered outside the scope of the DPE security model. However, the Service Session Manager in the service session can utilise the Quality of Protection Agent when requesting the CSM to provide a secured communications stream. The audit and non-repudiation service objects are available in the access or service session depending on the security policy requirements.

4.4.3 DPE Security Management Overview

The new security framework separates the management of services and mechanisms and thereby provides the required mechanism-independence. It involves the following steps:

- definition of new policy classes to separate management function;
- use of opaque data types to assist abstraction;
- definition of policies for all security functions for consistency;
- ability to locate the new policies;
- ability to handle security active/inactive policies.

4.4.3.1 New Policy Classes

A new **Policy** superclass is defined, see figure 4-4 below. It will have two derived classes, **ServicePolicy** and **MechanismPolicy**, to administer security services and security mechanisms respectively.

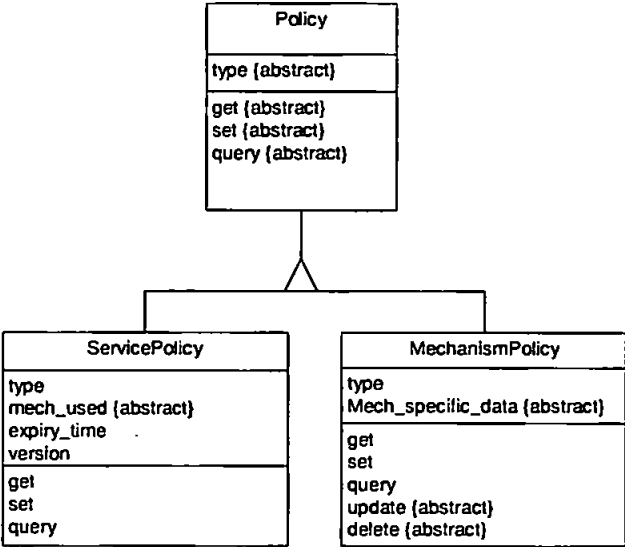


Figure 4-4 Administrative Policy Class

The Policy superclass has a single data attribute *type*, which identifies the object(s) or service objects to which the policy applies. Policy defines three abstract operations, *get*, *set* and *query*, used to maintain policy data. *get* retrieves a single administration record, based on a known identifier, while *set* updates a single administration record based on the identifier. *query* is able to retrieve a complete set of records (a ‘bulk’ implementation of the *get* operation – a standard ‘multiple record *get*’ optimisation used in OO programming). The *query* method also allows the user to use any data to select the records. For example, if the *get* method used on a video conferencing object, retrieved the identifier and reference to the videoconference, the user could then use the *query* method find out the details of the conference, e.g. bandwidth required, etc, to see if he could request access to the conference. ServicePolicy has three data attributes. *mech_used* specifies the mechanisms to be used implementing the policy. This is an identifier that refers to an instance of the MechanismPolicy

class. The *expiry_time* and *version* specify the expiration time of the policy and its version. Operations to update/delete entries are not listed, as some policies will not want this to occur, e.g. non-repudiation policies. Policies that do require such operations will add them to the their own class definition, which inherits the *ServicePolicy* class. The *MechanismPolicy* object has one data attribute, *Mech_specific_data*. This holds mechanism specific information, and thereby provides the means for mechanism (technology) independence. The *MechanismPolicy* also has *remove* and *update* operations.

4.4.3.2 Abstraction through generic data types

To preserve mechanism independence data items that are considered opaque data types should be used, because their internal structure has no significance to the interface or the caller, but has meaning to the underlying mechanism, e.g. GSS-API and CORBA use such data types. Another useful method of abstraction is the use of codes to indicate generic and mechanism specific errors (the use of exceptions can be implementation specific, e.g. C++ use of native exception handling). If required, the codes can be divided in to major (generic) and minor (mechanism-specific) structure to preserve the mechanism independent nature of the service, while still passing useful mechanism-specific failure information. Also another reason that exceptions are not used is because, by standard OO programming practice, they should indicate an exceptional circumstance that only occurs between 5-10% of the time. In this case the errors can be a valid response that occurs on a regular basis, e.g. security credentials expiring or corrupt signature.

4.4.3.3 Consistent Management Structure

Providing a structured management system that operates across all the security facilities ensures consistent management. Currently, there is no standardised administration for each of the DPE security facilities. No management contexts or policies are defined. Therefore the policy and mechanism administration service objects should be applied across all of the security facilities to provide a comprehensive and coherent administration structure.

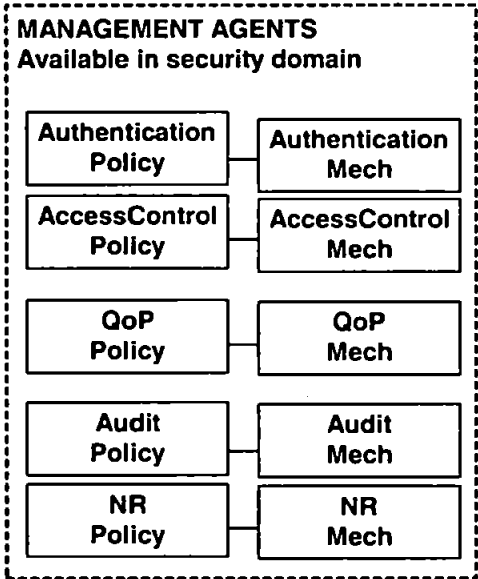


Figure 4-5 Security Service Objects - Management

- **AuthenticationPolicy (AuthPolicy) & AuthenticationMech (AuthMech):** Two objects now administer the authentication process. Authentication Policy is responsible for holding the following:

Chapter 4: Requirements for a new Framework for DPE Security

- identifying an access session User Application and the authentication mechanism associated with it, e.g. the video conferencing service may require smartcard authentication;
- authentication policy for a user, i.e. it identifies the authentication mechanism allowed for a user and the associated authentication data – for password identification it identifies the user's ID and password;
- security identity, which is used to create a user's security identity context that can be propagated throughout the TINA system.

Authentication Mechanism administers the authentication mechanisms. It holds data relating to the mechanism name and its object identifier within the system. The ISO/IEC specifications for Abstract Syntax Notation One (ASN.1) [90] and Basic Encoding Rules (BER) [91] are used to define what mechanism is used. For example, the Object Identifier 1.2.840.113554.1.2.2 identifies the Kerberos V5 mechanism. It may also identify the authentication data types required, e.g. character string is required for password authentication.

- **AccessControlPolicy (ACPolicy) & AccessControlMech (ACMech):** The access control service-level administration, AccessControlPolicy, includes the following data:
 - identification of the access control mechanism used for the service;
 - user privilege attributes that will be used to create the user's privilege attribute context to be propagated through the system (this will be linked with the

Chapter 4: Requirements for a new Framework for DPE Security

user's identity context), and will be used to evaluate whether a user is authorised to access a particular service. This will also include the delegation status of the credentials, i.e. which, if any, of the attributes that can be delegated through the TINA system;

- provider-required privilege attributes, which identify what privileges are required for a user to access the provider's service (the user's privilege attribute context will be evaluated against this).

The ACMech holds the data relation to the administration of the access control mechanisms. It identifies the mechanisms used, along with other implementation information such as version and expiry date.

- **QualityofProtectionPolicy (QoPPolicy) & QualityofProtectionMech (QoPMech):** Integrity and confidentiality management are handled by QOPPolicy and QOPMech. The QOPPolicy is responsible for the following data:
 - the QoP to be used, i.e. whether integrity, confidentiality or both are required to secure data;
 - identifying what mechanisms are required by a particular service object. This should identify the required mechanisms and other supported mechanisms that may be used instead, to provide flexibility;
 - for transit data, how much of a message will be encrypted, e.g. the whole message or just selected portions.

Chapter 4: Requirements for a new Framework for DPE Security

The QoPMech lists the mechanism and its object identifier. It will hold any other mechanism specific information, such as the location of keys, version number etc.

- **AuditPolicy & AuditMech:** The system audit information is administered by these objects. AuditPolicy manages the following data:
 - event selectors identify what event is considered security-relevant and what details of the event need to be recorded, e.g. an service access request should be audited and the selectors include the time, the user making the request, the service requested and whether the request failed or succeeded;
 - Audit Analyser and Audit Responder used. There may be several analysers and responders available, each possibly using different mechanisms or used for different sub-domains;

AuditMech is used to administer the entire audit mechanisms, Audit Analyser, Audit Responder, and Audit Sampling Agent. It will hold mechanism specific details, such as location, object identifier etc.

- **NRPolicy & NRMech:** *NRPolicy*, responsible for non-repudiation administration, specifies the following:
 - the type of evidence required by a service object for a particular request;
 - the QoP for the delivery of non-repudiation evidence;
 - the storage object where evidence will be held;

Chapter 4: Requirements for a new Framework for DPE Security

- the accepted authorities (includes adjudicators and delivery authority);
- the NR mechanism to be used to generate the evidence.

NRMech specifies the mechanism-specific details for the non-repudiation mechanisms.

- **Policy/Context Manager (PCM):** One further service object that can be included in this model is the Profile/Context Manager (already specified as an operational service object), which is used as the point of contact between the operational and management models. It is responsible for finding the appropriate administration object within a domain, retrieving security management information from policies and building security contexts with this data. The Profile/Context Manager abstracts the DPE-defined contexts from the actual security service implementations.

4.4.3.4 Facilitating Security Active/Inactive Applications

The security requirements analysis (see section 4.2.3) outlined the need for security-active and security-inactive applications. This implies that two security policies could exist for a single application, one as a domain default to handle all applications (both active and in-active) and then the particular security-active policy for an application. Therefore the management system needs to be able to administer the separate policies and define the rule of operation when two conflicting policies exist. The issue of allowing two policies to exist is addressed by identify the policy type, e.g. the policy

type can be defined as active or inactive. The second issue is more complex and will be addressed in more detail when interoperability is discussed in Chapter 5.

4.5 DPE Secured Service Example

The example presented in this section illustrates a service session that is secured using the new security framework. The following assumptions are made:

- only one user and one provider are involved;
- only one security domain is involved (even though the figure shows User/Provider domains which would normally be different security domains – the issue of interoperability between security domains will be addressed in the following chapter);
- not all service object interactions are shown, in order to simplify the illustration, e.g. Profile/Context Manager interactions to build contexts are not shown.

The video conferencing service example, which is based on the standard TINA service architecture example [15], presents two security relevant scenarios. The initial TINA scenarios assume that all the operations are successfully completed (no error, no fault, and no rejection) for simplicity. Some of the alternate outcomes will be outlined in each section. The example also does not address the issue of secure interoperability between disparate domains; it assumes that the security technology

and policies are compatible. Inter-domain secure interoperability will be dealt with in chapter 5.

4.5.1 Logging in to the Provider

This example shows userA establishing an access session with their named user agent of the provider. The user wishes to make use of the provider's services, which the user has previously subscribed to.

Preconditions:

The user has contacted the provider, and the Provider Agent has an interface reference to an Initial Agent of the provider.

The new security preconditions required:

- security is available in both the user and provider domains;
- user A is defined as an authorised user in the provider domain;
- a QoPPolicy is available for the Provider Agent and User Agent;
- there are no audit or non-repudiation policies related to the login process
- the user and provider domains are in a single security administration domain and therefore secure interoperability is not an issue.

Scenario:

(The new security interactions are steps 3, 4, 5, 6, 7, 8, 10, 12 and 13.)

Chapter 4: Requirements for a new Framework for DPE Security

1. User A uses an access session related User Application to login to the provider, as a known user. The access session related User Application requests the user authentication information such as the UserID and password. The user then requests the Provider Agent to login to the provider, as a known user. The access session related User Application supplies the security information to the Provider Agent.
2. Provider Agent requests that an access session is set up with the named User Agent of the user. Provider Agent provides the username of the user to the Initial Agent.
3. The Provider Agent sends the security information to the Security Agent in the user domain, Security Agent^U.

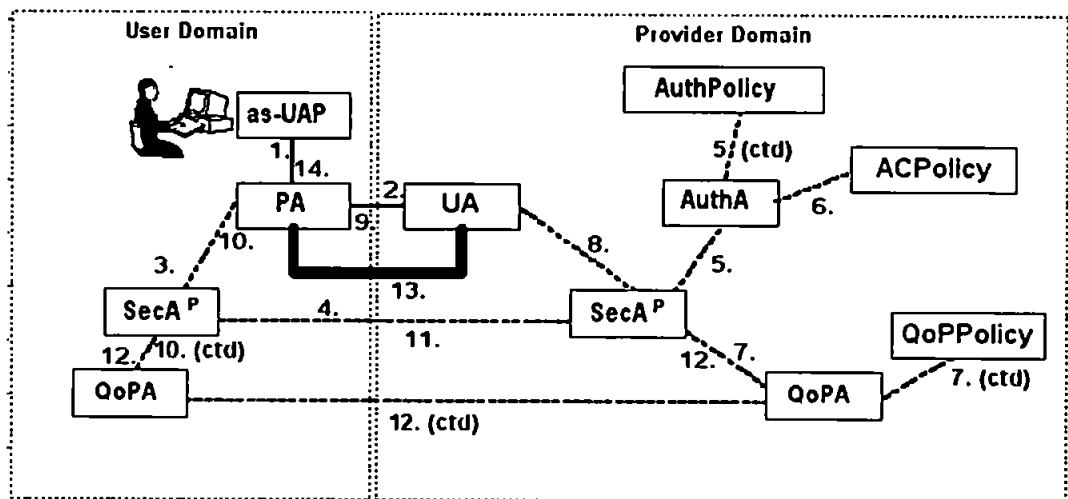


Figure 4-6 New Secure Login Examples

4. Security Agent^U sends the information to Security Agent in the provider domain (Security Agent^P), to authenticate the user.

Chapter 4: Requirements for a new Framework for DPE Security

5. Security Agent^P contacts Authorisation Agent, which retrieves the authentication policy from AuthPolicy to authenticate the user.
6. Once authenticated, Authorisation Agent also retrieves UserA's privilege attributes from ACPolicy, to create the user's security context.
7. Security Agent^P contacts the Quality of Protection Agent, to see if secure communication is required with the user domain; Quality of Protection Agent, finds the appropriate QoPPolicy, and returns the information to the user's security context.
8. The completed security context is associated with the User Agent.
9. An access session has been established. It returns the interface reference of the user's User Agent.
10. Provider Agent retrieves its security policy information, via the Security Agent^U to see if secure communication is required.
11. Provider Agent sends information about the user domain to the User Agent. This information is termed the Provider Agent context, and will include the security context information such as the QoP required by the Provider Agent.
12. In this example, both domains require secure communications, i.e. integrity and confidentiality. The Provider Agent and User Agent request the Quality of Protection Agents in both domains, to negotiate what mechanisms are to be used, via the security context information, i.e. they

Chapter 4: Requirements for a new Framework for DPE Security

find a common set of mechanisms to use for both integrity and confidentiality, such as MD4 and 3DES.

13. A secure communication channel is established between Provider Agent and Initial Agent.

14. Provider Agent returns success to access session related User Application.

Post-conditions:

User has setup an access session between the Provider Agent and named User Agent. The named User Agent is personalised to the user, and has knowledge of interfaces of the Provider Agent.

Any interface references of the Initial Agent held by the Provider Agent will be invalid.

The new security post-conditions required are:

- a Provider Agent context containing security information has been created;
- a security context containing UserA's security information is associated with the User Agent;
- a secure communication channel exists between Provider Agent and User Agent.

Alternatives within scenario:

There are several alternatives available within this scenario, of which two key issues are listed:

Chapter 4: Requirements for a new Framework for DPE Security

- The user may have been unknown. The authentication could have failed at step 6 or the user could have been logged in with guest privileges. The outcome would be dependent on the domain policy in relation an unrecognised user.
- The known user may have supplied incorrect authentication information, e.g. an invalid password. In this case the login would have failed at step 6 and the session would have been terminated. A similar situation would arise for an expired/revoked user ID.

Other alternatives relate to the security policies defined with in the domains, e.g. audit of the authentication process may have been required etc. For simplicity, it was not included in this scenario.

4.5.2 Starting a New Service Session

This example shows a user starting a new service session. The user is assumed to be in an access session with the provider and to have a valid subscription to the service (the service type is web-cast). The service session related User Application is assumed to be present on the user's terminal. Steps 2, 3, 4, 9, 10, 11, 12, 13 and 14 are new security related interactions.

Preconditions:

An access session exists between the Provider Agent (user A) and User Agent (in provider domain). An access session related User Application shows the user the services that can be started.

Chapter 4: Requirements for a new Framework for DPE Security

The new security preconditions are:

- a security service exists in both the user and provider domains;
- the user's has already been authenticated, a security context for the user is available to the provider and is associated with the User Agent;
- a secure communication exists between the Provider Agent and User Agent;
- the provider domain specifies an audit policy that records when a new web-cast session is started;
- the provider non-repudiation policy requires a proof of origin for a new web-cast request;

Scenario:

1. The access session related User Application requests a list of services from the Provider Agent, which the user has subscribed to. The Provider Agent makes the same request to the User Agent.
2. The User Agent contacts the Security Agent^P to see what services the user is authorised to see. Security Agent^P already has the user's security context and needs to compare it to the required attributes for services.
3. Security Agent^P contacts the AccessPolicy, via the Access Control Agent, to see what attributes are required for the list of available services.

Chapter 4: Requirements for a new Framework for DPE Security

4. Once Security Agent^P defines the list of authorised service, i.e. those services which the user has the required privilege attributes for, it returns the list to the User Agent.
5. The User Agent returns the list to the access session related User Application, which displays the list to the user. The user selects a service to start, a recorded web-cast. The access session related User Application requests Provider Agent to start the service.
6. The Provider Agent starts the service session related User Application, associated with this service session, and informs it of the service type that it should start (web-cast).
7. The service session related User Application requests a new service session of service type web-cast, from the Provider Agent. (The service session related User Application may pass information about itself to the Provider Agent, including session models and feature sets supported, and references to its operational and stream interfaces.)
8. Provider Agent requests to start a new service session of the service type (web-cast), to (user A's) User Agent. (It may also pass the information about the User Application.)
9. Before User Agent starts a service, it will contact the Security Agent^P to request that it checks the security policy for that particular service.

10. The Security Agent^P will contact the Quality of Protection Agent, which will use QoPPolicy, to see if there are any secure communication requirements for the web-cast session. In this case there is no QoP requirement.

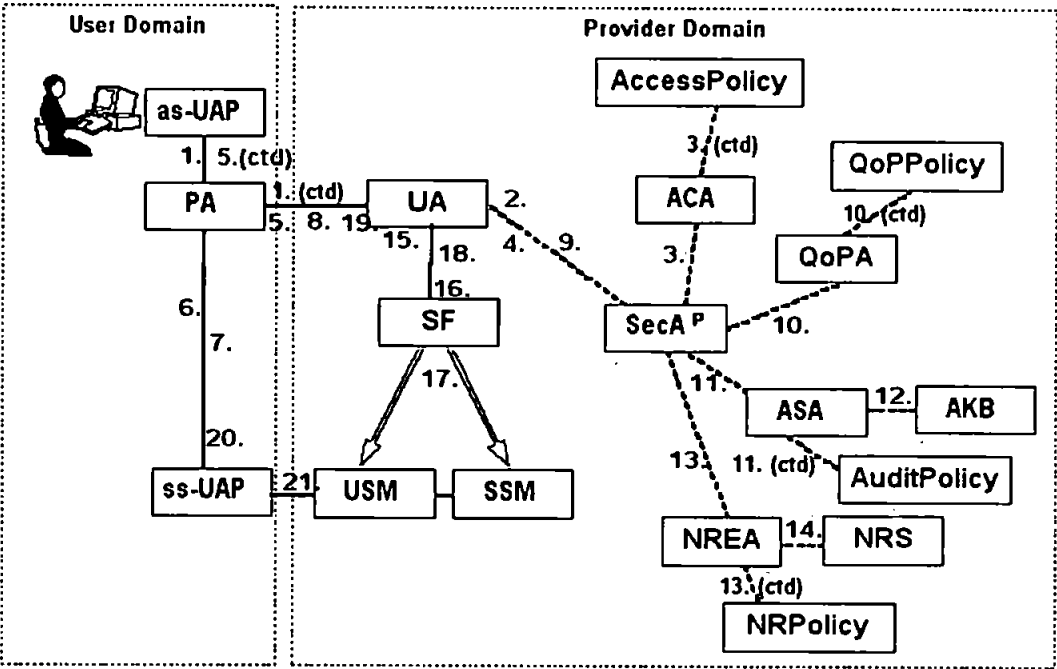


Figure 4-7 New Secure Service Example

11. The Security Agent^P will contact the Audit Sampling Agent, which will use AuditPolicy, to see if there are auditing requirements for a web-cast session. The policy specifies that new web-cast session requests will be audited.
12. In response to this, the Audit Sampling Agent generates an audit record, which is forwarded to the Audit Knowledge Base. No analysis is required, as the policy only requires the event to be logged. (For simplicity, the negotiation of

the secure connection, by the Quality of Protection Agents, between the Audit Sampling Agent and Audit Knowledge Base is not illustrated).

13. The Security Agent^P will contact the Non-Repudiation Evidence Agent, which will use NRPolicy, to see if there are non-repudiation requirements for a web-cast session. Their policy specifies that a proof of origin is required for the request.
14. In response to this, the Non-Repudiation Evidence Agent generates a proof of origin token using the user's security information associated with the User Agent. This token is forwarded the Non-Repudiation Store. (For simplicity, the negotiation of the secure connection, by the Quality of Protection Agents, between the Non-Repudiation Evidence Agent and Non-Repudiation Store is not illustrated).
15. User Agent gets a reference to a service factory, which can create service session objects for the service type (web-cast).
16. User Agent requests that a new session of the service type (web-cast) be created by the Service Factory.
17. Service Factory creates an Service Session Manager and a User Service Session Manager and initialises them. In this case a security context is associated with the Service Session Manager, which describes the services security requirements, e.g. if it has audit and non-repudiation requirements.
18. Service factory returns interface references of the User Service Session Manager and the Service Session Manager to the User Agent.

Chapter 4: Requirements for a new Framework for DPE Security

19. User Agent returns references of the User Service Session Manager and Service Session Manager to the Provider Agent.
20. Provider Agent returns references of the User Service Session Manager and Service Session Manager to the service session-User Application.
21. The service session related User Application and User Service Session Manager (and Service Session Manager) can interact using service specific interfaces or interfaces defined by session models, including the TINA session model. Some interactions between these objects may be necessary before the user can use the service.
22. At this point User A is the only user involved in the web-cast session. Some services may be single user services.

Post-conditions:

A web-cast session is established between the user and provider.

The new security post-conditions are:

- An audit record for the request exists in the provider's Audit Knowledge Base.
- An evidence token for the proof of origin (i.e. the user) exists in the provider's Non-Repudiation Store.

Alternatives within scenario:

- If the User Agent were unable to start the service, it would have raised an exception in step 15 or one of the later steps (e.g. when the Service Factory was creating the Service Session Manager, or User Service Session Manager).

- If the web-cast service were already running, then the audit and non-repudiation policy information would already have been available, via the Service Session Manager's security context.

4.6 Summary

This chapter has specified the DPE security requirements through analysis of the current literature and providing a new definition of the DPE security problem domain. As a result, a new DPE security framework has been defined. It provides a two-tier model that addresses the operational and administrative needs of a DPE environment. The operational service objects provides all of the ISO defined security services, and the management objects provides the flexibility and consistency required by using mechanism-independence, abstraction and a complete set of management objects to administer all of these services. As the framework has been defined using service objects, it is scalable and can be easily distributed across the DPE.

However, there are still two major issues that have to be considered, namely interoperability, and the interaction between the security framework and the existing DPE services, such as the trader. Both of these topics are covered in detail in the following chapters.

5. Secure Interoperability in a DPE

5.1 *Introduction*

A fundamental characteristic of a distributed system is that physical nodes and distributed objects require interoperability. It facilitates interaction between entities resident across heterogeneous platforms, where such entities may be implemented using different technologies or different paradigms. Although computers can be networked, this does not imply interoperability has been achieved. Interoperability has to solve the challenge of differences in protocols, data formats, programming languages and, paradigms. For example, although it may seem a simple task to provide a graphical front-end interface to a legacy mainframe system, interoperability obstacles have to be addressed including: the front and back-ends operating in two different paradigms, (object-oriented and procedural) and data conversion and manipulation so that it can be understood by each system. Even within the Internet, where currently millions of nodes are connected and are able to interoperate, new and more optimal solutions are still being sought to support distributed transparency, for example the Simple Object Access Protocol (SOAP) [92]. SOAP defines an Remote Procedure Call mechanism, using HTTP as the transport and XML documents for encoding requests and responses, in order to provide an object invocation mechanism built on standardised Internet solutions.

For DPEs, CORBA provides one interoperability solution using a combination of brokers (ORB) and language independent IDL interfaces (see Section 2.4)¹¹. CORBA also supports interoperability to other non-CORBA distributed systems, such as COM [93]. Other organisations, are now explicitly addressing interoperability with their frameworks, e.g. Microsoft committed one quarter of its budget in 1999 to interoperability [94, 95]. However, these solutions currently only address insecure communications. When security requirements are added a significant increase the difficulty of the task is noted. Although secure interoperability is not a new topic in DPEs, this chapter deals specifically with the issues currently encountered and presents proposals on how they can be alleviated with the New Framework previously described (see chapter 4).

5.2 DPE Secure Interoperability Requirements

Secure Interoperability within and between DPEs can be described as the ability to provide a secure association between a client and target even when they exist in different security domains.

In addressing secure interoperability, the requirements need to be established. Table 5-1 below summarises the DPE security requirements specified in the previous chapter. These requirements will be evaluated to see how they apply to secure DPE interoperability.

¹¹ The OMG has even sponsored CORBANet, a research project at the Distributed System Technology Centre in Australia, whose specific function is to demonstrate interoperability between different ORB vendors [11].

No.	Security Requirement
1	Identification and Authentication
2	Authorization & Access control
3	Propagation of security attributes
4	Secure communications
5	Secure stored data
6	Secure Auditing
7	Non-repudiation
8	Security Management
9	Interoperability
10	Scalability
11	Integration with existing environments
12	System Recovery
T1	Intrusion Detection
T2	Management of hardware/software protection
M1	Secure, authenticated inter-object communications
M2	Secure operation of TINA services
M3	Secure control of TINA services
M4	Secure administration of TINA services
M5	Inter-DPE security
A1	Secure participant interaction with applications
A2	Secure, usable administration of applications
A3	DPE security for security in-active and security active applications

Table 5-1 Requirements for DPE Security

Interoperability is listed as a requirement for secure DPEs. As DPEs are distributed, it would be unreasonable to assume that all objects would be distributed within a single security domain. Therefore all inter-domain communications still have the same security requirements as intra-domain communications, and many of the issues have already been addressed in chapter 4. Authentication and Authorisation (Requirements #1 & #2) stated the need for inter-DPE authentication, i.e. the ability to authenticate through a TTP and the need to allow access to remote clients (see section 4.2.2).

Chapter 5: Secure Interoperability in a DPE

Auditing, non-repudiation, secure communications and storage of data are still required (Requirements #4-#7). However, interoperability of these services now highlights new requirements.

REQUIREMENT #I1: The trust relationship between two disparate security domains has to be established. Authentication and authorisation already require some form of trust to exist between two entities or services. However, in the case of interoperability, the trust model (see section 3.3.1) needs to be defined between the domains.

REQUIREMENT #I2: Attribute mappings need to exist between disparate domains. Propagation of attributes constitutes a more complex problem in an inter-domain scenario; this is because the attributes in two domains may differ, and therefore a mapping needs to exist between the attributes so they can be converted when necessary.

REQUIREMENT #I3: Middleware sub-domain interoperability requires secure operational-level interaction. As the middleware sub-domain is responsible for operational services, it needs to ensure that compatible mechanisms can be used for secure inter-domain DPE interactions.

REQUIREMENT #I4: Middleware sub-domain interoperability requires secure control and administrative interaction. As the middleware sub-domain is responsible for control and administration of services, it needs to ensure that the policy configurations of services are compatible for secure inter-domain DPE interactions.

REQUIREMENT #15: Application sub-domain interoperability requires the negotiation of a secure inter-domain context. As the application sub-domain is responsible for establishing a secure context between a target and client, it needs to begin the negotiation process to allow all domains create the appropriate agreed environment.

5.3 DPE Secure Interoperability – the issues

The interoperability requirements highlight the fact that there are four possible inconsistencies that need to be resolved:

- conflicting *security mechanisms* (requirement #13);
- conflicting *security policies* (requirement #12, #14);
- conflicting *security protocols* (requirement #15);
- different *trust* domains (requirement #11).

These issues and their possible solutions are discussed in the following sub-sections and then summarised in table 5-2 ¹².

5.3.1 Conflicting Security Mechanisms

If two domains are using different security mechanisms, interoperability is a serious issue. For example, if they have differing encryption algorithms (e.g. DES and IDEA)

¹² The issue of different paradigms is not considered because it is assumed that all DPEs will be using object-oriented technology.

it will not be possible for these two domains to provide a secure association between their members because they cannot interact cryptographically.

Solutions to this problem include:

- the provision of a bridge (see section 3.3.3.1) that will perform the appropriate conversion between encryption mechanisms. However, when using this with cryptographic mechanisms, it complicates the situation by adding overheads (due to the decryption and re-encryption of messages) and providing another point of vulnerability between the two domains.
- the use of a standard set of mechanisms and the provision of a protocol to negotiate a common mechanism to be used (see section 3.3.3.2). For example, SSL provides such a facility for secure Internet communication. The drawback for DPEs is that they require access to a common set of mechanisms and a common protocol.
- the use of generic tokens (see section 3.3.3.3). For example, GSS-API provides such a facility, but it does require the definition of the token structure and the use of appropriate interfaces/protocol to utilise the tokens. Definition of a generic tokens can prove difficult as it needs to ensure that the token is truly generic and caters for all protocol requirements; also if opaque data types are used a performance overhead can be incurred due to the data marshalling required.

5.3.2 Conflicting Security Policies

A security policy is defined as a set of criteria for the provision of security services. It defines what is and what is not permitted in the area of security during general operation of a secured system. These criteria can and do differ between domains. For example, a client in domain A has an authentication policy specifying that server-side (peer-entity) authentication is required. A server in domain B has an authentication policy specifying that mutual (peer-to-peer) authentication is required; i.e. the client has to authenticate itself to the server as well. However, the client in domain A has no way of authenticating itself to the server, e.g. if digital certificates were used for authentication, the client in domain A may not possess a certificate. Therefore, in this example, a secure association, that satisfies both security policies, can never be established between domain A and B because the client will never be able to authenticate itself to the server.

A solution to this problem is to allow negotiations between the security policies of the two domains. It requires each domain to define what is supported and what is required by a policy. A policy requirement is a security service function that must be complied with; otherwise a secure association cannot be established, such as mutual authentication in domain B in the above example. A supported policy is one that is available but not necessary for a secure context. The requirements for a DPE in the provision of this solution are two-fold:

- the definition of security policy configurations for the security services between two domains;

- the definition of a protocol to negotiate the policy configuration.

This solution will also have an impact on the mechanisms used. Mechanism-independence (see section 3.3.4) facilitates the negotiation process. It will abstract the policy (security service) required from the mechanism. However, a successful policy negotiation does not necessarily guarantee a secure context can be established between two disparate domains. For example, if domain A requires non-repudiation, but domain B does not support any compatible mechanism with domain A, then secure interoperability will not be possible. The security policy defines the rules of secure engagement between two domains, and therefore it should be negotiated first.

A policy issue that is not dealt with by this solution is the existence of different attributes in disparate domains. For example, domains A and B both support access control using Access Control Lists. However, domain A defines different user roles and access rights to domain B. There are three possible courses of action.

- users can be logged in with restricted privileges, as their attributes are not recognised in the foreign domain; this solution will restrict the interactions that can occur.
- the administrators of both domains can add the appropriate foreign domain users to their own domains; this adds an administrative overhead, and requires the administrators to have agreed on the appropriate user access rules.
- the attributes could be mapped to appropriate corresponding attributes in the foreign domain; this approach also requires a certain amount of upfront

agreement/trust between the domains, but it considerably reduces the administrators' subsequent overhead while still providing interoperability.

5.3.3 Conflicting Security Protocols

A security protocol is used to establish a security context between the client and server and facilitate the secure association once it has been created. If a client and server are using different security protocols, they will never be able to agree on the security context that needs to be used because they will not understand each other.

The problem for DPEs is that there is currently no adequate security protocol defined. For example, the OMG solution is the Internet Inter-Orb Protocol (IIOP) and its secure version Secure Inter-Orb Protocol (SECIOP) [96], however it does not address all of the DPE requirements. Some protocols are functionally restricted, e.g. SSL does not provide any access control mechanisms. Others are environmentally restricted, i.e. they are designed for a particular environment such as Open Software Foundation's Distributed Computing Environment (DCE) [97, 98]. None of the available protocols address all the features mentioned in the sections above – negotiation of mechanisms and policy configurations, and the use of generic tokens.

5.3.4 Different Trust Domains

Interoperability between different security domains within a DPE brings to the forefront issues concerning the use of a distributed TCB (see section 3.3.2) and use of different trust models (see section 3.3.1). If a client and server exist in two separate domains, A and B, and they wish to communicate, although they are considered to be

Chapter 5: Secure Interoperability in a DPE

using a single TCB, it is distributed across two domains and so will have different levels of trust in different objects. For example, while the DPE will trust the security service in its own security domain, it may not trust the security service in the other security domain. Therefore trust has to be established between these objects. The security service in Domain A needs to trust the security service in Domain B is operating in a secure fashion, e.g. if single sign-on is operating between the domains, Domain A needs to trust the following:

- the authentication information that Domain B is holding, in order to login to domain A, is adequately protected;
- the security service in Domain B properly authenticated and authorised the administrator or user that entered the login information;
- the mechanisms used for the security service are trustworthy.

In a DPE, using a distributed TCB, the trust model will affect the amount of interaction required between two separate security domains. The three basic models define the amount of overhead required:

- **No trust:** Mutual suspicion exist and so all services are required, e.g. authentication, authorisation (providing restricted 'guest' privilege) and any user interactions should be carefully monitored and/or restricted;
- **Pre-existing trust:** The number of services may be reduced, e.g. authentication may not be required (user attributes may just be mapped to the local domain attributes), and monitoring may be reduced;

Chapter 5: Secure Interoperability in a DPE

- **Established trust:** The interaction initially requires the authentication process to validate a user via a TTP. Once authenticated, the can be authorised to operate as a trusted user.

Therefore a DPE needs to be able to adapt to each of these scenarios, and judge when each is required. This places three requirements on secure interoperability in DPEs:

- the interaction needs to be recognised as an inter-domain operation;
- the security services needs to be able to configure the security policy accordingly to the requirements;
- the security service needs to be able to adopt the required mechanisms to enforce the security policy.

The possible inconsistencies that may arise, along with their possible solutions are summarised in table 5-2 below.

Scenario	Addressing Secure Interoperability
Inconsistent policy	<p><i>Standard Policy Configurations</i></p> <p>Both domains need to use a common policy to interoperate. The use of standard policy configurations and any mappings to such configurations will overcome the inconsistent policy problem. Another possible requirement is the existence of an attribute mapping facility, between the domains.</p> <p><i>Mechanism-Independence</i></p> <p>Mechanism-Independence implies that security mechanisms and security services (and their governing policies) are managed independently of each other. Therefore it is possible to allow negotiations so that appropriate mechanisms or appropriate services (policies) can be selected to allow interoperation.</p>
Inconsistent mechanism	<p><i>Standard Mechanisms</i></p> <p>The provision of a standard set of mechanisms should ensure that the problem of inconsistent mechanisms does not arise, as both domains should always have at least one mechanism that they both support for the service.</p> <p><i>Generic Tokens</i></p> <p>The use of generic tokens, will allow help abstract security service implementations from the mechanisms used.</p> <p>Both of these facilities help provide mechanism-independence.</p>
Inconsistent protocols	<p><i>Standard Handshake Protocol</i></p> <p>A common protocol that can negotiate the security policy and mechanisms that will be used to provide a secure communication between a client and server.</p>
Inter-domain Trust	<p><i>Trusted Third Parties</i></p> <p>Trust is essential for secure interoperability. The domain administrator needs to know whether he can trust the foreign domain. The use of TTPs will be essential in any inter-domain service, e.g. X.509 Certification Authorities.</p>

Table 5-2 Addressing Secure Interoperability Scenarios

5.4 A New Secure Interoperability Framework

Based upon the previous analysis, a new secure interoperability framework has been defined for DPEs. The framework comprises three new elements:

- the policy configuration structure,
- a security interoperability protocol, and
- a set of security service objects together with their interactions.

5.4.1 New Policy Configuration Structure

The issue of ensuring compatible policy configurations can be used to facilitate secure interoperable control and administration of services, requires a standard policy configuration to exist between both domains (Requirement #14). While certain security policy features have been negotiated in existing protocols, e.g. the negotiation of mutual authentication in SSL, there is no definition of a complete DPE security policy negotiation. The objective is to specify all the services that need to be agreed and the possible options that will be used for these services. The configuration is summarised in table 5-3.

The services should include the full set of ISO 7498/2 facilities: i.e. authentication, access control, integrity, confidentiality and non-repudiation.

- Authentication has one attribute, 'Type', which defines the type of authentication required – 'client', 'server' or 'mutual'. 'client' and 'server' represent peer-entity options where only the client or the server needs to be authenticated.

- Access control also has only one attribute, 'Mapping'. Although the mechanism revolves around the comparison of attributes, there are still innumerable variations on the possible values of the attributes, e.g. 'read, write, execute', 'get, set, mange'. Another problem is the use of administrative aids such as roles, and groups, which increases the number of options exponentially. Although the research initially hoped to define common sets of attributes and possible their groupings (e.g. 'read, write, execute' defined as the 'Unix' attribute set) it was deemed to be an unrealistic solution. There were too many variables, and the set definitions could easily become outdated, and the resulting synchronisation of the sets among different domains would prove difficult. Therefore, a solution was to provide the option to reference a domain mapping object. This object would record the details of the mapping between two specified domains. If the object reference for a particular instance of the domain mapper is provided, the interacting domains will use the mappings specified. If the object reference is 'Nil', then no domain mapping exists and one of the following will occur, either they will authenticate the client and then assign rights, or restricted attributes such as guest-privileges, or the client will already be defined because of previous administrator interaction.
- Integrity and confidentiality services have been aggregated into a single service Quality of Protection (QoP). There are two options. The 'Type' option lists whether no protection or a selection from integrity, confidentiality, DetectMisordering and DetectReplay are applied. The last two options are

included because they can be addressed through time-stamping and sequencing when combined with the integrity and confidentiality algorithms. The second option is 'Message Part', this is defined under the assumption that a DPE protocol will be defined that will allow the QoP mechanisms to be applied in this fashion. If a non-DPE protocol is used, then the message part segment may not be applicable. In an effort to reduce the performance overhead, the policy tries to configure the amount of message sent between the two domains that should be protected. The protocol message is broken up into its constituent parts and the policy defines the portion to be protected, the operation called, the parameter passed, target destination and any other information that might be included, such as transaction related details.

- Non-repudiation is specified by identifying the types of evidence to that are required (see section 3.2.5). Each domain needs to know that the foreign domain can produce the requested evidence; otherwise one of the participants may be vulnerable to repudiation.
- TTP is specified by identifying the role the TTP will play in a service, e.g. a TTP for authentication, or a TTP for non-repudiation delivery authority. A object reference is then associated with the specified role.

Service	Policy Options	Policy Configuration Values
Authentication	Type	Client, Server, Mutual
Access Control	Mapping	Nil, <Reference>
QoP	Type	NoProtection, Integrity, Confidentiality, DetectMisordering, DetectReplay
	Message Part	parameters, operations, destination, info
Non-Repudiation	Evidence Type	Proof of Origin, Proof of Receipt, Proof of Submission, Proof of Delivery
TTP	<Role>	<Reference>

Table 5-3 Policy Configurations

No audit section has been specified because the audit information is considered to remain local to each domain. Each security service will create audit records for any security-relevant event that occur within their own domain, and this information does not need to be propagated across the domain boundaries. There are only two instances when such inter-domain audit would occur. Firstly, if a single audit service is running over both domains. In this case, the audit records could be directed to a single central repository that both domains could access, as a single audit policy would actually be in operation. Secondly, if a security incident, such as an attack occurred, then the administrators from both domains may wish to share audit information to help track the culprit. However, this administrative interaction would generally occur off-line and with the direct assistance of the administrators. It is not required for the inter-domain interaction described here.

5.4.2 New Secure Interoperability Protocol

Based on requirements defined in the previous chapter, the secure interoperability protocol needs to have the ability to:

- provide the appropriate translation ability between domains, e.g. the ability to map security attributes (Requirement #I2);
- utilise compatible security mechanisms (Requirement #I3);
- utilise compatible security policies (Requirement #I4);
- define a secure context through the negotiation of an agreed policy configuration and through the use of compatible security mechanisms (Requirement #I5);
- establish a trust relationship (Requirement #I1).

A protocol has been defined to meet the needs of secure inter-domain DPE interactions. The messages utilised are listed in table 5-4 below.

MessageName	Function
CreateContext	Passed by the client to the target when a secure context needs to be created.
NegotiateContext	Used by the client or target during context establishment to pass further messages to its peer as part of creating the context.
AcceptContext	Returned by the target to indicate that the association has been established.
DeleteContext	Used to indicate to the receiver that the sender of the message has discarded the identified context. Once the message has been sent the sender will not send further messages within the context.
ProcessContext	When a secure context is established, messages are sent within the context using this message.
ErrorContext	Used to indicate an error detected in attempting to establish an association either due to a message protocol error or a context creation error.

Table 5-4 Secure DPE Interoperability Protocol Message Types

All messages will contain a header with an object reference. An object reference identifies the target. It can contain information such as the host (a DNS name or IP address), port number (identifies the port the server is running on), server name where the object resides.

CreateContext will use four parameters, ContextIdentifier, ContextIdentifierType, PolicyConfiguration and Token. ContextIdentifier is a unique identifier created by the client and associated with the context. ContextIdentifierType is used to define describe the state of the identifier; it can be Client, Server, or Peer. A Client identifier is one that is used by the client before a context is agreed; similarly a Server identifier is used by the server; a Peer identifier is one that is used by both client and server once a context is agreed by both. PolicyConfiguration describes the client's policy settings. It defines the mechanisms, policies and mappings configurations, and lists those items

that are required (have to be provided to ensure a context can be created) and supported (other possible configurations available that meet or exceed the context requirements). It is the information held in the SecureInvocationPolicy described in section 5.4.3.1 below (see tables 5-5 and 5-6). Token is used to provide the client with the opportunity to send authentication information to the server, if client authentication is required. The server will use the configuration information specified in PolicyConfiguration to identify the mechanism and policy requirements of the token, i.e. what mechanism is used (e.g. X.509) and what type of policy is used (e.g. mutual authentication).

Once the client has established its parameters with the server, the server responds with the NegotiateContext message. NegotiateContext uses the same four parameters as CreateContext – ContextIdentifier, ContextIdentifierType, PolicyConfiguration and Token. It provides the sever with a means of specifying policy requirements, and also providing authentication information to the client. The NegotiateContext message can be used by both client and server until a policy configuration is found and one or both parties are authenticated, if required. Once negotiation is complete, the server will issue the AcceptContext message with two parameters, ContextIdentifier (an agreed Peer identifier) and PolicyConfiguration (the final and agreed security context configuration).

ProcessContext can contain any message internally and to accommodates this by providing two parameters, ContextIdentifier and MessageBuffer. The message buffer is an opaque datatype that can contain any datatypes thereby allowing the message to hold encrypted, or integrity checked messages; the context is providing the secure

channel between client and server and is not concerned with the content of the message that is secured (this includes any errors returned by the application or server that are not related to a context error); the message will be processed by the DPE.

The DeleteContext message has one parameter, ContextIdentifier and is used by either the client or server to indicate that the context will not be utilised and will be destroyed.

The ErrorContext indicate that an error has occurred in the context, either in the protocol or during context creation, therefore ErrorContext requires 3 parameters, ContextIdentifier, ContextIdentifierType(as the Peer identifier may not be available yet), and Error. Error contains the error data.

The message sequence chart, figure 5-1 below, is used to illustrated how the protocol operates. The interaction takes place between a client and a target that exist in different security domains, and therefore a secure context needs to be established.

The process begins with the client issuing a CreateContext message to the target. It provides the client's requirements for a secure context, i.e. the policy configuration (see section 5.4.1 above) the client requires. The server responds with the NegotiateContext message, which defines the servers' policy configuration. The NegotiateContext message can be used a number of times while the client and target establish the secure context. The server will eventually send an AcceptContext message if it agrees to the context; otherwise it will send a DeleteContext to stop any further interaction and remove the context.

Once a secure context has been established, both client and server use the `ProcessContext` message to send data. A `ErrorContext` message (not shown on chart) is used to show that a protocol or context error has been detected.

When interaction is completed, a `DeleteContext` message can be sent by either client or server. It is acknowledged by a corresponding `DeleteContext` from the other party.

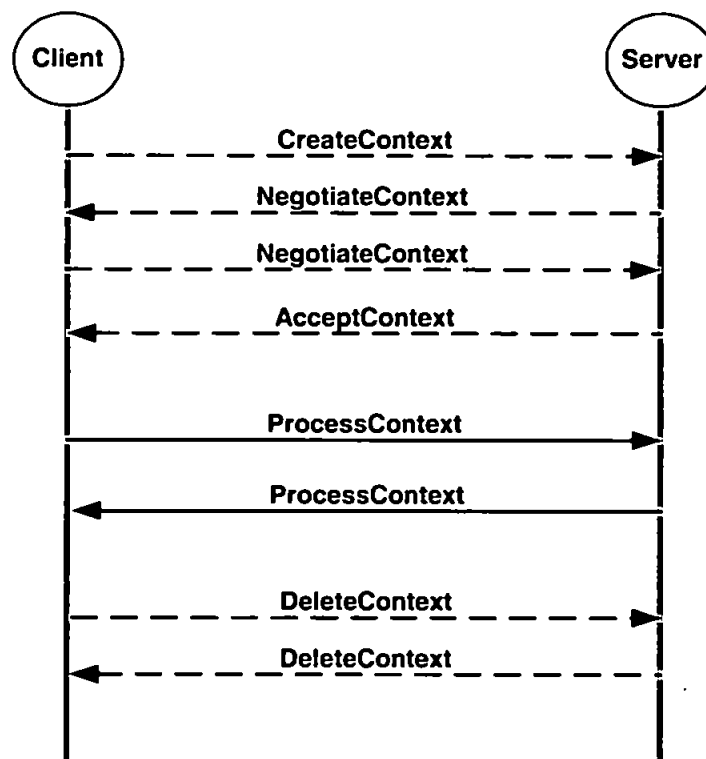


Figure 5-1 Secure Interoperability Protocol Message Sequence

5.4.3 New Secure Interoperability Service Objects

Three new service objects are required to provide the secure interoperability functionality for DPEs: the Security Interoperability Policy (SIPolicy), the Domain

Mapping Agent (DMA), and the Secure Interoperability Agent (SIA). They are described in detail in the following sub-sections.

5.4.3.1 Secure Interoperability Policy

The SecureInteroperabilityPolicy object caters for the negotiation of all security services to establish a security context between a client and target. As such, it holds the configuration information for each of the services specified in table 5-3 above. Therefore, a basic structure is applied to each of these security services to facilitate negotiation, as shown in table 5-4 below. Each component of the structure is then discussed.

Section	Structure
Mechanisms	<p>Required: identifies the mechanisms required for the specified service. This will be the <i>minimum</i> security required by the object for this service.</p> <p>Supported: identifies mechanisms that the object can support, other those that specified in Required. Again, this can be specified for each of the security services.</p>
Policy	<p>Configuration Identifier: identifies the policy configuration being used, be it standard or customised.</p> <p>Date: the date the policy was set.</p>
Mapping	<p>Mapping Identifier: identifies the domain mapping being used on the policy configuration</p> <p>Date: the date the mapping was set.</p>

Table 5-4 SecureInteroperabilityPolicy Structure

- **Mechanism** has a Required/Supported structure for the security services. This will list the required (i.e. minimum) security mechanism to be used and then any other possible supported mechanisms. Both required and supported mechanisms will supply references to the MechanismPolicy objects, so that any mechanism-

specific data required can be accessed from there. The reason that the SecureInteroperabilityPolicy does not access the security services policies, such as Authentication Mechanism (see Section 4.4.3), directly, is because the administrator may wish to enforce only certain mechanisms for interoperability, e.g. he may wish to make inter-domain operations use a higher level of security than that available in the local domain. Using the authentication service as an example, the Required mechanism may be 'Password', while the supported mechanisms could be 'SmartCard', 'Fingerprint', and 'Retinal Scan'. 'Password' is the minimum requirement, as it requires the user to possess certain knowledge, i.e. the password. However, the other mechanisms rely on users possessing another form of authentication, such as a card, or some biometric measurement. To enhance performance and reduce negotiation, at least some of the mechanisms used should be based on a standard common set of mechanisms.

- **Policy** defines the identifier of the policy configuration used. Use of standardised policies configurations (see sections 5.3.1) facilitates the negotiation of a common policy. As in the Mechanisms section, a Required/Supported structure will specify the required policy configuration, and alternate policy configurations that can be supported.
- **Mapping** includes an identifier to the mapping that is held by the Domain Mapping Agent object. This is used to locate the mapping that is applied to translate the policy configuration specified in the policy section to the policy configuration required for interoperation with a foreign domain. The section also contains a date field to identify when the mapping was set. The use of dates in the

Chapter 5: Secure Interoperability in a DPE

Policy and Mapping sections allow the DPE to check that the mapping remains in-synch with the policy configuration. If the policy configuration is updated, then the mapping should possess a date equal to or greater than the date specified for the policy; otherwise the Mapping may be from an old configuration and may no longer be sufficient to translate the new configuration.

The whole SecureInteroperabilityPolicy structure is summarised in table 5-5 below. The identifier section allows the administrator to set different policies for different object types with different domains and, thereby, optimise performance and tailor security requirements.

Service	Basic Structure	Function
General	Identifier	Uniquely identifies the policy - and the object and domain it applies to.
	Object type	Identifies the object type (i.e. class) the policy applies to. If set to default, then the policy applies to all Object Types (without a specified policy).
	Domain id	Identifies the foreign domain the policy applies to. If set to <i>default</i> , then the policy applies to invocations on all foreign domains (without a specified policy).
Authentication	Mechanisms	List the authentication mechanisms required and supported by the domain
	Policy	List the authentication policy configurations required and supported
Access Control	Mechanisms	List the access control mechanisms required and supported by the domain
	Policy – Attribute	Identifies the mappings that can be applied to the Attributes
QoP	Mechanisms	List the QoP mechanisms required and supported
	Policy – Type, Msg_part	Lists the QoP policy configurations for Type and Msg_part
Non-Repudiation	Mechanisms	Lists the non-repudiation mechanisms required and supported
	Policy - Evidence	Lists the non-repudiation policy configurations required and supported
Trust	Authority	Id of a TTP that can be used to validate domain administration (a public key certificate)
	Expiry time	Expiry time of SecureInteroperabilityPolicy

Table 5-5 SecureInteroperabilityPolicy Structure

5.4.3.2 Domain Mapping Agent

The Domain Mapping Agent can be considered a registry/repository for mappings between policy configurations. The policy can be accessed through a unique identifier, which identifies the domains involved in the mappings, and uniquely identifies the instance of the Domain Mapping Agent, e.g. A_B_1.0 identifies the version 1 mapping between domains A and B. The Domain Mapping Agent maps values between two sets of attributes, as agreed by the domain administrators.

5.4.3.3 Secure Interoperability Agent

The Secure Interoperability Agent is responsible for taking control of an inter-object communication, once it has been identified as an inter-domain interaction. Before contacting another service object, the client will need a reference to it. Within IIOP, this involves an object identifier that immediately identifies that object as originating in a foreign domain. Any DPE interoperability protocol will include such a facility, because even though location transparency is provided to the client, the DPE needs to possess this type of knowledge to locate the object. Once the QoP Agent has identified the target object as being in a remote domain, the SIA will be called. It will then retrieve the secure Interoperability policy and possibly any necessary domain mapping information, in order to begin negotiations for a secure context with the QoP Agent and Secure Interoperability Agent of the remote domain.

5.4.4 Secure Interoperability Example

The following example illustrates how the secure interoperability components would work together to help create a secure association between different domains. Table 5-6 below lists the relevant SecureInteroperabilityPolicy values for the client and server. As non-repudiation is not included in the example, it is not included in the table. The policy was specifically created to allow interaction between the specified domains. Both use ACLs for access control, but have agreed a domain mapping for their attributes and roles (see table 5-7). The use of RoleB2 and RoleB2_1 is deliberate, to illustrate the fact that Domain B only had two roles (RoleB1, RoleB2) while Domain A had three roles defined. Therefore, when agreeing the mapping, Domain B's

administrator created another role (sub-group), RoleB2_1. Both client and server require integrity checks.

Service & Level	Config.	User Policy (A)	Provider (B)
Identifier		Default_1.0	A_B_1.0
Object Type		Default	Default
DomainIDs		Default	A
Authentication Mechanism		Required: - Supported: -	Required: -
Authentication Policy	Type	Required: - Supported:	Required: Client Supported: -
Access Control Mechanism		Required: ACL	Required: ACL
Access Control	Mapping	Mapping: A \leftrightarrow B	Mapping: A \leftrightarrow B
QoP Mechanisms	Integrity/ Confidentiality	Required: - Supported:DES, RSA	Required: RSA, DES Supported: -
QoP Policy	Type Msg_part		Confidentiality
TTP		<TTP_A cert>	<TTP_B cert>

Table 5-6 User/Provider SecureInteroperabilityPolicies

A	\leftrightarrow	B
Attributes in A	\leftrightarrow	Attributes in B
read and/or execute	\leftrightarrow	get
write and/or execute	\leftrightarrow	set
read, write, execute	\leftrightarrow	manage
Roles in A	\leftrightarrow	Roles in B
RoleA1 (write, execute)	\leftrightarrow	RoleB1 (set)
RoleA2 (read)	\leftrightarrow	RoleB2 (get)
RoleA3 (read, write,execute)	\leftrightarrow	RoleB2_1 (manage)

Table 5-7 Attribute and Role Mappings

In the example, a user A (client), whose role is defined as 'RoleA2' and has access right 'read' in Domain A is requesting an existing service (server) in Domain B (see figure 5-2 below). The example is a sub-set of the 'logging into a provider' scenario (see section 4.5.1). Therefore the user in Domain A wishes to log into the provider in

Domain B, and both are in different administrative security domains. Once the Security Agent realises that the provider is in a remote domain, it will use the Secure Interoperability Agent and QoP Agent to create a secure context. The example is detailed below.

Secure Interoperability subset of Logging in to a Provider in a Foreign Security Domain:

This example shows the user A establishing an access session with their named user agent of the provider. The user wishes to make use of the provider's services, which the user has previously subscribed to.

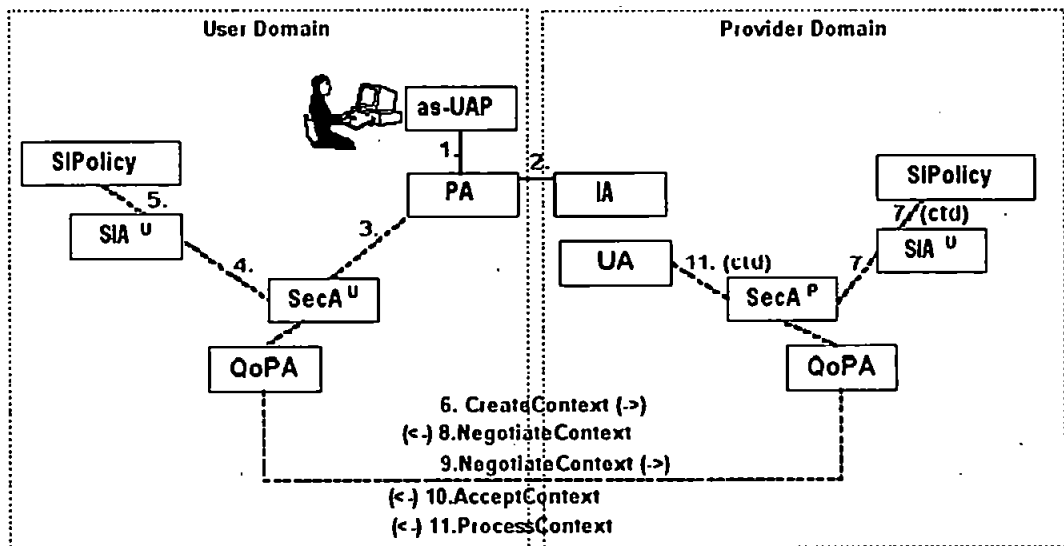


Figure 5-2 New Secure Interoperability Login Example

Preconditions:

The user has contacted the provider, and the Provider Agent (PA) has an interface reference to an Initial Agent (IA) of the provider.

The new security preconditions required:

Chapter 5: Secure Interoperability in a DPE

- security is available in both the user and provider domains;
- user A is defined as an authorised user in the provider domain;
- a QoP Policy is available for the Provider Agent and User Agent;
- there are no audit or non-repudiation policies related to the login process;
- to facilitate secure interoperability, the administrators have defined a mapping between the security attributes and roles of the domains.

Scenario:

(The new secure interoperability interactions are steps 4 to 11.)

1. User A uses an access session related User Application to login to the provider, as a known user. The access session related User Application requests the user authentication information such as the UserID and password. The user then requests the Provider Agent to login to the provider, as a known user. The access session related User Application supplies the security information to the Provider Agent.
2. Provider Agent requests that an access session is set up with the named User Agent of the user. Provider Agent provides the username of the user to the Initial Agent.
3. The Provider Agent also sends the security information to the Security Agent in the user domain, Security Agent^U.
4. Security Agent^U now has the Initial Agent reference and is aware that it is in a foreign domain. Therefore, in order to send the information to the Security

Chapter 5: Secure Interoperability in a DPE

Agent in the provider domain (Security Agent^P) to authenticate the user, a secure association must be established. Security Agent^U now contacts the Secure Interoperability Agent in the user domain (Secure Interoperability Agent^U).

5. Secure Interoperability Agent^U retrieves user A's secure interoperability policy information for Secure Interoperability Policy^U.
6. In conjunction with the QoP Agent^U, Secure Interoperability Agent^U contacts the Secure Interoperability Agent^P in the provider domain to establish a secure association, using the CreateContext, which contains the client's security context information, which is comprised of the association options and a security token. The security token is a generic token that is hiding the security mechanism-dependent information (in this instance the security token consists of user A's request for the TTP_B's RSA public key certificate, along with TTP_A's public key certificate).
7. Secure Interoperability Agent^P receives the request, via Security Agent^P. It extracts the security context information (the certificate request), along with the other interoperability options defined from Secure Interoperability Policy^U. Secure Interoperability Agent^P then contacts the Secure Interoperability Policy^P to obtain the provider's policy. In comparing the options, the provider decides to use RSA and DES to provide QoP, and utilise the domain mapping. It also extracts TTP_A's certificate.

8. Secure Interoperability Agent^P then sends a NegotiateContext, in conjunction with QoP Agent, to Secure Interoperability Agent^U. It defines the accepted association options, and the security token contains the TTP_B's certificate, and is encrypted using TTP_A's public key.
9. Secure Interoperability Agent^U receives the message, decrypts it with the private key, extracts TTP_B's certificate, and is ready to accept the association options. Therefore it now responds with NegotiateContext, to Secure Interoperability Agent^P where the security token now contains a DES secret session key and user A's security id and privileges, which is encrypted using TTP_B's public key.
10. Secure Interoperability Agent^P receives the message and decrypts it using TTP_B's private key. It can extract the secret session key and also extract user A's security privileges. The user and provider have now established a trust relationship between the domains, using the TTPs. It then access the appropriate Domain Mapping Agent, and can translate user A's privileges using the mapping. Secure Interoperability Agent^P sends a AcceptContext, using the DES session key.
11. A secure communication channel has now been established between the domains, and the user's original request to the Initial Agent can be transmitted in a ProcessContext message, via the Secure Interoperability Agent.
12. Session will continue using the Secure Interoperability Agents and QoP Agents for secure interoperability.

Post-conditions:

The new security post-conditions required are:

- secure context exists between the domains;
- user A's privileges have been mapped to the provider domain, even though user A is not identified as an authenticable user by the provider.

Alternatives within scenario:

There are several alternatives available within this scenario:

- if both domains did not support compatible mechanisms for any of the services, the interaction would fail;
- if the user did not exist in the provider's domain, and there was no TTP to authenticate the domain, or domain mapping, available then the interoperability would fail;
- Secure Interoperability Agent/Secure Interoperability Policy can be used to ensure that domains are compatible, even if a domain mapping is not required (i.e. the user profile exists in the provider domain, and so the user can be authenticated and have id and privilege attributes assigned in the usual manner).

Although the introduction of the new Secure Interoperability Service constitutes additional overheads, it should be noted that the service allows two domains to

interoperate with the minimum administrative interaction, e.g. domain A and B in the example above can interoperate if they provide TTP certificates and a secure Interoperability mapping (assuming that the domains have common mechanisms and certificates are available to the appropriate clients/servers) – this is instead of domain A having to add all of domain B's users to its security policies and domain B having to complete similar actions for domain A's users.

5.5 Summary

Although secure interoperability is not a new concept, its application to DPEs is new in this research. In existing DPEs, secure interoperability is a real problem. It exists because disparate domains can have different mechanisms, policies, protocols, and trust models. All of these differences have to be overcome in order to provide interoperability. There are several mechanisms that can be used to help overcome this issue. Bridges provide a quick and easy solution, but they do have limitations. Immediate bridges are not flexible for large numbers of interoperating domains, whereas mediated bridges can increase the performance overhead because they increase the number of times a single message has to be encrypted and decrypted. Standardisation of mechanisms and policy configurations provides two benefits, it allows clients and servers to negotiate their secure context, and it also facilitates mechanisms and service (policy) independence. The use of generic tokens also facilitates these characteristics. All of these methodologies can be applied to any DPE to help solve the secure interoperability problem.

Chapter 5: Secure Interoperability in a DPE

Although the standardisation of mechanisms used, policy configuration and mappings, may seem unrealistic, it is not. In the selection of standard mechanisms for use, there already are obvious leaders in each of the security services. For example, access control is generally done by ACL or Capability lists; QoS would find 3DES, RSA, and MD5 as some of the most frequently used mechanisms. Policy configurations are limited to the key issues, which are completely mechanism independent, and the mappings apply to these policies. The fact of using such standards can be seen as a limitation to any system. However, as always there is a trade-off, limitation of mechanisms used versus the ability for interoperability to occur without any user or administrator intervention. In large-scale distributed systems, which may cross many boundaries, the administrative overhead would be prohibitive if interoperability required specification of mechanisms, policies and agreements for each domain-boundary crossing.

Up to this point in the research, consideration has been given to security in the DPE, with respect to the security services itself and how it can interoperate between disparate domains. However, the research needs to extend this scope and look at secure interaction with other DPE services. The following chapter will now look at how DPE supporting services can improve security by becoming 'security-aware'.

6. Security-Aware DPE Services

6.1 *Introduction*

The research has, until now, concentrated on end-to-end security between a client and server. However, this is making the assumption that only the core ORB and its security service are necessary to complete a secure client-server invocation. But a DPE is more complex than this. According to TINA-C (see Section 2.3.3), the main function of DPE [99] is to provide uniform execution environment and basic capabilities for interaction between objects in heterogeneous network, and this is supported by a range of DPE services to provide extra functionality to application objects.

According to the ISO architecture, security should be provided in a modular format [41]. This architecture divides system management into functional units, FCAPS – the ‘S’ being the security module. A system should be able to function independently of the security service, and when the security module is introduced the same system should now operate in a functionally similar but secured fashion. In other words, the service should be a self-contained module that can provide security without having to change any other services. This type of thinking is practical in a centralized system such as IBM’s Resource Access Control Facility (RACF) [80]. Here the TCB is contained within a single system. The security service can monitor all requests and provide the required security functionality. However, distributed systems are more complex. As previously discussed in section 4.2.1 distributed objects introduce complications and the TCB is no longer contained in a single system and may need to

operate across multiple systems, i.e. security domains (see Section 3.3.1 and 3.3.2). This results in an extended set of security requirements for a DPE (see Section 4.2). Therefore the modular solution may be inadequate.

While it is recognized that security should be pervasive [41], the issue in a DPE is what the term pervasive means. If pervasive security in a DPE should be part of the whole environment, which implies that the supporting services should also be secured, then the modular solution may not be sufficient.

The objective of this chapter is to look at these services with particular reference to a Trader Service (see section 2.5) and see if in the current modular security architecture is adequate to secure them. This topic has not been investigated in previous literature. It concludes with recommendations for supporting secure operation of the Trader.

6.2 Security Issues for Supporting Services in a DPE

As described in Chapter 2, the DPE is reliant on a set of services to provide support for distributed objects, i.e. to handle distributed processing and provide transparency between clients and servers. Some of the TINA DPE services previously identified (see section 2.3.3) are listed below:

- **Trading:** provides a binding between objects that use a service (importer) and objects that provide the service (exporter);
- **Notification:** enables objects to receive notifications without being aware of the set of recipient objects;

- **Transaction:** consists of three main management functions: transaction, concurrency control and deadlock management;
- **Security:** authentication, authorisation and security controlling.

Each service is implemented by a number of objects. Currently security is implemented by applying the security rules to these service objects. This means that access can be granted to a client, when requesting use of a service object, if the client possesses the appropriate privilege attributes. However, even looking at an overview of the services some security issues become apparent. They are outlined below:

- **Persistence Service:** The Persistence Service stores components persistently on a variety of storage servers. Although access to the persistent storage objects are controlled, the stored data is not secured – the security service has no control over this; it would be an implementation level detail, i.e. if the data was stored in a database, the implementer would enable database security.
- **Naming Service:** The Naming Service locates components by name. Once an object can access the naming service, it can access all names in the service, as there are no security restrictions. Also Naming services can be federated, i.e. two naming services are linked together to operate like a single service. If the federation exists across different security domains the client is unaware that he is crossing a domain boundary and security controls could be by-passed
- **Event Service:** This service allows ‘consumers’ to register/unregister interest in specific events. The ‘suppliers’ then generate information about this event and send it to the consumers via an event channel. It is a basic

publish/subscribe or notification service. Security has not been defined for the event channels, i.e. access control is not available for specific events on a single channel, and there is no indication whether the channel requires encryption. Also the event service demands a certain amount of Quality of Service (QoS), i.e. guaranteed delivery, persistence of event data in the event of an event channel failure and use of logging facility. If the event channel was subject to encryption then the supporting QoS mechanisms, would also need to ensure security, e.g. the persisted data would have to be protected.

- **Query Service:** This allows a client to use query operations for attributes associated with objects, in much the same way SQL can be used to query a database of records by querying the fields in the records. It provides for asynchronous query, so that the query can be issued and the client does not have to block while waiting for a response. No security precautions have been added and so there is no way to identify what attributes a client can perform queries on, e.g. does the client have the security clearance to query a payroll attribute on an employee database. Another problem is Denial of Service, e.g. a rogue client can flood the query service with too many asynchronous or long running synchronous queries thereby causing the services to halt or crash.
- **Trader Service:** Similar in function to the Naming Service, the Trader allows an importer to locate an object, published by an exporter, but it does so by identifying a set of required properties. A security problem could arise if some of the services offered by the trader require higher security clearance than others; there is no way of controlling access to particular offers in a single Trader.

There are security issues that exist in DPE services that are not currently addressed. The above descriptions are just high-level overviews of such problems, but the problem demands further detailed investigation. Therefore a single service, the Trader, was selected and examined in detail (see section 2.5 for a detailed description of the Trader).

6.3 Security issues related to Trading & Traders

Traders in a distributed environment are open to attack, as is any part of a system. The research has defined the areas where Traders are most vulnerable to security breaches, and categorised them below within the five ISO security concepts.

6.3.1 Authentication

Traders receive requests for imports/exports from members of the trading community. Like any system resource, they are susceptible to masquerade (see Section 3.2.2). Authentication is the service required to counteract this threat. It is a two-way process; Traders, as well as importers and exporters should be identifiable and authenticatable.

6.3.2 Access Control

Access Control needs to be handled at two different levels. Firstly, access control of the Trader itself should be considered, i.e. who has access to the Trader. Secondly, access control of service offers must be handled, i.e. which service offers an importer

can see within a Trader. The access control rules need to be preserved across linked Traders.

6.3.2.1 Unauthorised Trader Access

Traders should have access control information, just like other objects in a distributed system. It should be listed in the access control mechanism, e.g. an ACL (see section 3.2.1). If trading community objects, e.g. Trader and exporter, are listed in the ACL, then the access control manager, i.e. Authorisation Agent (see section 4.4.1.2), would be able to make decisions relating to access, e.g. who can make requests on a specified Trader. For example, a Trader operating in a domain where access is controlled on the basis of roles, may use the roles of 'Role2' and 'Role1', where 'Role1' has a higher security classification than 'Role2', i.e., 'Role2' < 'Role1'. In figure 6-1 below, the Trader1 can only be accessed by 'Role1', where as the Trader2 can be accessed by both 'Role2' and 'Role1'.

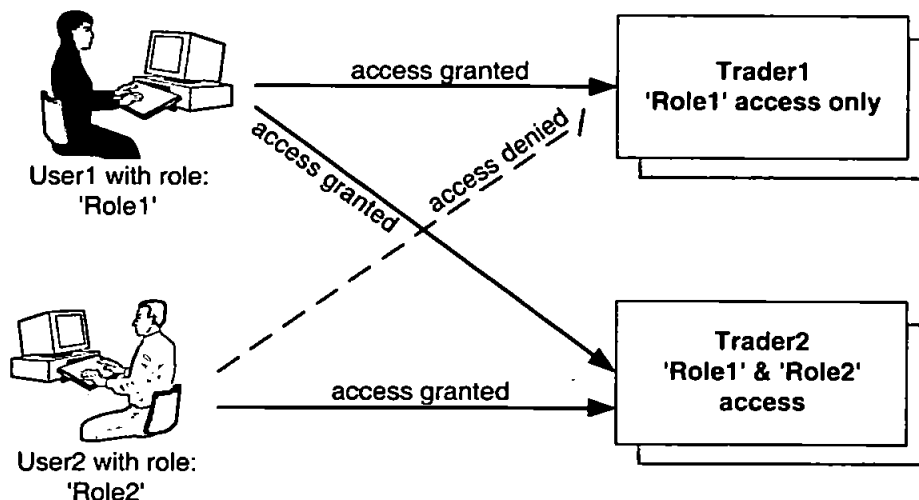


Figure 6-1 Trader Access Control

6.3.2.2 Unauthorised Service Offer Access

Even if an importer has access to a Trader it may not have access to all the service offers that the Trader holds. Some of the service offers may be of a higher security classification, for example, the security classification of the exporter could, by default, be assigned to the service offer. Alternatively the exporter could specify a security classification equal to or lower than its own classification.

Taking the scenario in the previous section 6.3.2.1, where Trader2 allows both 'Role1' and 'Role2' to access the Trader, if service offer access is enforced then some of the service offers will only allow 'Role1' to view them and some service offers will allow both 'Role1' and 'Role2' to view them, as illustrated in figure 6-2 below.

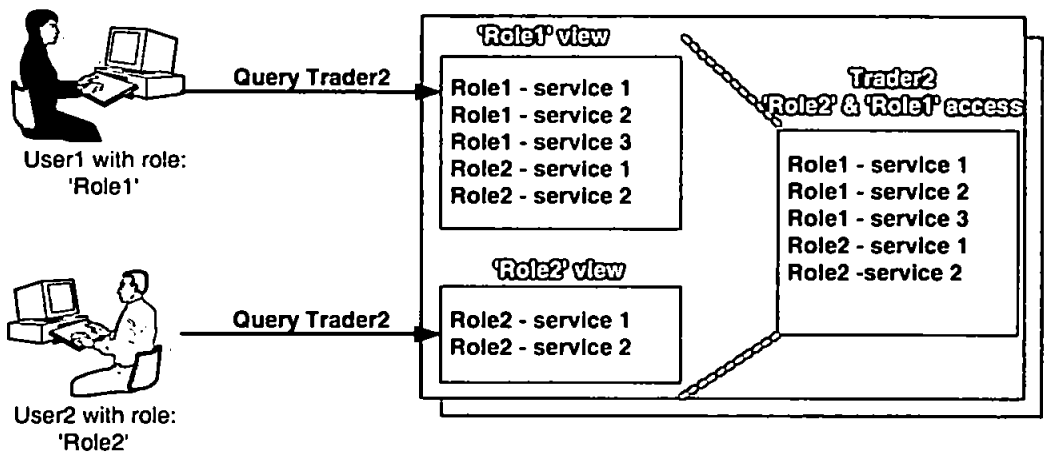


Figure 6-2 Trader Service Offer Access Control

6.3.3 Integrity and Confidentiality

Integrity and confidentiality of data, stored [100] or in transit [101], must be guaranteed in a distributed system; this has to include trading-related data.

6.3.3.1 Stored Data

Details of service offers, including an object reference, are stored in the Registry. It must be protected, as an intruder may try to gain access to a service by gaining illegal access to the object. Similarly details of the Service Type held in the Repository, should be protected to ensure that intruders do not have knowledge of ‘how’ to use the service type, i.e. interface details, parameters, etc.

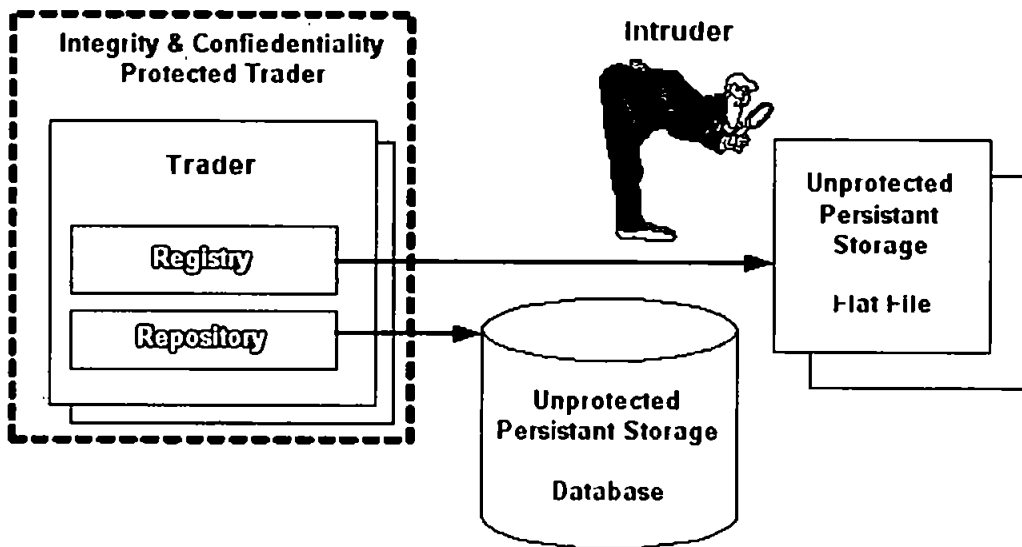


Figure 6-3 Protecting Stored Data

It cannot be assumed that the Trader’s backend data, i.e. the data stored in the Registry and Repository, is hidden behind object interfaces and, therefore, is not as

vulnerable to attack as object references that are exported through the interface. Intruders do not always use legitimate access mechanisms and, therefore, the ‘backdoor’ entry must be considered – see figure 6-3 above. Such data will usually be held in persistent storage, such as a database, or flat file. Therefore the Trader, if operating as a security-aware service, should be able to guarantee that the data is secure, even when it is in storage. Cryptographic mechanisms (see section 3.2.3 and 3.2.4) are used to ensure that the confidentiality and integrity of the data is preserved.

6.3.3.2 Inter-Community Communications

Since a Trader is operating in a distributed environment, this provides an intruder with ample access to intercept any communications between members of a trading community. Object references and service type details are transmitted to exporters, importers and other Traders. From such interceptions, one may be able to re-construct Registry/Repository information. Therefore transmitted data has to be protected. All communications between trading community members should be secured to ensure the confidentiality and integrity of all messages.

6.3.3.3 Secure Interoperability

The issue of secure interoperability was covered extensively in the previous chapter, and is particularly pertinent to the issue of federated trading, when the Traders exist in disparate security domains.

6.3.4 Non-Repudiation

The trading community is made up of distributed objects, which are less predictable due to their flexible and granular nature [24]. There are two problems. Firstly, if the

intruder is an authorised user, or is successfully masquerading as an authorised user, how can their actions be discovered? For example, an intruder can masquerade as an importer, and query Traders to find useful service offers. The process of monitoring a database may help, by providing clues to an intruder's activities. Secondly, if interactions are taking place, how can it be proven that a specific interaction or event took place, if one party wishes to deny the event, i.e. accountability? Irrefutable evidence is required from a non-repudiation service.

1. **Monitoring:** All security related events should be monitored. These events are defined by the security policy. Apart from notifying an administrator, via an alarm, that an illegal action has been taken, monitoring could also provide clues to a previously unknown intruder, e.g. an importer making multiple unauthorised import requests on several Traders. However, this requires data filtering to find trends, which can be used to raise a system administrator's suspicions.
2. **Irrefutable Evidence:** Non-repudiation is used to provide irrefutable evidence that certain events took place. For example, digital signatures can be used with audit logs to record events. Just as other system resources are subject to a non-repudiation policy, so too are all the trading community members.

6.4 Current Limitations

Within the current DPE specification of TINA, security of a DPE service is not defined. Although the access session does provide a limited notion of authentication and authorisation (see section 4.3), there is no specification of how this is applied to a

service. As the location of Trader objects, within the service environment has not been specified, it is initially assumed that they are available only within service sessions. The current model suggests that there is no security available and so the trading actions are not secure. If, however, the assumption is made that a Trader can be available in both service and access sessions, then access-session objects can be secured through authentication and authorisation, but the service session Trader is still insecure. Additionally, in both of these scenarios, there is no Quality of Protection (QoP), audit or non-repudiation security available. Similarly the lack of a secure interoperability protocol provides a problem, especially in the case of federated trading across security domain boundaries.

The current DPE specification is insecure for DPE services. If, however, the new Security Framework is applied it still does not address all the issues specified in the previous section. Although access control of the Trader can be handled by the security framework, via the Authorisation Agent (see section 4.4), the access control of the service offers within the Registry cannot. The new security service has no way of associating security data with a particular service type instance stored in the Registry; it only associates security policies with objects or methods on an object. It would require the storage of a security property in the Registry itself. The reason for this is that such a property would be used to sort and make selections when providing service offer lists to importers. This problem is also linked to delegation as the security property would be set in the Registry and would probably be delegated from the exporter, e.g. use the exporter's security level.

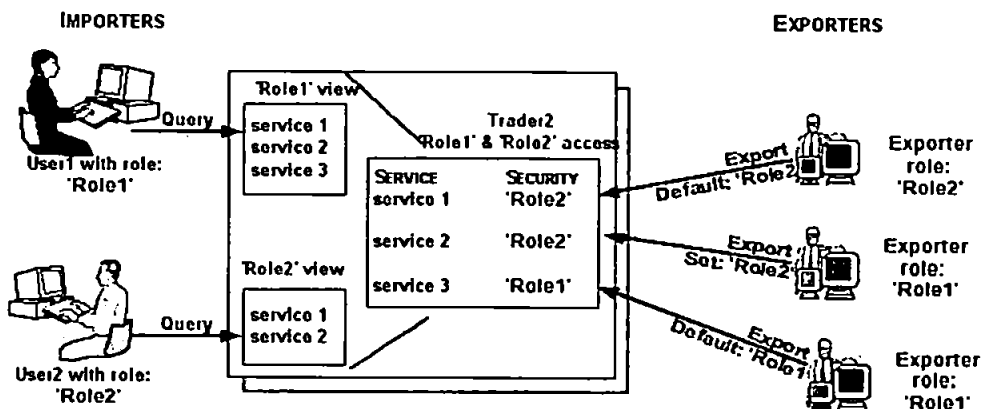


Figure 6-4 Service Offer Access Control with Registry Security Property

In figure 6-4 above, three exporters are exporting services to a Trader. The first exporter has the 'Role2'. When it exports a service offer, the Trader takes 'Role2' as the required security role for access to the services, i.e. as the security property values in the Registry. The second exporter has 'Role1'. When exporting its service offer, it specifies 'Role2' as the required security role. This is possible because 'Role2' has a lower security classification, i.e. 'Role2' < 'Role1'. Finally, the third exporter has a 'Role1'. It exports its service offer to the Trader and accepts the default security property of 'Role1'. In the example, when an importer invokes the lookup operation on the Trader, only the appropriate services offers are returned, i.e. the importer can only view service offers with security properties (Role) less than or equal to their own security property (Role).

Securing trader data, such as that held in the Registry and Repository, needs to be addressed. Currently these databases are not encrypted. In addition, trading community communications should be secured. The level of security would depend on the objects involved and their security level, as well as the level of the service offers being exported/imported.

Securing transmitted data requires the use of cryptographic mechanisms to preserve the integrity and confidentiality of the messages. The use of secure contexts, as specified in section 4.4 via the QoP Agent, would provide protection.

As for stored data, there are a several possible solutions. The data could be encrypted before it is written to storage and then decrypted after it is read. This is a solution most suited to flat file systems. It could also be applied to database systems. However, most databases today employ a security service of their own, i.e. they will secure the data [102]. These systems are designed to maximise efficiency while still ensuring the security of the data and, as such, it would be preferable to utilise these facilities.

There is one further option that would offer a generic DPE solution as opposed to the product-dependent solutions above. This option involves the use of a DPE Persistence Service. This service would have to be aware that security was in operation and that the stored data needed protection, i.e. it needs to be security-aware. The data for the Registry is stored in some persistent storage facility such as a database or file. The data is stored using the Persistence Service [103]. If the Persistence Service is security-aware it will ensure that when the data is held in the data stores (e.g. a database or file) it will be protected. However, DPEs are unable to deal with securing stored data because they do not provide security-aware services, and there are no other facilities to handle the encryption of stored data or utilise product-encryption facilities.

6.5 New Facilities Required

The previous section illustrates that the new Security Framework and Trader specifications are inadequate to provide security. Both the Trader modifications, described in this chapter, and the Security Framework (including secure interoperability), described in chapters 4 and 5, are required for a Security-Aware Trader. The new Trader facilities will now be discussed.

6.5.1 Security-Aware Trader Attributes

Attributes are already used in the Trader specification to provide a framework for describing the behaviour of any Trader (see Section 6.3.2). Security Attributes are now introduced into the Trader. They will control the security behaviour of a Trader, by specifying which security services it uses, i.e. just how security-aware the Trader is. Security Attributes are defined in Table 6 -1 below.

Security Attributes	Function Indicated
<i>Security-aware</i>	Indicates that some attributes are checked as the Trader is using security (at some level)
<i>Access_control_trader</i>	Include Trader in ACL and uses authentication with trading community members, etc.
<i>Access_control_service_offers</i>	Provide access control on the service offers listed in a query
<i>Encrypt_stores</i>	Encrypts <i>Registry</i> and <i>Repository</i> according to policy
<i>Encrypt_comms</i>	Encrypts communications according to policy
<i>Integrity_check_stores</i>	Integrity checks <i>Registry</i> and <i>Repository</i> according to policy
<i>Integrity_check_comms</i>	Integrity checks communications according to policy
<i>NR_trade</i>	Non-repudiation of Trading related events
<i>Audit_trade</i>	Audit Trading related events

Table 6-1 Trader Security Policies

It is now possible to have several types of secured Trader. For example, a Trader could be a 'Public Trader'. This means that everyone would have access to it and it would have no security applied, i.e. the *Security-aware* attribute would be set to off, indicating that all other security attributes were also turned off. Alternatively a Trader may be a 'Secured Trader'. It would be security-aware and have *all* other attributes turned on, i.e. it would use all the available security services. Another option is to make a Trader a 'Security-Aware Trader'. In this case the security-aware attribute would be on, and *some* of the other attributes would be on, e.g., *Encrypt_stores* and *Integrity_check_stores*, but not *NR_trader* or *Audit_trader*, thereby providing a specified level of security according to the policy within the domain.

6.5.2 Security-Aware Trader Data Structures

The two Trader data structures are the Repository and the Registry. The Repository should not have to be modified significantly, as it will hold the security properties in the same manner as it currently holds any other properties. The only change that is required is operational, i.e. if the Trader is security-aware or secure, then there must be a security property available in the data structures. The security property will be 'mandatory' and 'readonly', to ensure that it is available and cannot be modified. Table 6-2 below shows an example entry in the Repository. The security property is highlighted in ***bold italic***.

Service	Property Name	Property TypeCode	Property Mode
DataStore	Supports SQL	Boolean	
	Available Space (M)	Long	Readonly
	Location	String	Mandatory
	<i>Security</i>	<i>String</i>	<i>Mandatory, Readonly</i>

Table 6-2 Security-Aware Trader's ServiceType Repository Example

Table 6-3 below shows an example of two entries in the Registry that are based on the Repository service type example in Table 6-2 above. The example assumes that Roles are used as the security property and that 'Role2' < 'Role1'. Each entry holds the service type that is being specified; in this case it is a DataStore service. It specifies the service instance name and the list of appropriate properties and their values. The 'Supports SQL' property has no mode specified, and therefore is an optional parameter; as a result there is no entry for it in the 'DB Store'. Since the Security property is 'mandatory' and 'readonly', it always has a value, which cannot be subsequently modified. For the 'DB Store', the service exporter was a Role1, and so his 'Role1' role was delegated to the service offer. In the case of the 'File Server Store', the service exporter was a 'Role1', however the exporter specified the Security property as 'Role2' so that all staff members could access the data store.

ServiceType	Service	Property Name	Property Value
DataStore	'File Server Store'	Supports SQL	No
		Space Available	600
		Location	'Server room 2'
		<i>Security</i>	<i>'Role2'</i>
DataStore	'DB Store'	Supports SQL	Yes
		Space Available	800
		Location	'Server room 1'
		<i>Security</i>	<i>'Role1'</i>

Table 6-3 Security-Aware Trader's Registry Entry Example

6.5.3 Security-Aware Trader Interfaces

There are eight interfaces defined. However, only five of these interfaces should have to be modified, namely the Admin, Lookup, Register, Proxy and Link interfaces.

6.5.3.1 Admin Interface

The *Attributes* and *Set* methods will now have to deal with the additional security attributes specified in table 6-1 above. The *Attribute* methods allow the administrator to query the security attributes to find their current values. *Set* allows the administrator to modify the security attribute values, thereby allowing the administrator to specify the 'security-awareness' of a Trader.

If *Security-aware* is set to 'on', then at least one other security attribute must be set to 'on'; otherwise an error will be returned on the *Set* method. If *Security-aware* is set to 'off', then all other security attributes must also be set to 'off'; otherwise an error will be returned on the method. These attributes control interaction with the Security Framework. When a security attribute is set to on, it implies that a security service is

available and that a security policy for the Trader must exist. The following example in figure 6-5 will illustrate this. A Public Trader has been created by User1, i.e. it is security-unaware and all the security attributes are set to 'off'. User1 then calls the *Set* method to make the Trader security-aware and sets the *Access_control_trader* attribute to 'on', i.e. access to the Trader is subject to the Security Framework's Access Control Agent. User1 has a security role of 'Role1'. The Default AccessPolicy in the system assigns the security role 'Role2' to the object because 'Role2' is the lowest security role available in this system. Therefore when the Trader becomes security aware and requires an AccessPolicy, the Security Service checks the system to see firstly if an AccessPolicy already exists for the Trader; if it existed it would be used by the service to control access to the Trader. However, in this case, no such policy exists. Therefore the security service finds the Default AccessPolicy and also User1's AccessPolicy. It finds that User1's policy is of a higher security classification and, therefore, creates a new AccessPolicy for the Trader and assigns the higher classification 'Role1' to it. This will be achieved through the AccessPolicy component, via the Policy/Context Manager.

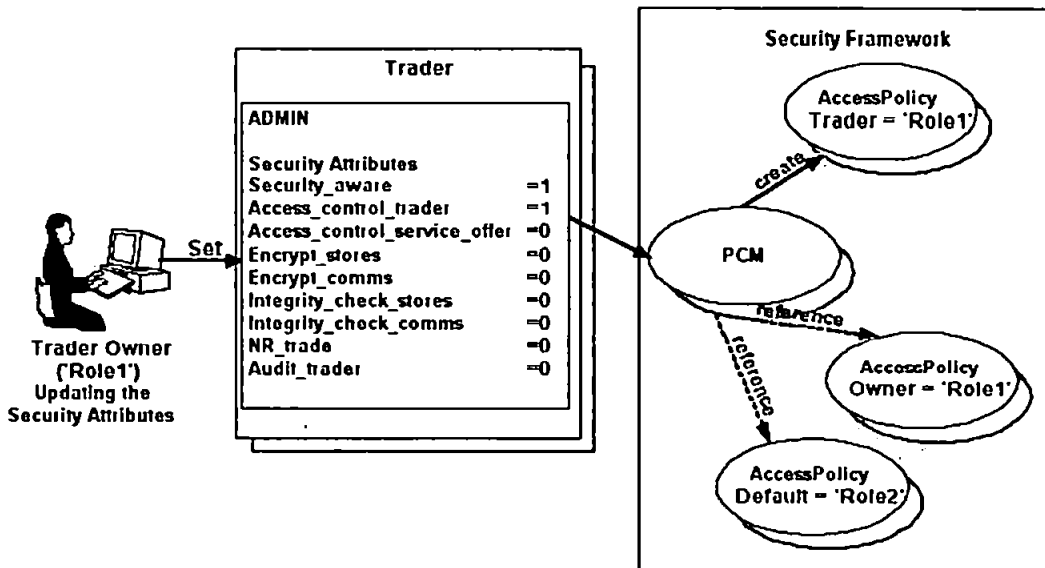


Figure 6-5 Security-Aware Trader's Admin Interface

The same procedure would apply to all security attributes:

1. Set the attribute to on;
2. Check if the appropriate policy object exists for the attribute;
3. If it exists, the policy will be used; if it does not exist then find the Default and Owner policies;
4. Create a new policy for the Trader based on the most secure option available between the Default and Owner policies.

6.5.3.2 Lookup, Register and Proxy

The Lookup, Register and Proxy interfaces now inherit the security attributes, i.e. an object with a reference to one of these interfaces will be able to query the security attributes to see how 'security-aware' a Trader is. This will allow trading community

members to make decisions relating to how they will behave in response to a Security-Aware Trader. The following example, depicted in figure 6-6 below, will illustrate this.

In this scenario, User1 again has a security classification of 'Role1' and is acting as an Importer. She wants to query a Trader to look for a DataStore service, but also wants to ensure that the Trader is security-aware and controls access to its data. User1 has the object reference for the Trader's Lookup interface, and so reads the *Security-Aware Attribute* for the Trader to see if it is secured. She can also read the other security attributes to check what security facilities are used – in the example both access control attributes are set. Now that User1 knows she is dealing with a secure Trader, she invokes the *Lookup::Query()* method to find service offers for DataStores. On the Trader side of the invocation, the attributes indicate that the Trader firstly needs to check if User1 is authorised to Query the service offers. The Security Service uses Access Control Agent (ACA) to find whether a client needs to have security classification of 'Role2' or 'Role1' to access the Trader. User1's credentials, 'Role1', can be delegated through the DPE. The Access Control Agent then decides that she can access the Trader interfaces. Secondly, the attributes show that the service offers themselves are access controlled. Since both service offers are less than or equal to the 'Role1' classification, the Trader returns both DataStore service offers to User1.

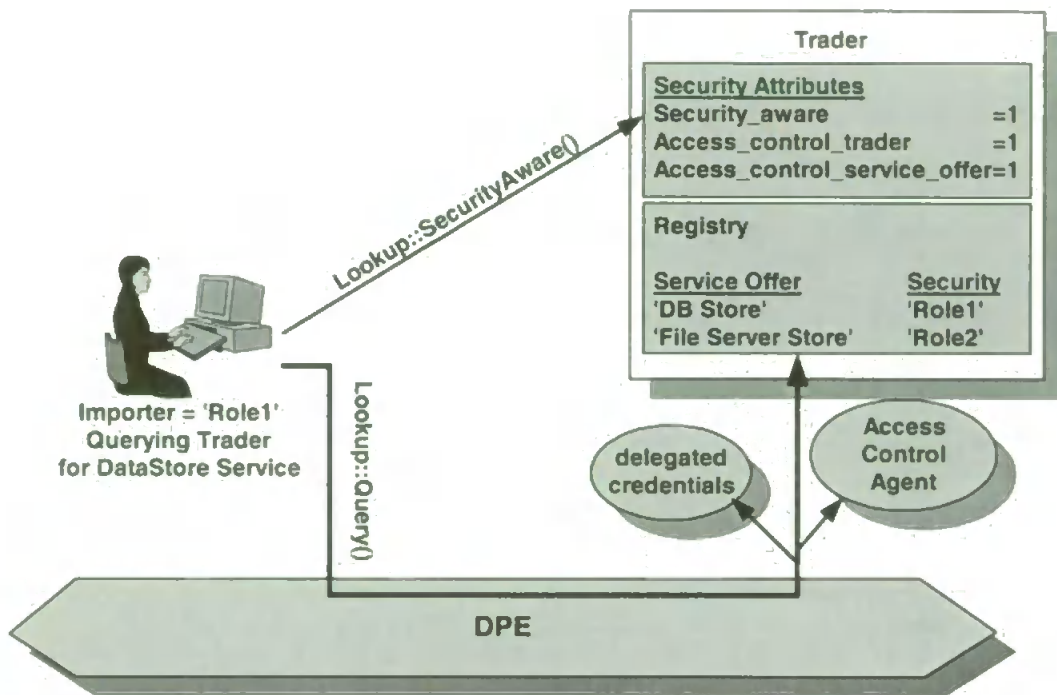


Figure 6-6 Security-Aware Trader's Lookup Interface

The Register and Proxy interfaces operate in a similar fashion, but are used by Exporters. Again an exporter can check the type of Trader it wants to export its services to – a Public Trader or a Security-Aware Trader. Then, on the method invocation, the Trader is able to use the Security Service to provide the functionality set by the security attributes, i.e. access control, confidentiality, integrity, non-repudiation or audit. The one difference would be when the Exporter is exporting a service, the security property for that service will either be taken from the exporter's own security classification (the default action) or the Exporter can specify a security classification equal to or less than his own.

6.5.3.3 Link Interface and a New Link Policies

The Link interface also inherits the Security Attributes as the Lookup, Registry and Proxy interfaces did above. It will affect Trader behaviour when two Traders are creating a link. However, there is one other change – the introduction of a new policy. This link policy will define how Security-Aware Traders can be linked. The new policy is *Link_security* and it defines the lowest security classified Trader that can be linked with, e.g. if *Link_security* is set to 'Role1' in Trader T1, then Trader T2 must have a security classification of 'Role1' or higher if it wants to invoke *Link::Add_Link()* on T1. This preserves the security of the immediately linked traders. However, in order for this to operate effectively the security interoperability service is necessary. If the two linked traders are in disparate security domains, then the credentials may have to be mapped so that the *Link_security* policy can be preserved. For example, Trader T1 is in domain A, is classed as a 'Role1', and the *Link_security* is specified as 'Role1'. Trader T2 is in domain B, is classed as an 'administrator' and the *Link_security* is specified as 'administrator'. A mapping exists between A and B so that 'Role1' maps to 'administrator'. Without secure interoperability, T1 and T2 could not be linked; however, with the mapping available, they can be linked and allowed to communicate securely.

6.5.3.4 Other Interfaces

For all other interfaces and methods:

- Security attributes will be treated like the other attributes;
- Security properties in the Repository will be handled like any other 'mandatory, readonly' property;

- Security properties will be handled like all other properties in the Registry;
- Security properties will be able to be used in Constraints and Preferences;
- Security properties will raise property errors as all other properties do, e.g. `PropertyTypeMismatch` in *Export* method on the Register interface.

6.5.4 Security-Aware Trader and the new Framework for DPE Security

The security attributes now allow the Trader to make use of the Security Service. The following sub-section looks at the problem areas identified in section 6.2 and describes how the problems, that would have been experienced by the DPE, have been overcome using the new Security Framework.

The *Access_control_trader* and *Access_control_service_offers* attributes allow the Trader to make use of the access control facilities. *Access_control_trader* ensures that a Trader's access control information, e.g. a security level, is available in the system, i.e. it has an `AccessPolicy`. A principal will own the Trader object, and the principal's credentials will be delegated to the Trader. Alternatively, the principal may specify a security level lower than its own for the Trader, e.g. the Trader may be specified as a 'Public Trade' (see Section 6.6.1). The Access Control Agent now supervises all access requests to the Trader (in accordance with the `AccessPolicy`), and all requests made by the Trader, e.g. a 'Role2' importer will not be allowed access a 'Role1' Trader, as it is considered less secure. *Access_control_service_offers* enables a security property value in the *Registry* and places an exporter's access control information in the Registry as the security property whenever *Export* is invoked, e.g. if an exporter has security role 'Role1', then the service offer exported will automatically take a default value of 'Role1' as its security property value. The

exporter may also specify a security level lower than his own, e.g. he may specify a security role 'Role2' which is lower than his own, 'Role1' role. The Security-Aware Trader now makes selections based on the security property when creating service offer lists, e.g. if a 'Role2' importer is looking for a service, it will only be shown 'Role2' service offers – it will not see any offers with a security level higher than its own.

The *Encrypt_store*, *Encrypt_comms*, *Integrity_check_stores*, and *Integrity_check_comms* control integrity and confidentiality in a Trader. All four security attributes enable the encryption and integrity facilities that are specified in the QoPPolicy object. This facilitates the separation of both stored and transit data policies and therefore the level of protection can vary if required. For example, stored data is held for a longer period of time than transmitted data and, therefore, it is more vulnerable to attack and so it may require a higher level of security.

Transmitted data will utilise the secure service objects, QoP Agent, and SecureInvocationPolicy. For stored data, the most generic solution was described in section 6.5, and involves the use of a security-aware Persistence Service. In this case the Persistence Service has two options, it can apply mechanisms to the data before it is written to storage or it can utilise the security facilities of the storage product.

The *NR_trade* flag enables/disables non-repudiation for a Trader, i.e. non-repudiation is available but can be disabled if not required, e.g. a Public Trader may not require it or it may be a trade-off in an effort to improve performance. A Security-Aware Trader, with enabled *NR_trade* flag, will utilise the non-repudiation service objects, i.e. Non-Repudiation Agent, Non-Repudiation Store, Non-Repudiation Adjudicator and QoP Agent in accordance with the specified Non-Repudiation Policy.

The *Audit_trade* flag controls the Trader's access to the audit service. When set to on, *Audit_trade* allow the Audit Sampler Agent to decide whether events are to be audited, in accordance with the Audit Policy. If an event is audited the Audit Analyser and Audit Responder will decide what to do, by referencing the Audit Knowledge Base.

6.5.5 New Facility Summary

The following figure 6-7 (based on figure 2-9 of the Trader, see section 2.5.4), summarises the modifications that are required to create a Security-aware Trader:

1. New Trader security attributes;
2. Use of 'mandatory, readonly' security property in Repository;
3. New Registry security property;
4. Modified Admin interface, inherits Security Attributes;
5. Modified Lookup interface inherits Security Attributes;
6. Modified Registry interface inherits Security Attributes;
7. Modified Proxy interface inherits Security Attributes;
8. Modified Link interface, inherits Security Attributes, and new link policy *Link_security*;
9. Use of the new Security Framework, including secure interoperability;
10. Use of security-aware DPE services.

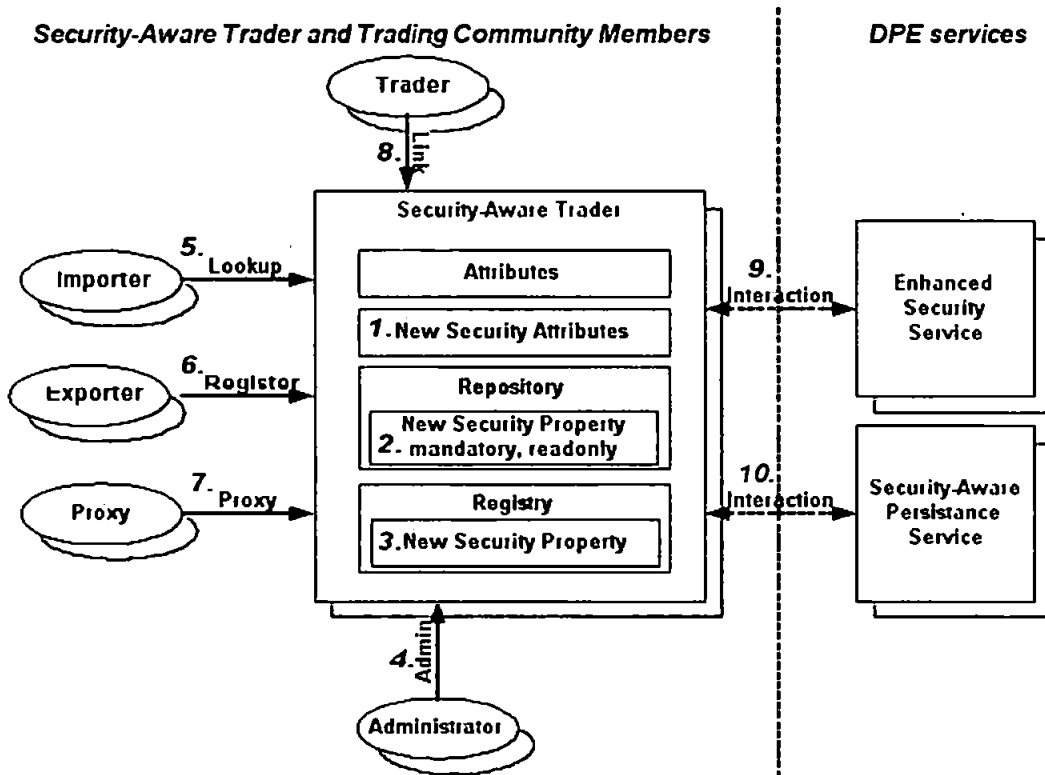


Figure 6-7 Security-Aware Trader

6.6 Other Security-Aware Services in a DPE

The previous sections have concentrated on the issues surrounding security and the trading services. The problems were addressed by the security behaviour of the trader using attributes and the new security framework.

Section 6.2 highlighted that security problems are apparent in other DPE services and not just the Trader. They can also be addressed using the same mechanisms as the Trader. The Naming service could also utilise attributes to decide whether a client has access rights to view a particular object name or reference. The Persistence Service is rather more complex. It could use attributes to decide whether data needs to be

encrypted, but then it would need to provide generic interfaces that allowed it to issue encryption commands. This would require integration with the security service's encryption mechanisms or with the encryption facilities of the data storage mechanisms, e.g. a database. These issues can be dealt with by providing separation between policy (service) and the mechanisms used to implement them.

In all of these cases it could be argued that the 'security-awareness' characteristic is not necessary. Instead the system administrators could, for instance, set up multiple Traders, each of which would have different access rights and therefore the service would not have to concern itself with the security attached to the individual offers available within the system. However, this increases the administrative overhead and therefore the likely-hood of human errors, which could result in a security vulnerability. In the case of large scale distributed systems it is not always possible to set up multiple Traders each with different access rights. This also relies on the fact that each exporter will know the Trader it is supposed to advertise its services in, i.e. know the Trader with appropriate security clearance and also assumes that enough resources will be available to allow multiple traders to exist concurrently. A security-aware Trader provides a simpler solution – it is less costly on resources and simpler to administer because the trader can handle security, and therefore provides a more secure solution. The same arguments apply to the other DPE services.

It can be surmised that at a DPE level, supporting services are required to be security-aware in order to fully secure the environment. This can be accomplished by using the following devices:

- Use of security attributes to indicate that security is required within a supporting service;

- Separation of mechanism and policy (service), so that when a security attributes indicates the security service is required, the ability to provide the service is not mechanism dependent;
- Secure interoperability to allow this functionality to operate across disparate domains.

6.7 Summary

Security is an issue for supporting DPE services. Although many of the services appear to have security issues, the only way to investigate fully was to select a specific service such as the Trader. Traders are an important DPE service because they allow clients to finding objects that are required, whether they are local or remote, which is pivotal to the success of a DPE. However, the Trader provides a very vulnerable point for attack, providing an intruder with access to a multitude of services. Therefore, it should be made security-aware. It should be able to ensure that only authorised clients can access it, and that clients can only view the service offers that they are authorised to see. To provide a Security-Aware Trader, new facilities are required in the Trader. This entails providing the Trader with security attributes that will govern its security behaviour. The Trader's Registry will also hold security properties that are associated with each service offer held. The security attributes will decide which security services the Trader will have access to, and the security properties will be used in access control. Therefore, the administrator can decide just how secure a Trader should be.

Security cannot be completely treated as an add-on facility. Within DPEs, each service has to be aware of security. This does not just apply to the Trader. It has already been suggested that other services such as the Persistence Service need to be security-aware if a distributed system is to provide a truly generic and secure environment.

The lessons learned from the Trader study can be applied to all DPE services. Security attributes, a complete security service that is mechanism-independent, and the use of secure interoperability, allow services to become security-aware and work together to provide a more secure environment. Having covered the theory of how a new Security Framework and security-aware service, such as the Trader, would operate to provide a more secure environment, the discussion now moves on to look at mapping this work to an implementable DPE specification.

7. Verification of the New Framework

7.1 *Introduction*

A new framework to provide security in DPEs has been defined in the previous chapters. It comprises three main components:

- security service objects – operational and management providing the main security service functionality;
- secure interoperability service objects to provide secure interaction between disparate security domains;
- security-aware DPE services, such as the Trader.

The entire framework has been defined in accordance with the TINA specification, which describes at a high-level, how DPEs operate. To verify the work, this chapter will map the framework to a current, OMG DPE specification CORBA.

7.2 *Mapping to CORBASec*

Before performing the mapping, it is necessary to first understand what aspects of the new security framework are missing from CORBASec. This is accomplished by evaluating CORBASec against the DPE security requirements previously specified in section 4.2.

7.2.1 CORBASec vs. DPE Requirements

Table 7-1 below summarises CORBASec against the list of security requirements defined by this research (see section 4.2). The “✓” indicates that the required functionality is present. The “-” indicates that while some of the functionality may be present, the full requirement is not met by CORBASec.

	Security Requirement	Functionality required	
1.	Identification and Authentication	Identify entities & generate identity attributes Use multiple authentication mechanisms	✓ -
2.	Authorization & Access control	Generate privilege attributes Use multiple authorization mechanisms Use role/groups	✓ - ✓
3.	Propagation of security attributes	Specify when propagation is required Specify constraints on propagation	✓ -
4.	Secure communications	Ability to select Quality of Protect Ability to select amount of message to be protected	✓ -
5.	Secure stored data	Ability to specify that data needs to be secured Ability to specify the Quality of Protection	- -
6.	Secure Auditing	Audit security relevant events Produce audit records Issue alarm Protect audit information in transit or in trail Should be extended to facilitate intrusion detection	✓ ✓ - - -
7.	Non-repudiation	Generation/Verification of evidence Storage of Evidence Secure transport of evidence Adjudicator facility	✓ - - -
8.	Administrative interfaces	System Management Service Management Mechanism Management	- ✓ -
9.	Interoperability	Interoperability at all levels- Invocation Security Service, Mechanism and Protocol Mapping of attributes between domains	- - -
10.	Scalability	Object system that can be distributed Use of domains Use of groups etc in administration	✓ ✓ ✓
11.	Integration with existing environments	Flexible structure to allow the model to integrate with other technology environments/security models Facilitates regulatory requirements	-

	Security Requirement	Functionality required	
I2.	System Recovery	-	-
T1	Intrusion Detection	Physical/logical procedures to prevent intrusion/modification	-
T2	Hardware/software protection	Mechanism management	-
M1	Inter-object communications	Authenticated & authorised object access Secure object communications	✓ ✓
M2	TINA services	Authorised service subscribers Secured service operation	✓ -
M3	TINA services	Secure control data Audit service available	- -
M4	TINA services	Secure administration data Secured access to administration data	- ✓
M5	Inter-DPE security	Authentication & access control	✓
A1	Secure participant interaction	Authentication & Access control Secured participant communications Audit Non-repudiation	✓ ✓ - -
A2	Application Admin	Usability Secured	- ✓
A3	DPE applications security	Security active Security in-active	- -
I1	Establish Trust	Authentication & TTP	✓
I2	Attribute Mappings	Domain mapper	-
I3	Operational interoperability	Mechanism-compatibility	-
I4	Control/Administration interoperability	Policy configuration compatibility	-
I5	Application Security Context	Secure interoperability protocol	✓

Table 7-1 DPE Security Requirements available in CORBASec

It is clear from the above table that the main areas of concern can be addressed by applying the proposed new framework, as it addresses the following issues, which are missing or inadequate in CORBASec:

- **Management:** requires consistent, comprehensive management framework that separates mechanism and service administration;

- **Securing Stored Data:** requires management of relevant policies and the ability to integrate with a security-aware DPE service;
- **Audit:** requires full auditing facility that can address IDS requirements;
- **Non-repudiation:** requires the full compliment of non-repudiation facilities (storage, delivery and adjudication);
- **Interoperability:** requires secure interoperability with entities in a disparate security domain;

Some of the DPE specific requirements are not fully addressed in this verification. Firstly, the segment of the Native Computing and Communications Environment (NCCE) security domain, i.e. mechanism management, is not addressed because the CORBA services are not mechanism-independent. Secondly, the differences in the DPE Services and Kernel security are not addressed. Security of DPE services is not considered. However, the issue of a distributed TCB (of which the kernel is the main component) is discussed. It is reliant on two elements – the use of interceptors and the trusted installation of security mechanisms. Interceptors are resident in the ORB and are able to catch all invocations at particular points in the invocation path, e.g. when leaving the client or when arriving at the server process. Security interceptors catch every invocation and call the appropriate security services to ensure that a request is in-line with the current security policy. Finally, application security is addressed. The notion of active and in-active security applications is addressed by CORBA's security-aware and security-unaware applications. Security unaware applications do not have any knowledge of security and rely on the security interceptors to provide

security. Security aware applications can use the defined security objects specified by CORBASec.

The shortcomings found with CORBASec can be further illustrated by looking at the products that are based on the specification [104, 105, 106]. All of the products have certain features in common because they all need to extend past the CORBASec specification because it is too restrictive:

- Extending the administration features through defining new interfaces;
- Using additional features to integrate with existing technologies, i.e. unitary logon, bridge technology;
- Extending the audit facilities to help secure audit records or make them available to monitoring tools.

However, there are still a number of restrictions:

- Replaceability is difficult and so they are all limited to specific sets of security technologies/mechanism;
- Data storage is proprietary, e.g. use of LDAP;
- There is no monitoring/IDS integration available;
- Non-repudiation is not available;
- Interoperability is still limited to compatible domains and technologies (although most have consulting divisions that provide customised solutions).

Another important point to note is that, while it has not been tested by any of the vendors, it would appear that none of these products will interoperate, out of the box, because they all support different technologies.

7.2.2 Mapping to the new Comprehensive CORBASec

Applying the new security framework enhances the CORBASec specification, and therefore it will be referred to as the Comprehensive CORBASec (CCS). The complete IDL for CCS is available in Appendix A. The mapping preserves the overall CORBA structure of an ORB using security interceptors. Therefore a direct mapping from the TINA structure is not appropriate or possible, i.e. a one-to-one mapping between TINA service objects and CORBASec objects is not possible. Defining new objects and modifying existing object within CORBASec provides the required functionality. Figure 7-1 below, summarises all of the objects involved in CCS. It highlights three object types:

- Objects that were defined in CORBASec and remain functionally unchanged from that specification;
- Objects that were defined in CORBASec, but are now *significantly* changed in order to facilitate *modified* or *new* objects;
- Objects that are completely new to the DPE and are used to facilitate the CCS's new functionality.

The figure has been divided into sections that represent the main service facilities available within the CCS. This can be compared with figure 2-11 in section 2.6, which shows the objects defined within the CORBASec.

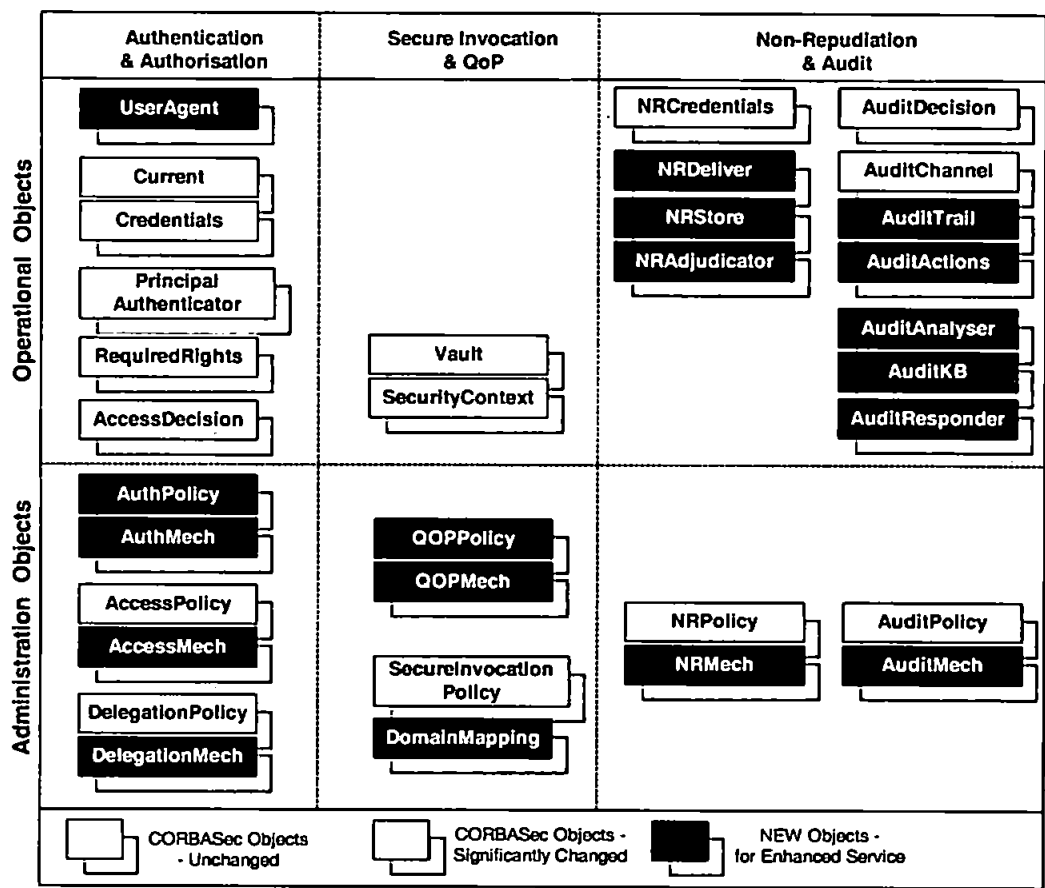


Figure 7-1 Comprehensive CORBA Sec objects

The following sections will examine the new and modified objects and how they provide the new functionality, to ensure the new DPE security requirements are met.

7.2.3 Management and Mechanism-Independence

The CCS separates the management of services and mechanisms and thereby provides the required mechanism-independence. In uses the four methods identified in section 4.4.3:

- 1. definition of new policy classes to separate management function;

2. use of opaque data types to assist abstraction;
3. definition of policies for all security functions for consistency;
4. ability to locate the new policies.

The CCS proposes the introduction of several administration objects, which are listed in table 7-2 below:

Security Service	Administration Objects
Authorization/Access control	AuthPolicy, AuthMech, AccessPolicy, AccessMech, DelegationPolicy
Integrity/Confidentiality	QOPPolicy, QOPMech
Non-Repudiation/Audit	NRPolicy, NRMech, AuditPolicy, AuditMech
Interoperability	SecureInvocationPolicy

Table 7-2 Administration Objects

To facilitate mechanism-independence, a new set of mechanism policy objects is introduced for each of the security services – AuthMech, AccessMech, QoPMech, AuditMech and NRMech. Each of these will describe the mechanisms used for the service. There is no mechanism policy for delegation, e.g. DelegationMech, as delegation is not handled by a separate mechanism; it will use those employed by the authentication and access control mechanisms, e.g. X.509 certificates, rights from an ACL.

Two new policy objects are introduced, AuthPolicy and QoPPolicy. AuthPolicy is responsible for the authentication security policy, i.e. the mechanism to be applied by an application, the valid authentication mechanisms available to a user and the relevant authentication data, such as ID and password. QoPPolicy holds the policy

information in relation to establishing a secure context between a client and server, i.e. the level of secure communication required.

The functionality of the remaining policy objects, which previously existed in CORBASec, is significantly changed in CCS. In CORBASec, non-repudiation policy was only supported at application level and only defined the rules for generation and verification, while the audit policy simply listed the types of application events audited and specified an associated AuditChannel, i.e. where the record was written. NRPolicy is now available at application and invocation level, and manages authorities, event types and mechanisms (via NRMech). AuditPolicy now manages the event selectors, the new audit objects responsible for monitoring, filtering and delivery, and the multiple AuditChannel options now available (see section 7.2.6.1 below). SecureInvocationPolicy is still used to manage secure invocations, however its functionality has been significantly extended. It now provides more configuration options, and is an inherent part of a new CCS interoperability service (see section 7.2.5.1 below), which manages negotiations between security domains at both service and mechanism levels. Therefore, there is no separate mechanism policy because this object is primarily used for negotiation, and it is more efficient to do so at one level rather than involving another object in the communication protocol.

One further issues is the ability to find the new policies as accomplished through the PCM in TINA. This is addressed by extending the DomainManager functionality (previously in CORBASec). Getting access to a policy via the domain manager needs to be updated to handle the new MechansimPolicy objects, as illustrated in figure 7-2 below. The DomainManager can now be queried to find the mechanism policies using

a newly defined method called *get_domain_mechanism*, e.g. *DomainManager::get_domain_mechanism(access)*.

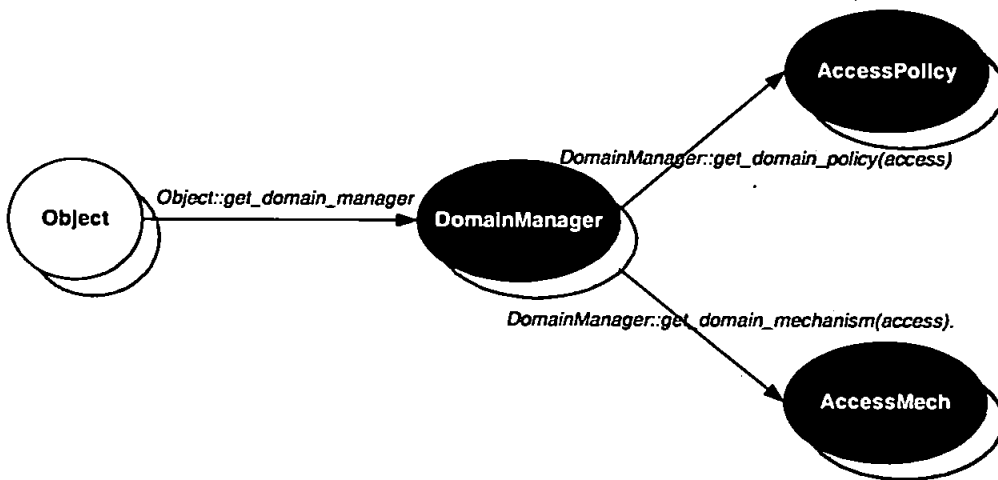


Figure 7-2 CCS DomainManager

Therefore the functionality of all the management service objects in the new security framework has been mapped to the new and modified management objects in CCS.

7.2.4 Authentication & Authorisation Enhancements

CORBA groups authorisation and authentication together, because they are so closely linked. Therefore they will both be studied under this section.

7.2.4.1 CCS Authentication & Authorisation Overview

The Authentication & Authorisation services provide three new facilities:

- mechanism-independent alternative to the User Sponsor Code;
- delegation controls;
- parameterised access control.

CORBASec refers to User Sponsor Code (USC), which is not part of the object system because it is mechanism specific (it represents a logon module). Therefore the CCS defines a UserAgent as a new object to provide a mechanism-independent means of communication with users. It is representative of some of the access session related User Application and User Agent functionality, which relates to security (see section 4.4). Although both of these objects already existed in the TINA model, they are responsible for interfacing with (access session related User Application) and representing (User Agent) the user in the system. Therefore it would be beneficial to include them in the CCS model as it would allow the system to interact with users irrespective of the logon mechanisms used, i.e. smartcard, biometrics, or password. In this way the UserAgent will provide the parameters to CORBA's PrincipalAuthenticator. The parameters are initially provided to the UserAgent using the operations; *set_security_name*, *set_auth_data*, *set_privileges*, *set_name*. This provides a mechanism independent way of getting authentication information because any product/mechanism can use these operations.

The UserAgent can then invoke the PrincipalAuthenticator with the user information. If the UserAgent is required to store any data it will have to consider the issues of securing authentication information. It could do this by defining the QoP required for the stored data (see section 7.2.5 below). A factory is a standard OO design pattern that allows the creation of a particular object type [107]. The logon module will have an associated UserAgent factory because it will need to generate a UserAgent for each user logging into the system. It is evident therefore that the logon module does not have a specific entity or principal, on whose behalf it is acting, as all other objects in the system would. It would be beneficial to allow the logon module to be mechanism

independent and therefore able to easily adapt to multiple authentication mechanisms, e.g. a single console that can deal with password, certificate or token authentication depending on the application being accessed. This is achieved by modifying how the authentication policy works. AuthPolicy usually requests all the information that a single principal requires to authenticate itself, e.g. a user's ID and password. The AuthPolicy needs to differentiate between a logon module and normal system object. This is accomplished by defining a PrincipalType – 'Principal' indicates a usual system object that has a principal and therefore authentication data is required, while 'UserAgent' (UserAgent system entry point) indicates that authentication data is not required because no single principal is involved, instead it associates the type of authentication with a particular service/logon module.

With regard to authentication, the Authentication Policy and Authentication Mechanism objects map directly onto the new and corresponding objects in CCS. The PrincipalAuthenticator is representative of the AuthenticationAgent. When the user has been authenticated it is the responsibility of the PrincipalAuthenticator to generate identity attributes for the user, the Credentials object. There is no service object that directly represents the Credentials, or indeed the Current objects. However, the data in these objects will be available through management contexts and the UserAgent.

CORBASec previously defined two access related policies – DomainAccessPolicy and AccessPolicy. The mapping now just provides for AccessPolicy because the distinction between the previous two objects was that DomainAccessPolicy managed the privilege attributes while AccessPolicy was used to query the access policy for a particular set of Credentials. To preserve a consistent management framework, DomainAccessPolicy functionality is provided in AccessPolicy. The

AccessControlAgent is mapped directly onto AccessDecision as both are responsible for deciding if the presented credentials allow a user to perform a particular operation.

Another issue that is intrinsically part of authentication and authorisation, in DPEs, is delegation. The main issue noted in [36], regarding delegation, is the inability to restrict where and when credentials can be delegated. This also includes what delegation modes (composite, simple) can be delegated. The issue is addressed by modifying several object interfaces. Firstly, the new administration object, DelegationPolicy will now also handle restrictions on where and when attributes can be delegated. Two new operations are introduced - *set_controls* and *get_controls*. These operations specify what privileges can be delegated, the delegation mode to be used, the number of invocations permitted and an expiration time for when these privileges can be delegated. This handles delegation from an administrative perspective. However, CORBA already allows privileges held in the Credentials object to be updated 'on-the-fly' using the *set_privileges* operation. Therefore the *set_control* and *get_control* operations need to be added to *Credentials*, so that delegated privileges within it can be controlled.

The mapping can be summarised in the following table.

New Security Framework	Comprehensive CORBASec (CCS)	Functionality
Access session-User Application, User Agent	UserAgent	Act as interface between user and system
AuthorisationAgent	PrincipalAuthenticator	Authentication Generation of identity attributes
User Agent	Credentials Current	Hold user related information
AccessControlAgent	AccessDecision RequiredRights	Decides if access is granted to a particular object/operation
AuthPolicy, AuthMech AccessPolicy, AccessMech	AuthPolicy, AuthMech AccessPolicy, AccessMech	Management of security services and mechanisms

Table 7-3 Authentication & Authorisation Security Service Object Mappings to CCS

7.2.4.2 CCS Authentication & Authorisation Example

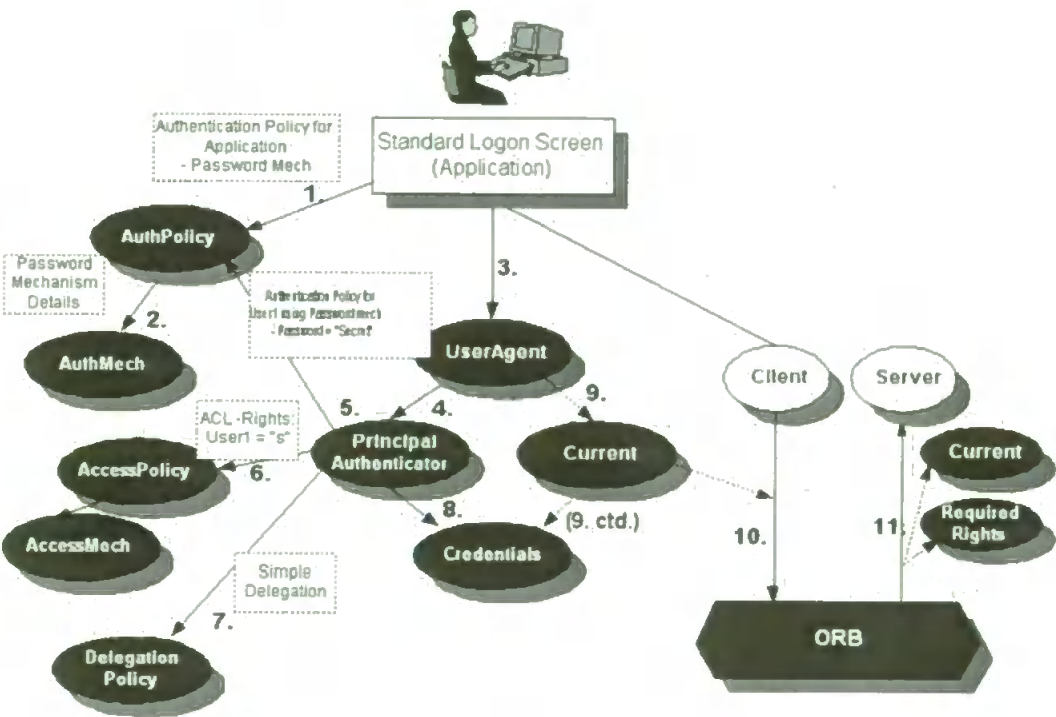


Figure 7-3 CCS Authentication

Figure 7-3 above illustrates the new method of authentication within the CCS. The process is described below.

A standardised logon screen for an application exists, it is a generic front end that can be modified to suit multiple authentication mechanisms.

1. **Find the current authentication mechanism for the application.** The logon module will query the AuthPolicy to find out what authentication policy is implemented for the current system. In this example a password mechanism identifier is specified in the application's AuthPolicy.
2. **Obtain the authentication mechanism details.** AuthPolicy will query AuthMech to find the details of the identified mechanism, e.g. what authentication parameters are required. The logon module screen is populated and in this example the system now waits for the user to enter a user ID and password.
3. **Principal completes login to system.** 'User1' now enters her user ID and password. The interaction is now taking place with the UserAgent object. This object will process the user authentication data, in this case a password and ID. However, if a smartcard logon were used, the UserAgent would process the user ID, user PIN and smartcard data.
4. **Authenticate the Principal.** The UserAgent, having all of the required authentication data, now calls the PrincipialAuthenticator to authenticate the user.

5. **Verify the authentication data.** The `PrincipalAuthenticator` will now query the `AuthPolicy` object to see if the user can be authenticated. `AuthPolicy` will confirm that `User1` with password “Secret” is a valid user of the system.
6. **Get the user’s access privileges.** The `PrincipalAuthenticator` now queries the `AccessPolicy` object to see what the user’s access privileges are. In the example, an ACL with a `Role` attribute is used. `User1` is defined as a ‘`Role1`’ role with access rights ‘s’.
7. **Get the delegation policy.** The `PrincipalAuthenticator` will query the `DelegationPolicy` object to see what the delegation mode is to be used. In the example, `SimpleDelegation` is used.
8. **Create the credentials object.** `PrincipalAuthenticator` returns the authenticated *Credentials* object to serve as the user’s security ticket. It contains attributes such as ID and privileges. This instance will hold the ‘`Role1`’ role with right ‘s’ and `SimpleDelegation` mode for `User1`.
9. **Set the credentials of the execution environment.** The `Credentials` object reference is passed to the `Current` object.
10. **Client invokes a secure method on a server.** The security service mediates the client/server interaction, by accessing the `Current` object to ensure that the interaction is in accordance with the security policy.
11. **Server executes the secure method.** The server can access the `Current` object to get information on the incoming client request, such as the client’s rights and privileges. The `RequiredRights` object can then be accessed to find what rights are required to access the server method. This information will allow the

server-side security service, i.e. the `AccessDecision` object, to make an informed access decision. If the client is allowed access, the server will execute the method. `User1`'s privileges can now be delegated to the server object if further invocations are required for the server to complete its operation.

The authentication and authorisation of a user to the system is complete.

7.2.5 Integrity & Confidentiality Enhancements

CORBASec deals with integrity and confidentiality together under the title of Quality of Protection (QoP). The new features that are added to QoP are:

- flexibility in configuring QoP by defining new policy objects;
- QoP for stored data.

7.2.5.1 CCS Integrity & Confidentiality Overview

Previously in CORBASec, there were no objects to independently handle integrity and confidentiality. The `SecureInvocationPolicy` had a `set_association_options` operation, which allowed the administrator to specify whether confidentiality and integrity were to be applied to secure invocations. The CCS, however, now has two new objects specifically dedicated to Quality of Protection (QoP), `QOPPolicy` and `QOPMech`. These objects are used to define a secure context between a client and server. The `SecureInvocationPolicy`, is now specifically devoted to secure associations between disparate security domains (see section 7.2.7).

Section 4.2.2 noted that there is an issue regarding the security of stored data. Security of stored data in this instance is defined as the implementation of a specified level of QoP on the data held in a persistent data store. It could be assumed that database integrity and security system would be able to handle these secure storage issues without DPE intervention. However, database integrity is not the same as security integrity. Database integrity refers to the accuracy, correctness and validity of data (referential integrity) [108], and does not specifically deal with the issue of unauthorised modification. With regard to database security, the mechanisms used are very much product specific and, in many cases, database security revolves around authentication, access control and the use of specific file formats that prevent file modification. However, this does not protect data that is illegally viewed, and some encryption mechanism has to be employed to ensure confidentiality and integrity. In addition, there are other methods of storage that can be employed (e.g. a flat file) and a DPE also has to be able to administer security for stored data in these implementations.

The CCS proposes the use of the QOPPolicy and QOPMechanism to also administer stored data security. QOPPolicy will use *get_stored_QOP_policy*, *set_stored_QOP_policy*, and *query_stored_QOP_policy* operations (as opposed to the *get_QOP_policy*, *set_QOP_policy* and *query_QOP_policy*). The reason that separate methods are required is that the administrator needs to distinguish between a secure communications context with an object, such as a database, and securing the data stored within a database. Therefore two policies can exist for the same object, but they will mean very different things. The parameters are almost identical to those used for secure contexts, except that the administrator does not need to specify a direction or a

message part that requires protection, because it is not dealing with a transmitted message, it is protecting stored data. The QOP mechanisms can apply the policy-specified encryption to a data structure before it is written to a database or file. The process will be reversed when the structure is then read.

The following table summarises the QoP mappings.

New Security Framework	CORBASec	Functionality
QoPAgent	Vault, SecurityContext	Negotiate and build a secure association
QoPPolicy, QoPMech	QoPPolicy, QoPMech	Management of secure association options

Table 7-4 QoP Security Service Object Mappings to CCS

7.2.5.2 Example of CCS Integrity & Confidentiality

Figure 7-4 below illustrates the new method of QoP within the CCS. The process is described below.

1. **Client invokes a secure method on a server.** The security service intercepts the client/server interaction. The Vault object is used to establish a secure context. It recognises the object as belonging within the trusted domain.
2. **Client checks the QoP policy.** The client queries the QoPPolicy to see what is required for a secure context between the client and server.
3. **QoPPolicy references QoPMech:** It also returns the mechanism to be used, via the QoPMech.
4. **Secure context negotiation begins.** A secure context will be initiated using the client's QoP.

5. **Server Vault finds its secure invocation requirements.** The server Vault intercepts the new request, and queries its QoPPolicy and QoPMech.
6. **Server returns its QoP.** It can use this information to finalise the negotiation with the client and so complete a secure context. Both client and server are now utilising SecurityContext objects and can communicate in accordance with the security policy. **Server queries a data storage object.** The server needs to query a data storage object DBStore, in order to complete the method invoked by the client. The DBStore queries the QoPPolicy to see if the data is securely stored and, if so, what QOP is applied. In this example, the data is stored with a QOP of Confidentiality, using DES to encrypt the data structures (in this scenario no secure communication with the DBStore was required, i.e. the data is transmitted in the plaintext but stored in an encrypted format).

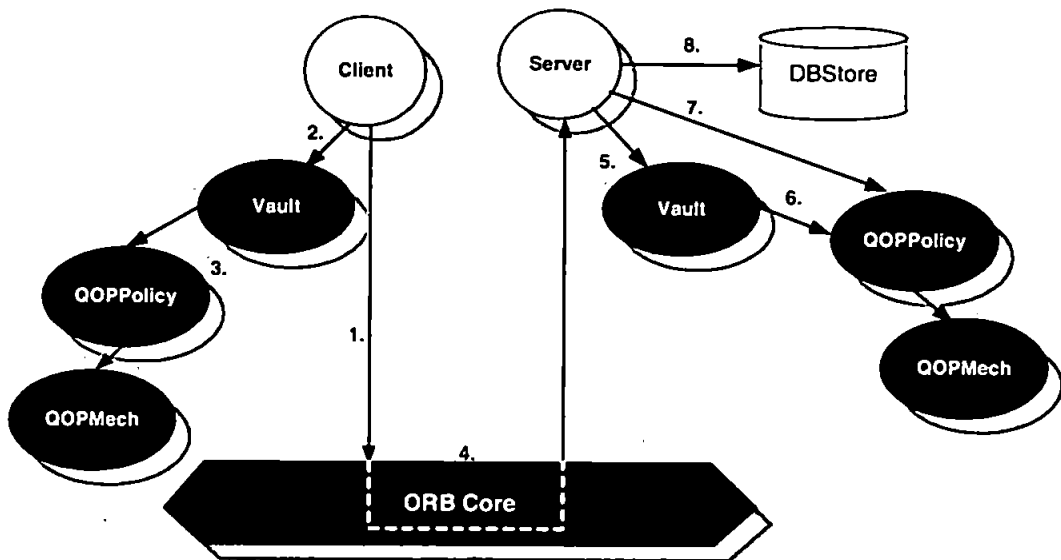


Figure 7-4 CCS Integrity and Confidentiality

7. **Server method can be completed.** When the record is read from the data structure it is decrypted using the appropriate mechanism, DES (Note: in this instance, any required key exchange is part of the read mechanism for the data structure). The data is returned to the server. The server can now complete the method invoked by the client. The response is returned via the secure context.

This completes the QoP events for transit and stored data.

7.2.6 Non-Repudiation & Audit Enhancements

Non-repudiation and audit have changed significantly in the CCS. Both employ new objects and provide more facilities.

7.2.6.1 CCS Non-Repudiation Overview

The first change is that Non-Repudiation is no longer considered an optional service as it was previously specified in CORBASec (Optional in CORBASec means that it was not available to security-unaware application – the non-repudiation interfaces had to be invoked by a security-aware application). It is available on every object invocation. However, the service is also configurable so that it does not provide an unacceptable overhead on ORB operations. Non-repudiation will be enforced on every object invocation, in accordance with the specified policy. This policy will be dictated by the new administration objects, NRPolicy and NRMech, which correspond to the TINA service objects of the same name. NRPolicy is used to configure the general non-repudiation policy – this means that it covers all of the non-repudiation facilities, mechanisms, evidence types and adjudicators. The NRMech object holds details of the non-repudiation mechanisms including authorities used and evidence

types. The NRCredentials object, as defined in CORBASec, is still used for evidence generation and verification. The NRCredentials information is held in contexts and User Agent.

Three other new objects are defined to provide the missing non-repudiation facilities as defined by the ISO – delivery, evidence storage and retrieval, and adjudication. The delivery service is made up of two key elements – a delivery authority and the NRDeliver object. The delivery authority (DA) is a TTP (see Section 3.3.2) that is identified in the NRPolicy authorities list. The authority list provides the name of the authority and its role, in this instance the role is “Delivery Authority”. NRDeliver uses the Delivery Authority, to provide a trusted delivery service. It makes use of the SecurityContext objects already defined in CORBASec, but creates new contexts to deliver its own tokens and data as opposed to using the client/server context that would already exist for an object invocation. For optimisation purposes, NRDeliver could use the existing context if it had the appropriate QoP, i.e. greater than or equal to the non-repudiation QoP specified in NRPolicy. NRDeliver will be able to send both generated and verified security tokens using the *NR_deliver_token* method. Another issue with the non-repudiation delivery authority is how it can prove that it performed its function. This is achieved by adding two more proofs to the process (see Section 3.2.5). This will include the client producing a *Proof of Submission* to provide irrefutable evidence that the client submitted the non-repudiation request to the Delivery Authority and secondly *Proof of Delivery* to create irrefutable evidence that the server received the original invocation and token from the Delivery Authority. These are created by the Delivery Authority for every delivery request and the

evidence tokens are stored in the client's evidence store. The NRDeliver functionality is mapped from the QoPAgent.

NRStore is the second of the new facility objects for non-repudiation, and corresponds directly to NRStore in the new TINA framework. It provides the interface to a storage facility for the tokens and certificates. It can add, get and query stored records relating to non-repudiation evidence, and does so using the *NR_record_set*, *NR_record_get* and *NR_record_query* operations.

The NRAdjudicator is mapped to its namesake in CCS. It is an interface to a notary that can make judgements on any disputes. A TTP will be used to verify evidence and then prove/disprove claims made by clients or servers. The adjudication process has two phases –the first is an on-line adjudication. The on-line adjudication allows the adjudicator process (without any human intervention) to validate the evidence tokens, i.e. make sure they have valid signatures and that the times are correct. If one evidence token is found to be invalid, then the process will be able to settle the dispute by deciding in favour of the valid token holder. However, if both tokens are valid, then one of three options is possible. If the adjudicator is implemented as an expert system, then it may still be able to settle the dispute based on some existing rules it contains. If the adjudicator still cannot settle the dispute, it can either signal for human intervention and request assistance in the adjudication process or it can return a judgement of “undecided”. This process is implementation independent and is not of any concern to the CORBA objects involved in the dispute.

The following table summarises the mappings between the framework and CCS.

New Security Framework	CORBA Sec	Functionality
NRCredentials	Contexts, UA	User credentials used for evidence generation etc
NRDeliver	QoPAgent	Creates a secure context to deliver non-repudiation tokens
NRStore	NRStore	Holds non-repudiation tokens securely
NRAdjudicator	NRAdjudicator	Makes judgements in the case of disputes
NRPolicy, NRMech	NRPolicy, NRMech	Management of secure association options

Table 7-5 CCS Non-Repudiation Mappings

7.2.6.2 CCS Non-Repudiation Example

The following section describes how the objects of the new Non-repudiation Service interact in the CORBA environment (see figure 7-5 below):

1. **Client invokes a secure method on a server.** The security service mediates the client/server interaction.
2. **Client checks the Non-repudiation policy.** The client knows it is about to invoke the server and so in preparation it queries the NRPolicy to see what non-repudiation actions need to be taken, if any. In this example, Proof of Origin is required.
3. **Non-repudiation mechanisms used are identified.** NRPolicy queries NRMech to find the non-repudiation mechanisms used, e.g. X.509 certificates, and the accepted TTP acting as Notary.
4. **The client requests the generation of irrefutable evidence.** The client requests the NRCredentials object to generate a token, using the appropriate mechanisms.

5. **The Client's token is securely delivered to the Server.** NRDeliver is used to deliver the token. The Delivery Authority used by NRDeliver was identified in NRPolicy. NRDeliver will query NRPolicy of both the client and server objects to find the non-repudiation QoP (NRQoP) defined for each. If they are different, they are then merged to find a QoP that will meet both of their requirements. The NRQoP will then be compared to the QoP of the invocation SecurityContext. If the NRQoP provides an equal or lower level of security, then NRDeliver uses the existing SecurityContext; otherwise it will create a new Security Context using the higher NRQoP level. Another function that has to be completed at this stage is the generation of evidence to ensure that NRDeliver has completed its task. This involves creating two proofs, firstly *Proof of Submission* to provide irrefutable evidence that the client submitted the non-repudiation request to the Delivery Authority and secondly *Proof of Delivery* to create irrefutable evidence that the server received the original invocation and token (not illustrated in the diagram for simplicity).
6. **The Client's token is stored for possible future adjudication.** During step 5, NRDeliver will have retrieved the name/identifier of the data store to be used by the client and server to hold evidence. In this example both are using a single data store for the domain. NRDeliver will have to create another SecurityContext to deliver the token to NRStore; if that store is not available in server object, e.g. as in the example a separate data store is used by all the objects. The token is stored using the *add_record* method

on the NRStore object. In addition, the Proof of Creation and Proof of Submission described in step 5 are also stored in the client's data store.

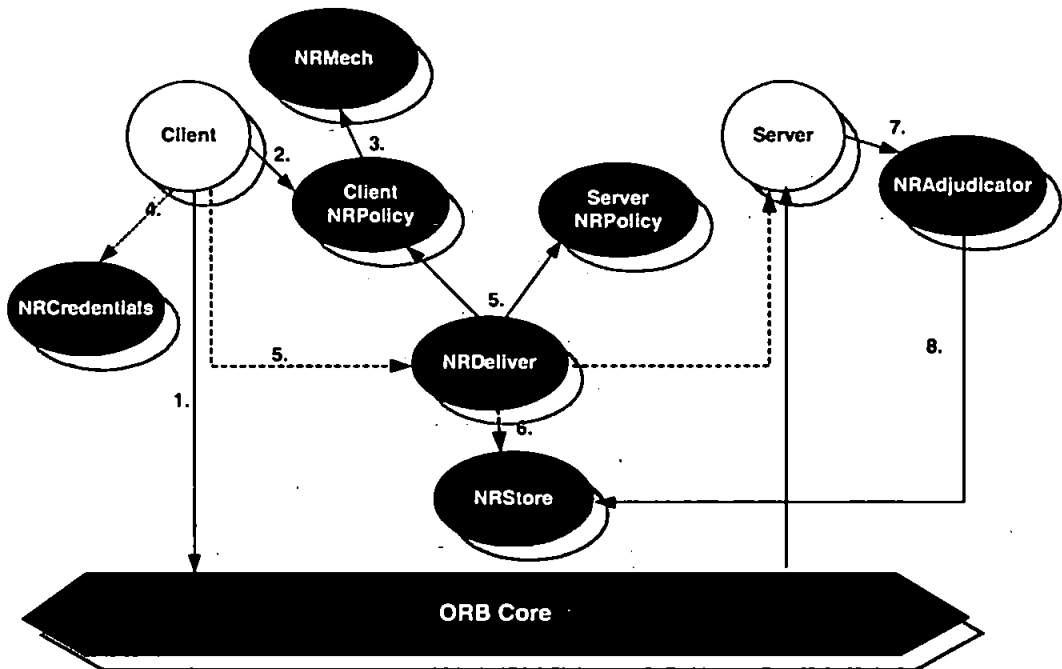


Figure 7-5 CCS Non-Repudiation

7. **The Server may dispute the invocation call origin at some later time.**
The server can call on the NRAdjudicator to settle the dispute.
8. **The dispute is deliberated and settled.** The NRAdjudicator can query the NRStore to validate the client and server claims, i.e. validate their *supporting* tokens. This would be done via a secure context through NRDeliver (not shown in the diagram for simplicity). The NRAdjudicator will return a decision and the supporting token.

The non-repudiation action is completed.

7.2.6.3 CCS Auditing Overview

Within the auditing service, there were several facilities, which were not catered for in the original CORBASec specification. New objects have been introduced to accommodate administration, filtering, routing, reporting and analysis. Firstly, the administration objects, AuditPolicy and AuditMech. Although CORBASec specified an AuditPolicy, this has been significantly modified. The original operations *set_audit_selectors*, *clear_audit_selectors*, *replace_audit_selectors* and *get_audit_selectors* remain in place to manage the event types to be audited. They are now extended to include the selection of which AuditAnalyser and AuditResponder (see below) are to be used with these selectors. The AuditMech allows the administrator to manage all the different mechanisms employed in the auditing facility (this includes analyser, responder, and knowledge-base mechanisms).

AuditDecision was specified in CORBASec, but its function has been modified. Previously it was used to decide if an audit record should be written to an AuditChannel. It now just decides if the event needs to be audited, using the *audit_needed* operation, because the AuditChannel has a new purpose (described below).

New objects for the sampler, AuditSamplerAgent, and knowledge base, AuditKB, are not required. The sampler is an object that is deployed in the system, but is not accessed by any other object and therefore does not need an interface definition in CORBA. Sampling will be achieved through the security interceptor. The knowledge base does not need any interface because some of its data is already handled in other CORBA objects, e.g. the security policy information is available through administration objects and the security log information is available in the AuditTrail

(see below). Therefore the only information required will be the profile, analysis and response information and it will be utilised by the analyser and responder. This will be mechanism dependent and so will not require an object definition to be available.

After the AuditDecision has decided that an event needs to be audited, the new AuditAnalyser analyses the information using the chosen analysis mechanism (e.g. rule-based, profiling, etc.) to decide if the event is anomalous. The analyser employs two operations – *analyse_data* and *justify*. The former is used to request AuditAnalyser to analyse the event data, indicate the analysis result (i.e. whether a system violation has occurred or whether suspicion levels should be raised) and produce an analysis token; the latter is used to provide a full justification of the analysis results, if required.

The analysis result and token are then sent to the AuditResponder, which decides what to do using the *define_response* operation. This operation decides what AuditChannel will be used to implement the appropriate response and it will generate the corresponding data to be processed by that channel's log or action.

The previously specified AuditChannel object in CORBASec was linked to a specific AuditDecision object and used a single operation *audit_write* to write an audit record. However, AuditChannels are now linked to two new objects, either an AuditTrail or an AuditAction, e.g. alarms. This is accomplished in the AuditPolicy object using the *set_audit_channel* method. This means that several channels can now exist simultaneously for a single AuditDecision, providing greater flexibility and efficiency from the single object. The AuditChannel can, if required, establish a secure context to the log or event action. This will be specified by QoPPolicy or

SecureInvocationPolicy objects and is implemented by using the secure invocation objects. This was not the case in the original CORBASec specification. The AuditChannel now writes the *audit_data* specified by AuditResponder to its linked object, i.e. trail or action.

The AuditTrail is a new object that represents an audit log. As the log now has a standard interface, it can be easily accessed and queried. This will facilitate the generation of user-friendly interfaces to it. AuditTrail employs *read_record*, *write_record* and *query_record* operations. The AuditActions object will allow the administrator to define other generic responses to an audit event, e.g. sounding an alarm or emailing a security supervisor. AuditActions uses the *get_action_info* operation to return details of what the required action is and *execute_action* to perform it.

7.2.6.4 CCS Audit Example

The following section describes how the objects of the new Audit Service interact in the CORBA environment.

1. **Client invokes a secure method on a server.** The security service mediates the client/server interaction.
2. **Client checks if the event should be audited.** The client will query AuditDecision to see if the event should be audited.
3. **AuditDecision checks the Audit policy.** The client queries the AuditPolicy to see if the action should be audited. In this example the server invocation is an auditable event. AuditPolicy can query AuditMech to identify the specifics of

the auditing mechanisms. `AuditDecision` returns a response to the user, indicating that the event should be audited and identifying the `AuditAnalyser` to be used.

4. **Client initiates the audit.** The client invokes the `AuditAnalyser` identified by `AuditDecision`. The possibility of multiple instances of `AuditAnalyser` exists because of the multiple types of analysis mechanism that may be employed.
5. **Information is accessed to help analysis.** The `AuditAnalyser` can query the data in the knowledge base to help it complete its analysis. This occurs at the mechanism level.
6. **Appropriate response is formulated.** The `AuditAnalyser` then passes on its analysis to the `AuditResponder`, where the appropriate response to the audited event will be taken. The response can vary from writing a record to the audit log, sounding an alarm, sending an alert message to the administrator's screen, or even shutting down a specific application. (Note the `AuditResponder` can also access the knowledge base in order to formulate the appropriate response; this is not shown in the example).
7. **Alert sent to administrator screen.** In this example, the `AuditResponder` decided that two actions were to be taken. The first action is to send an alert to the administrator's screen. This is accomplished by invoking the `AuditActions` object where the alert function is defined, via the `AuditChannel`. `AuditChannel` will provide a suitable context if required. If the data is considered security sensitive, an appropriate secure context will be established to deliver the data

to the AuditAction, thereby preventing any unauthorised access to the data during transit.

8. **Audit record written to log.** The second response that AuditResponder required was to write the audit record to a log. The AuditTrail object is the interface to the audit log, and again if required, AuditChannel will provide a secure context to deliver the data.

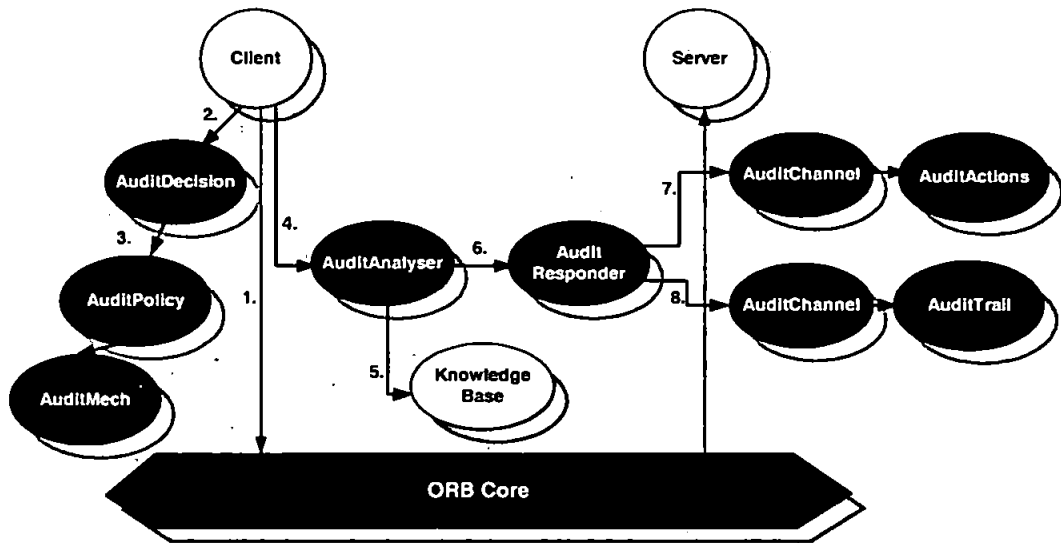


Figure 7-6 CCS Audit

The auditing of the event is completed.

7.2.7 Secure Interoperability

The mapping of security interoperability has two parts, firstly the mapping of the protocol and secondly the mapping of objects.

The protocol defined in chapter 5 (see section 5.4.2) is mapped to the OMG's Common Secure Interoperability (CSI) [37] protocol. The table below lists the CSI

message types, their function and the Interoperability Protocol Messages to which they can be mapped.

CSI Message	Function	DPE Message
EstablishContext	Passed by the client to the target when a secure context needs to be established.	CreateContext
ContinueEstablishContext	Used by the client or target during context establishment to pass further messages to its peer as part of establishing the context.	NegotiateContext
CompleteEstablishContext	Returned by the target to indicate that the association has been established.	AcceptContext
DiscardEstablishContext	Used to indicate to the receiver that the sender of the message has discarded the identified context. Once the message has been sent the sender will not send further messages within the context.	DeleteContext
MessageError	Used to indicate an error detected in attempting to establish an association either due to a message protocol error or a context creation error.	ErrorContext
MessageInContext	When a secure context is established, messages are sent within the context using the MessageInContext message.	ProcessContext

Figure 7-7 CSI Message Types

The object mapping involves adding a new object, the DomainMapping, and significantly altering the functionality of the SecureInvocationPolicy object. Both of these objects provide the functionality of the DMA and SecureInvocationPolicy in the TINA model (see section 5.4.3). However the SecureInvocationAgent function is added to the CORBASec Vault and SecurityContext objects. As these objects are already providing the negotiation process of the QoPAgent, the functionality only

needs to be extended to address the issues of interoperability across disparate domains.

The following table summarises the secure interoperability mappings.

New Security Framework	CORBASec	Functionality
DMA	DomainMapping	Define mappings between two trust domains
SecureInvocation Agent QoPAgent	Vault, SecurityContext	Negotiate and build a secure association between different trust domains
SIPolicy	SecureInvocation Policy	Management of secure association options between different trust domains

Table 7-6 Secure Interoperability Service Object Mappings to CCS

7.2.8 Security-Aware Trader

The CORBA Trader [109] is an implementation of the ODP Trader, as describe in chapter 6. Therefore the mapping to CORBA of the new security-aware trader is simplified as it can be accomplished by following the modification summary stated in section 6.5.5 (the modifications are again listed below):

1. New Trader security attributes;
2. Use of 'mandatory, readonly' security property in Repository;
3. New Registry security property;
4. Modified Admin interface, inherits Security Attributes;
5. Modified Lookup interface inherits Security Attributes;
6. Modified Registry interface inherits Security Attributes;
7. Modified Proxy interface inherits Security Attributes;

8. Modified Link interface, inherits Security Attributes, and new link policy *Link_security*;
9. Use of the new Security Framework, including secure interoperability;
10. Use of security-aware DPE services.

The new security-aware trader IDL for CORBA is available in Appendix B. It covers the modifications 1 to 8. However, the modifications for 9 and 10 do not produce any IDL changes, but rather functional changes in how the Traders interact securely with each other when they reside in disparate domains, and how they interact with the security service.

7.3 Summary

The DPE specifications provided by TINA are high level and do not address many implementation issues. To ensure that the new Security Framework is applicable, it was considered necessary to map it to a DPE implementation specification. CORBA is the leading specification and therefore was used for the mapping exercise. The existing CORBA Security Service has been significantly re-designed to provide a new more comprehensive and configurable service, in order to meet the needs of a DPE. Firstly, the new administration structure, which facilitates service and mechanism independence, is provided through the introduction of a new Policy super-class and the use of this class to build service and mechanism management objects for each of the service facilities. Secondly, each facility within the security service is also enhanced. Audit is extended to include new monitoring (IDS) and data filtering

facilities. Non-Repudiation provides new delivery and storage facilities. An interface to an adjudicator is also provided to help settle disputes. Integrity and Confidentiality are extended to provide greater configurability and deal with the issue of stored data QoP. Thirdly, secure interoperability has extended its negotiation capabilities to handle all of the new Security Service facilities, and is now capable of negotiating a secure context between security domains with conflicting policies. Finally, the CORBA Trader is now security-aware and can interact with the security service to eliminate the trading security threats identified in section 6.4.

All of these mappings go beyond any enhancements planned by the OMG in the future [110, 111] and far exceed the current realisation of security within current implementations of CORBA. The next chapter will now investigate the prototype implementation of the new mappings to CORBA and verify their feasibility.

8. Proof of Concept

8.1 Introduction

The previous chapters have covered the core concepts of the new security framework – the new security service, new secure interoperability and the security-aware DPE service (Trader). All of these features address security vulnerabilities in DPEs and offer improved secure functionality within the distributed environment. However, a theoretical specification alone is not adequate if the research is to prove useful in the world of distributed object systems. Therefore, the purpose of the implementation is to build demonstration software that will act as a Proof of Concept for the theoretical research defined. The prototype can then be used in the verification work.

This chapter will look at both implementation and verification. With regard to implementation, it will define the different aspects of the work - hardware, software, the IDL defined and how the object implementations were achieved. Issues relating to the implemented IDL, which were identified during the process, will also be examined. The verification will be performed in two ways.

- Performance Modelling;
- Standardisation (including implementation issues);

Firstly, performance modelling of the work is required to determine the implications of implementing the new security services in real-world environments, and not just the research environment described in this chapter. Secondly, the work needs to be acceptable within the current standards for ISE and DPEs. These standards have also

progressed since the initiation of the research and the work will be evaluated to ensure that it is still new and novel. Finally, the verification will entail looking at current real-world problems that the new security framework will solve.

8.2 The Proof of Concept Prototype

With regard to the scope of the implementation, it was decided that it should include all three aspects of the work:

1. Comprehensive CORBAMSec (CCS);
2. Comprehensive CORBA Secure Interoperability Service;
3. CORBA Security-aware Trader.

Each one needs to be a workable part, in order for the whole security solution to be implementable. Within the CCS, the major security facilities implemented are as follows:

- Authentication;
- Access Control;
- Integrity/Confidentiality (QOP);
- Non-Repudiation.

These enhanced facilities include all the new objects as defined in chapter 7, both at administration and operational levels. The only facilities not implemented for the service were the Audit and Recovery. Although Audit was theoretically defined in section 7.2.6.3 (based on the components in Chapter 4), neither Audit nor Recovery was implementable within the timeframe of the research. The implementation was

restricted to a selection of facilities that were considered sufficient to provide a prototype for the CCS.

The Comprehensive Secure Interoperability Service has also been implemented. It operates at both the mechanism and policy levels defined in section 7.2.7 (based on the components from chapter 5). It includes the extension of the secure context objects so that they can handle the new policy configuration negotiations within the new administrative structure of the CCS, as well as the introduction of a new object to provide mappings between these configurations.

The final part of this implementation is the construction of a security-aware service, which is the Security-aware Trader as described in chapter 6. The Trader implemented is a 'Stand-alone Trader', as defined in [59], implements the Lookup, Register and Admin interfaces. Currently available trader implementations are generally only *Query* or *Simple Traders*, i.e. they implement the Lookup or Lookup and Register interfaces.

The justification for this research has already been covered in previous chapters and has shown that DPE specifications for Security, Interoperability and the Trading service, while providing a basis for secure operations, are still incomplete (CORBA was used as an illustrative example). The inadequate management and operational facilities leave DPEs open to many security vulnerabilities, which are listed in sections 4.3, 5.2, and 6.3. This prototype verifies the work by defining a new DPE specification framework (CORBA) comprised of new objects, administrative structures, policy configuration structures and modes of operation between services to ensure greater security. These issues can be summarised as follows:

Topic	Problem	Research Solution
Security	Missing facilities	Provision of facilities with the addition of new objects
	Inadequate administration	Design and Implementation of new administration system
	No Mechanism Independence	Separation of mechanism and service management
Interoperability	Insufficient negotiation abilities	New negotiation abilities of interoperability objects
	Unable to handle disparate domains, e.g. different policies	New structures introduced to handle conflicting policy negotiations and inter-domain mappings
DPE Service	Not security aware	Creation of security-aware service that utilises the CCS

Figure 8-1 Summary of Issues in Research

8.2.1 Implementation of the Prototype

The following subsections describe how the prototype was implemented. It details the hardware platform, software configuration, structures used and how the Interface Definition Language (IDL - see section 2.4.1) was used to ensure implementation-independence at several levels within the demonstration software. Further prototype hardware and software information is provided in Appendix D.

8.2.1.1 IDL

CORBA IDL allows the specification of object interfaces in an implementation-independent manner (see section 2.4.1). It is used by the middleware implementation, i.e. Orbix, to generate C++ code for the implementation. The IDL interface code generates the client stub and server skeleton (see figure 2-8 in section 2.4.2). In the implementation, three features were implemented using IDL-defined interfaces:

- Comprehensive CORBASec;
- Security-aware Trader;
- GSS-API.

The CCS and Security-aware Trader were defined in IDL as they are new services and had to be proven to be implementable and operational within the distributed environment. The GSS-API server was defined in IDL because it was necessary to ensure that the other services could utilise GSS-API in order to preserve standardisation.

The CCS IDL has several modules:

- **Security**: defines the data types used in the service;
- **SecurityLevel1**: defines Level 1 security;
- **SecurityLevel2**: defines Level 2 security and basically includes all the operational level objects required by the service;
- **SecurityAdmin**: defines the new security administration features;
- **SECIOP**: defines the enhanced Secure IIOP required for the new service.

In CORBA there were two other modules, the NRService and SecurityReplaceable modules; both have been integrated in to the SecurityLevel2 and SecurityAdmin modules. There are two reasons for this. Firstly, non-repudiation is no longer an optional service and, therefore, is now included in the main modules so that it is accessible with all the other facilities. Secondly, security replaceability is no longer required as mechanism and service independence is now built into the CORBA

security structure and can transparently handle any replacement of mechanism and policy objects that is required.

The Security-Aware Trader IDL module structure has not changed from the original CORBA structure. It is still as listed below:

- **CosTrading**: defines the Security-Aware Trader that contains attributes, including the new security attributes, and core interfaces, i.e. Admin, Lookup, Register, Proxy, Link, and OfferIterator;
- **CosTradingDynamic**: defines the Trader's Dynamic Property interface, i.e. DynamicPropEval;
- **CosTradingRepos**: defines the Trader's Service Type Repository interface, i.e. ServiceTypeRepository.

While the CORBA object interfaces are still used, some of the parameter lists are extended and a new security attribute interface has been added along with the new link policy, *Link_security*.

The GSS-API IDL was created from the version 2 specification [85]. It contains a single module:

- **GSSAPI**: defines all the data types and operations required by the version 2 specification.

The full IDL descriptions for these services and GSS-API are available in Appendices A, B and C respectively.

8.2.1.2 Object Implementation

This section will look at the structure of the Prototype software, what objects were implemented and how they were utilised.

8.2.1.2.1 Implementation structure

The object implementation was accomplished in using Visual C++, cryptlib and Microsoft Access in the Orbix environment. It was structured as depicted in figure 8-2 below:

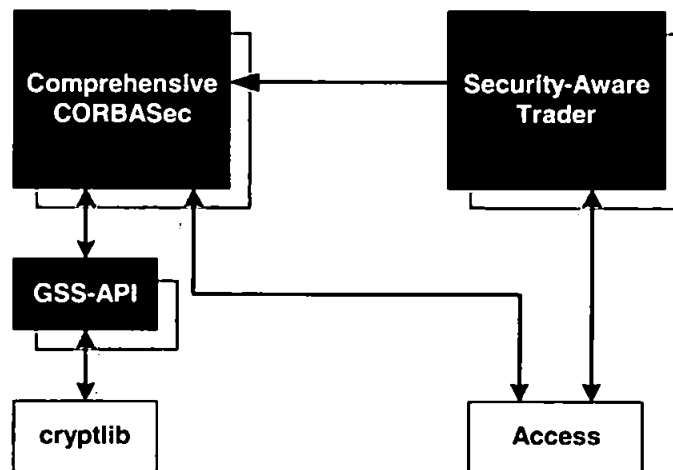


Figure 8-2 Object Implementation

The GSS-API server used the cryptlib software to provide the mechanisms required to accomplish the security context, credential and protection operations defined in the service. The CCS was then able to utilise these generic operations to complete its defined methods on the security objects. It also used Microsoft Access as a method of persistent storage for administrative data held in the security administration objects.

The Security-Aware Trader utilised Microsoft Access as a mechanism for persistent storage of trading data, this includes attribute values and *Repository* and *Registry*

information. The Trader will use the CCS to accomplish any security-related functions that are required – such functionality or behaviour is indicated by the security attributes.

8.2.1.2.2 Objects Implemented

As previously stated the prototype implementation does not implement all of the newly defined security facilities. Figure 8-3 below clearly illustrates which objects have been implemented (it is based upon figure 2-11 in section 2.6).

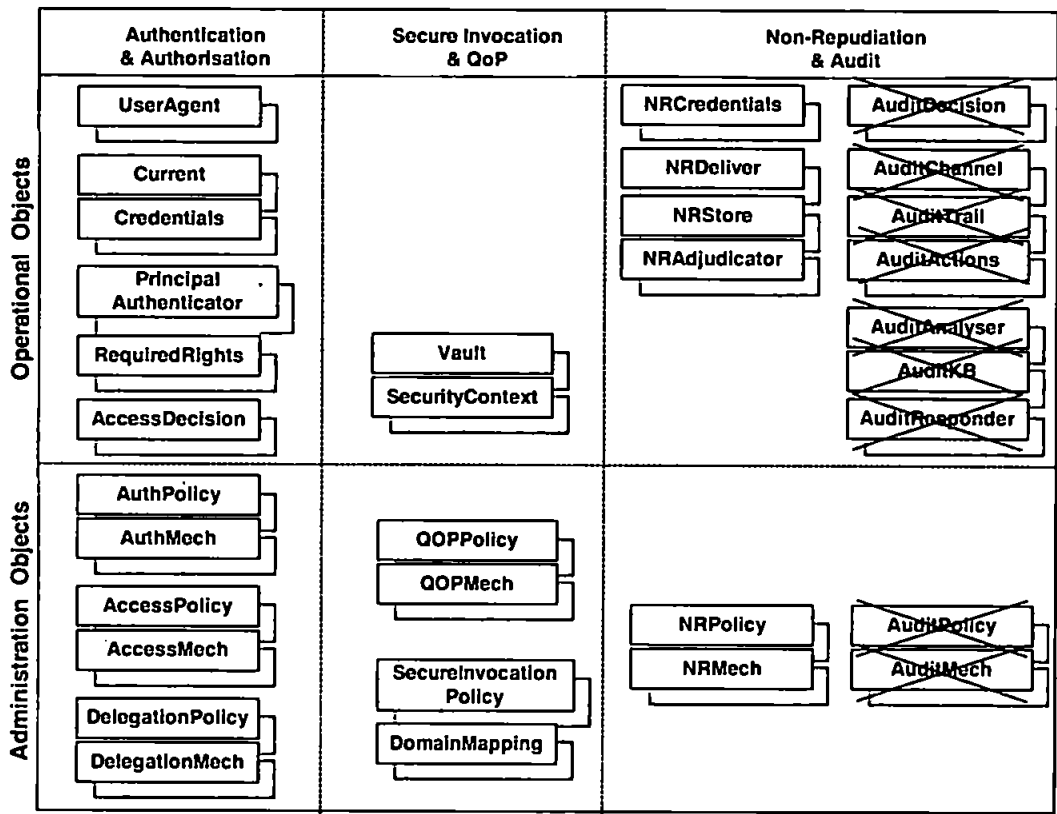


Figure 8-3 CCS Objects Implemented

In the Security-Aware Trader, the Lookup, Register and Admin interfaces (see section 6.5) are implemented. It was not necessary to implement the Link or Proxy interfaces

to prove the concept of security-aware services. Figure 8-4 illustrates the interfaces implemented in the Security-Aware Trader.

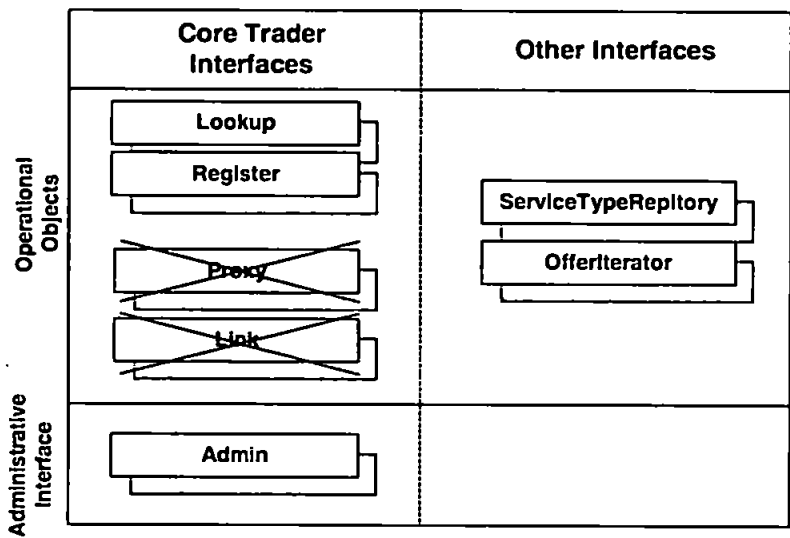


Figure 8-4 Security-Aware Trader Interfaces Implemented

Within the GSS-API IDL, the core operations relating to credential management, context-level (establishment and management), and message-level (integrity and confidentiality) were implemented. The support calls were not implemented for the demonstration software, as the purpose of utilising GSS-API operations was to use a standardised API within the CCS.

8.2.1.2.3 How objects were utilised in the implementation

There are two basic methods of utilising the security objects:

- Interceptor initiated calls;
- Direct call from security-aware applications.

Both of the methods are utilised in the implementation, and are necessary as they can correspond to the security-inactive and security-active invocations (see section 4.2.3). This section will illustrate how this is accomplished by showing some examples of where these methods are applied in the demonstration software.

Interceptor initiated calls are utilised by the Security-Aware Trader. A security interceptor has been created in the Security-Aware Trader. When a call is invoked on the Trader, the interceptor intercepts it. It can then interrogate the Trader's security attributes and apply the appropriate security facilities to the inbound call. For example, as show in figure 8-5 below, if a Security-aware Trader is using the *access_control_trader* and *nr_trade* attributes, the interceptor will firstly run an access control on the call, using *AccessDecision*, to ensure that the client is authorised to access the Trader. If the client has authorisation, the interceptor will then check the Trader's non-repudiation policy (*NRPolicy*) and find the evidence types required and comply with the policy. If '*proof of origin*' is required, then the interceptor will ask the client for such evidence. When it is received and verified, it will be stored in case of future disputes. The interceptor will then forward the call to the Trader.

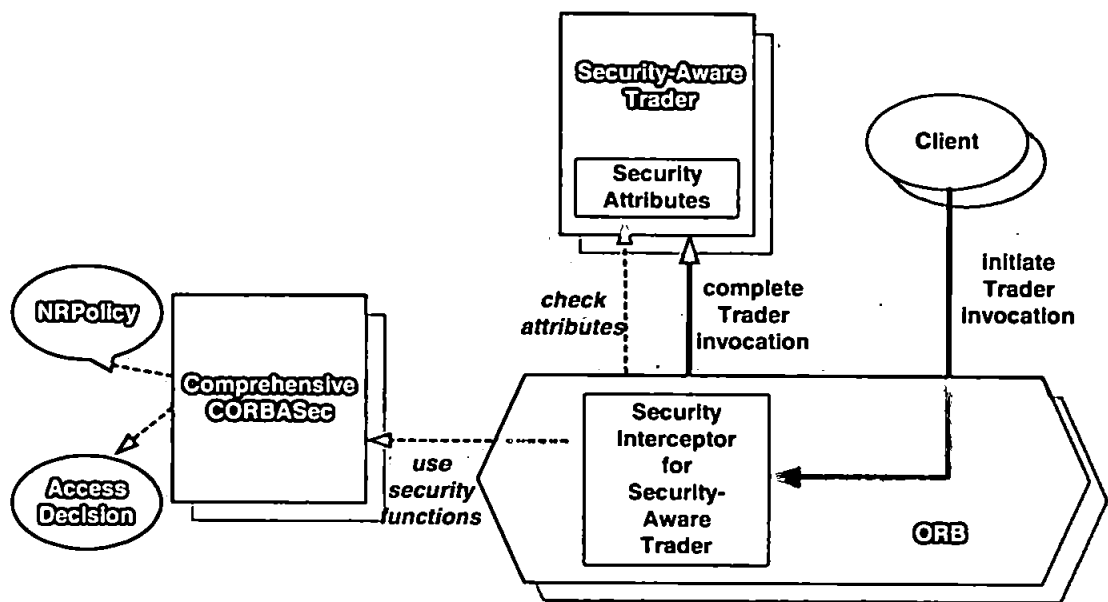


Figure 8-5 Interceptor initiated calls

Security interceptors can process all call invocations when security is to be applied across a distributed system. This will ensure that the domain security policy will be applied to all application calls whether they are security-aware or not. This is how Level 1 security is applied. However, as in the example above, security-aware applications or services can also utilise interceptors to process all incoming invocations.

The second method of utilising objects is that of a direct call from security-aware applications. In this case, applications can make calls on security objects directly as opposed to relying on interceptors. An example of such an application in the implementation software is during the logon process. The logon facility initially queries the AuthMech, via the AuthPolicy object, to find the authentication mechanism used, e.g. password or smartcard. In this instance, the password logon is defined as the required mechanism for the system and so a password logon screen is presented. After the user has entered his ID and password, the logon facility generates

a UserAgent to act on the user's behalf. The UserAgent will then initiate a call to the PrincipalAuthenticator in an attempt to authenticate the user. If successful, a Credentials object will be generated for the user and he will be allowed access to the remote system.

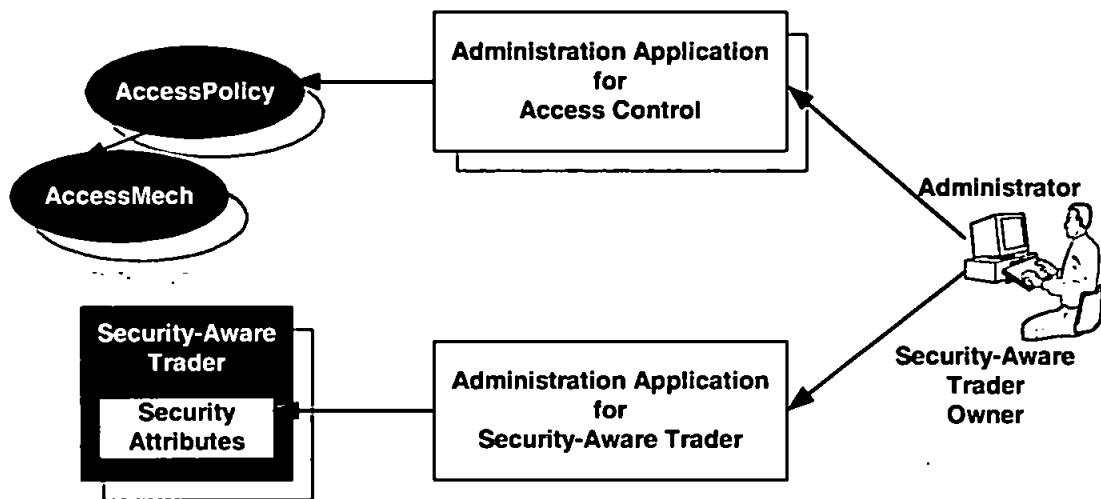


Figure 8-6 Direct call on objects

Another scenario in which an application can call security objects directly is administrative applications as illustrated in figure 8-6 above. The demonstration provides administration applications for each of the security services and does so by accessing the administration objects, both service and mechanism level, to populate, verify and update the security policies. An administrative application has also been written to allow the Trader owner update the features used in a Security-aware Trader, i.e. set the appropriate security attributes.

8.2.1.3 Implementation Issues

The first issue relates to the number of parameters that were used in the IDL operations, e.g. the Vault object's *init_security_context* method clearly shows how the number of parameters can become very long. The bold highlight shows the new parameters added to the IDL.

```

Security::AssociationStatus init_security_context (
    in CredentialsList                creds_list,
    in Security::SecurityName         target_security_name,
    in Object                         target,
    in Security::OptionsDirectionPairList association_options,
    in Security::MechanismType        mechanism,
    in Security::Opaque               mech_data,
    in Security::Opaque               chan_binding,
    inout short                       lifetime_rec,
    out Security::MechanismType        out_mechanism,
    out boolean                       deleg_state,
    out boolean                       mutual_state,
    out boolean                       replay_det_state,
    out boolean                       sequence_state,
    out boolean                       anon_state,
    out boolean                       trans_state,
    out boolean                       prot_ready_state,
    out boolean                       conf_avail,
    out boolean                       integ_avail,
    out Security::Opaque               security_token,
    out SecurityContext                security_context,
    out Security::errormsg              error,
    out Security::major_status          major_error,
    out Security::minor_status          minor_error
);

```

The above method was constructed in this manner so that it was easily map to the GSS-API method, *GSS_Init_sec_context* (see below) [85].

```
OM_uint32 GSS_Init_sec_context(
    in gss_cred_id_t claimant_cred_handle,
    in short input_context_handle,
    in internalname target_name,
    inout obj_id_seq mech_type,
    in boolean deleg_req_flag,
    in boolean mutual_req_flag,
    in boolean replay_det_req_flag,
    in boolean sequence_req_flag,
    in boolean anon_req_flag,
    in short lifetime_req,
    in octetstring chan_bindings,
    in ByteBuffer input_token,
    in short tincount,
    out short major_status,
    out short minor_status,
    out contexthandle output_context_handle,
    out ByteBuffer output_token,
    out short tcount,
    out boolean deleg_state,
    out boolean mutual_state,
    out boolean replay_det_state,
    out boolean sequence_state,
    out boolean anon_state,
    out boolean trans_state,
    out boolean prot_ready_state,
    out boolean conf_avail,
    out boolean integ_avail,
    out short lifetime_rec
);
```

There are many other similar examples in the new IDL. GSS-API provides a generic and standardised method to create a secure context and so is used by many security implementers. Therefore it was used as the basis for the security implementation in the demonstrator. However, the number of parameters in IDL is generally smaller, as this helps reduce programmer error. Therefore it may be more conducive to place the large number of parameters in a structure instead of listing them sequentially.

The second issues relates to the number of invocations required. While the demonstrator was able to implement and operate all of the interfaces, it was realised that the number of invocations required had substantially increased. This issue is analysed in detail in the section 8.3.1 later in the chapter.

8.2.2 A Practical Demonstration Scenario

The prototype incorporates the use of a demonstration scenario, which itself involves two applications, 'Local Application' and 'Remote Application', both of which have a set of facilities within an new CORBA environment, i.e. they both utilise the CCS, new Secure Interoperability Service, and 'Local Application' also utilises the Security-Aware Trader. The scenario involves 'Local Application' users completing a service authorisation request. In order to do so they need to access user profile information in 'Remote Application', for the user making the service request.

'Local Application' has the following applications:

- Logon Application;
- Service Authorisation Application;
- Security-Aware Trader to find datastore services;

- Security Administration Application;
- Security Aware Trader Administration Application.

‘Remote Application’ has the following Application:

- User Information Application, which can be remotely accessed by other applications with the proper security authorisation.

For the purposes of the demonstration, a domain mapping has been agreed between the administrators of the two security domains where ‘Local Application’ and ‘Remote Application’ reside.

The following table 8-1, illustrates the users in ‘Local Application’ and their roles and the applications that they are authorised to use:

Users		Local Application				Remote Application
Name	Role	Service Request	Trader Service	Trader Admin.	Security Admin.	User Information
User1	‘Role1’	X	X	X		X
User2	‘Role2’	X	X			
Admin	‘administrator’				X	

Table 8 - 1User Roles and Authorised Access

The main features of the demonstration are show as follows:

- **CCS with new and enhanced facilities:** the ‘Local Application’ will be able to operate in two modes, with security switched on or off. When security is on,

the CCS will authenticate users and only allow them to access authorised applications. It will also employ the appropriate QoP and non-repudiation functions;

- **New Secure Interoperability Service:** When users try to access the 'Remote Application' User Information Facility, the new Secure Interoperability will come into operation. If authorised, the user will be allowed access to the application;
- **New Administration Structure:** Only Admin is authorised to access the Administration applications for the CCS. Admin will be able to make changes to certain policies, and the updates will be reflected the next time a user tries to access the 'Local Application';
- **Security-aware Trader:** The Security-Aware Trader offers datastore services, e.g. a user can query the Trader to find the most appropriate datastore to store 'Local Application' details. The Security-Aware Trader is administered by its owner, i.e. User1 can update the security attributes. The Trader will initially operate as a Public Trader, i.e. with no security, and then after an administration update it will operate as a Security-Aware Trader. The differences can be illustrated by observing who can access the Trader and what service offers are returned.

Figure 8-7 below illustrates the authorised paths through the Prototype, when the new CCS is in operation. A security service is active is active in both domains. User2 is not authorised by the security service to access the remote application or to authorise a service request.

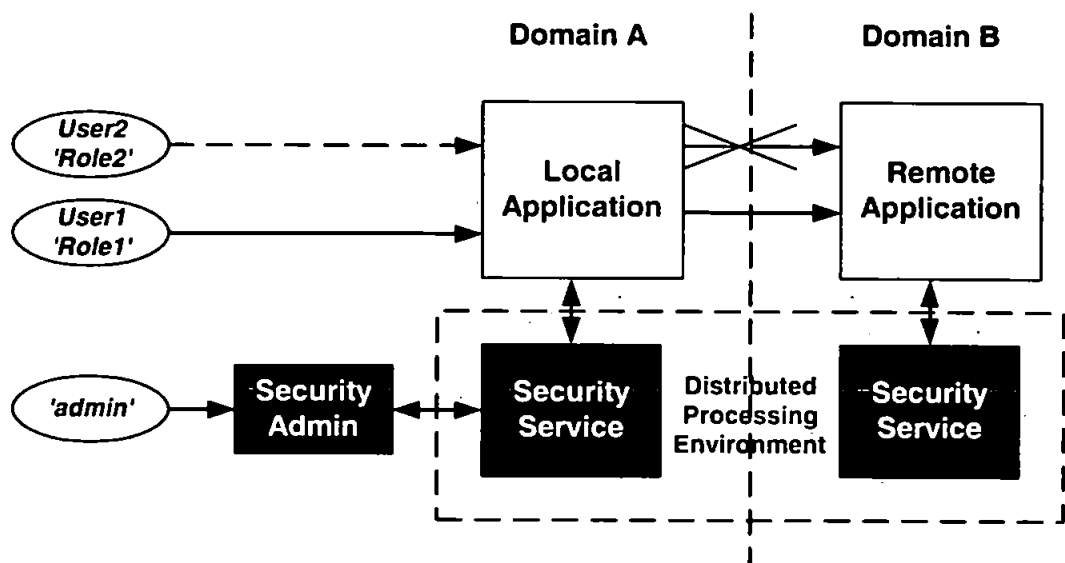


Figure 8-7 Authorised paths through the demo with New Security Service
Only an administrator is allowed to access the security service through the administration console, see figure 8-8 below.

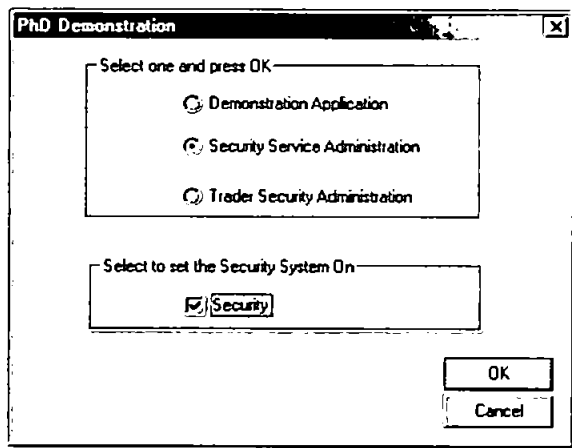


Figure 8-8 Administration Selection screen

Any access request, other than those displayed, should be denied. Without the new Security Service, User2 would be able to access all functions – there would be no protection.

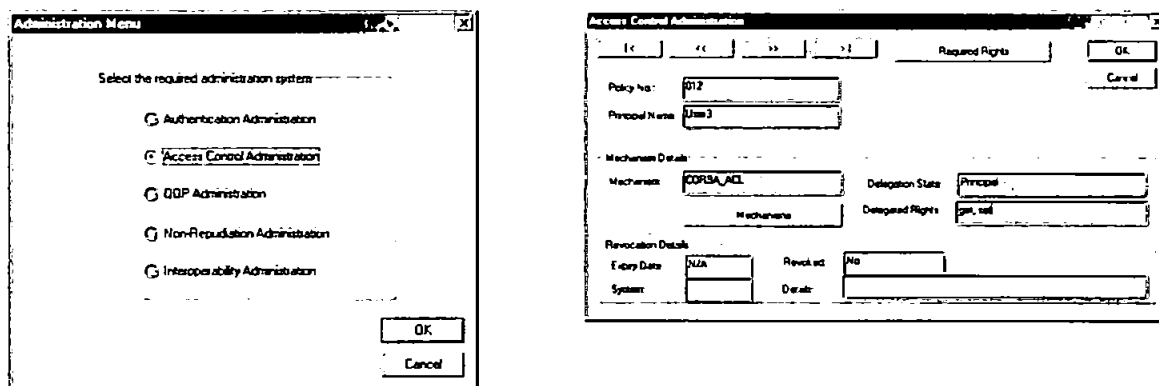


Figure 8-9 Security Service Administration Screens

The administrator will be able to select the service he wants to manage by selecting it from the security menu as shown in figure 8-9 above. The details of how the services are administered can be entered using the individual security service screens, such as the access control administration screen also illustrated in figure 8-9.

Figure 8-10 below, illustrates the authorised paths through the Security-aware Trader demonstrator. User1, as the trader owner, is the only authorised user allowed to access the Trader administration console, i.e. she is the only one allowed to set the Trader attributes. Both User1 and User2 are allowed to access the Trader; however, when the new security service is in operation, User2 will only see the trader offers that he is authorised to see. User1, acting with 'Role1', is allowed to view all the offers in the Security-aware trader. Without the new security, User2 would either be denied any access to the trader, or he would have full access and therefore the service offers would be unprotected.

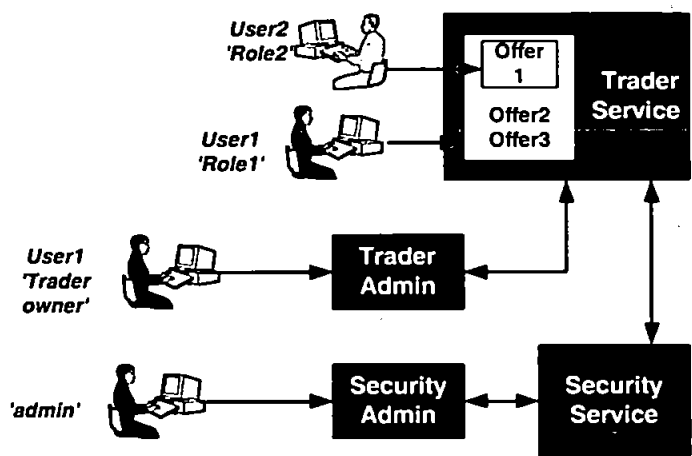


Figure 8-10 Authorised paths through Trader Demo with New Security Service

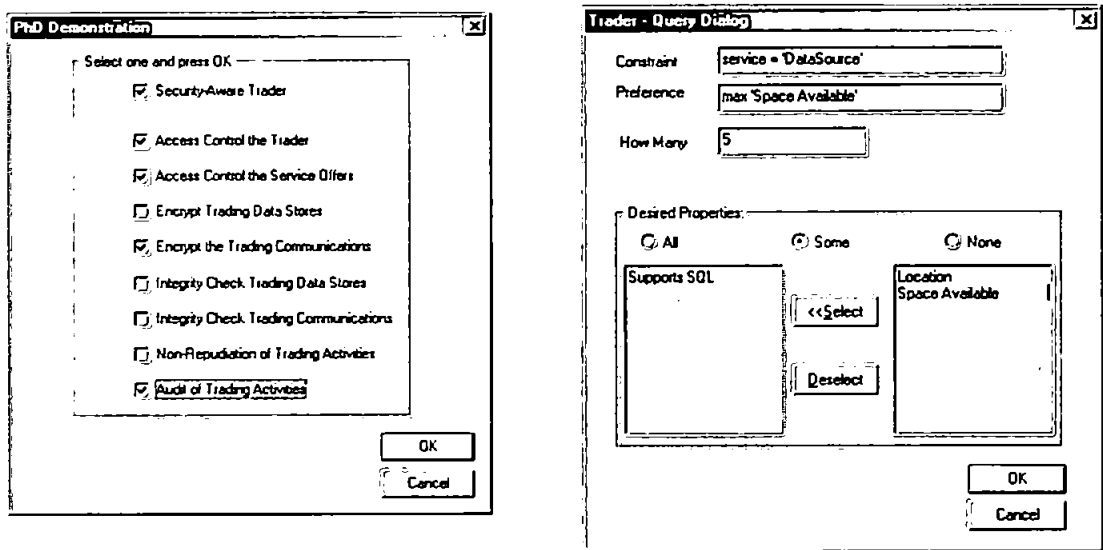


Figure 8-11 Trader Security Administration & Query Screens

8.2.3 Requirements Matrix

To further illustrate the ability of the New Security Framework and the prototype to meet the security requirements, the table below lists all of the DPE security

requirements, and identifies which ones are met by the original TINA security model, the New DPE Security Framework, the original CORBA Security Service and the Prototype (implemented as an extension to CORBA) (see section 4.3 and 7.2.1).

The matrix illustrates how the DPE Security Framework (and the resulting implemented Prototype) is able to provide the necessary requirements for a secure DPE. This has been achieved by several means, such as extending facilities such as auditing and non-repudiation, which were previously not present or incomplete; management has been re-structured and administration interfaces have been added or extended to provide the flexibility required; and interoperability has been introduced to deal with disparate security domains.

	Security Requirement	TINA	New DPE Security	CORBA Security	Prototype (CCS)
1.	Identification and Authentication	✓	✓	✓	✓
2.	Authorization & Access control	✓	✓	✓	✓
3.	Propagation of security attributes	-	✓	✓	✓
4.	Secure communications	-	✓	✓	✓
5.	Secure stored data	-	✓	-	✓
6.	Secure Auditing	-	✓	-	✓
7.	Non-repudiation	-	✓	-	✓
8.	Administrative interfaces	-	✓	✓	✓
9.	Interoperability	-	✓	-	✓
10.	Scalability	✓	✓	✓	✓
11.	Integration with existing environments	-	✓	-	-
12.	System Recovery	-	-	-	-
T1	Intrusion Detection	-	-	-	-
T2	Hardware/software protection (Mech.Mgmt.)	-	✓	-	-
M1	Inter-object communications	✓	✓	✓	✓
M2	TINA services - operational	✓	✓	✓	✓
M3	TINA services - control	✓	✓	✓	✓
M4	TINA services - administration	-	✓	✓	✓
M5	Inter-DPE security	-	✓	✓	✓
A1	Secure participant interaction	-	✓	✓	✓
A2	Application Admin	-	✓	✓	✓
A3	DPE applications security	-	✓	✓	✓
I1	Attribute Mappings	-	✓	-	✓
I2	Operational interoperability	-	✓	-	✓
I3	Control/Administration interoperability	-	✓	-	✓
I4	Application Security Context	-	✓	✓	✓

Table 8 - 2: DPE Security Requirements Matrix

8.3 Verification

In order to verify the prototype, two approaches were used. Firstly, the practical verification by performance modelling was used to analyse the system. Secondly, a more theoretical verification by analysing current standards was also used.

8.3.1 Performance Modelling

When assessing performance in a distributed object system, the cost of object invocation is measured in milliseconds and so the number of invocations should be carefully considered when analysing a system [112]. Therefore when considering the performance of the CCS, it will be measured in object invocation calls. This will be compared with the CORBA Security Service to see if significant overheads have been added. The actual time of the invocation is not measured because it is subject to too many other variables, e.g. platform, network load, bandwidth. Therefore the number of invocations is deemed to be a more realistic measurement.

The modelling will consider three areas, operational level security, administration of security and the security-aware CORBA service.

8.3.1.1 Operational level

For each of the six main security facilities in the security service, i.e. authentication, access control, QoP, audit, non-repudiation and secure invocation, an event sequence chart is presented. It will map the number of calls used and provide a comparison between the new Comprehensive CORBA Sec (CCS) and the current CORBA Security Service by highlighting the new enhanced operations with a broken line. Operations, which previously existed in CORBA, are illustrated by a solid line.

When considering the number of invocations, the chart may illustrate the actual method invocation on a server that initiates the security service, however this invocation (identified as 'invoke' on the chart) will not be included in the object invocation count – the count is restricted to security object invocations. Also in some instances a '*create object*' invocation is counted, because although it is not a specific operation specified in the IDL it is considered an invocation on the constructor of a security object.

Each chart will reference previous examples that have been presented in the thesis descriptions of CORBASec and CCS, which will provide a basic explanation of the objects utilised in that service.

Firstly, the authentication event sequence chart. In this scenario, a principal is logging on to a system and wishes to be authenticated and presented with valid credentials (see section 7.2.4.2 for the CCS example). CORBASec utilises 3 object invocations, while the Comprehensive CORBASec (CCS) makes 8 object invocations.

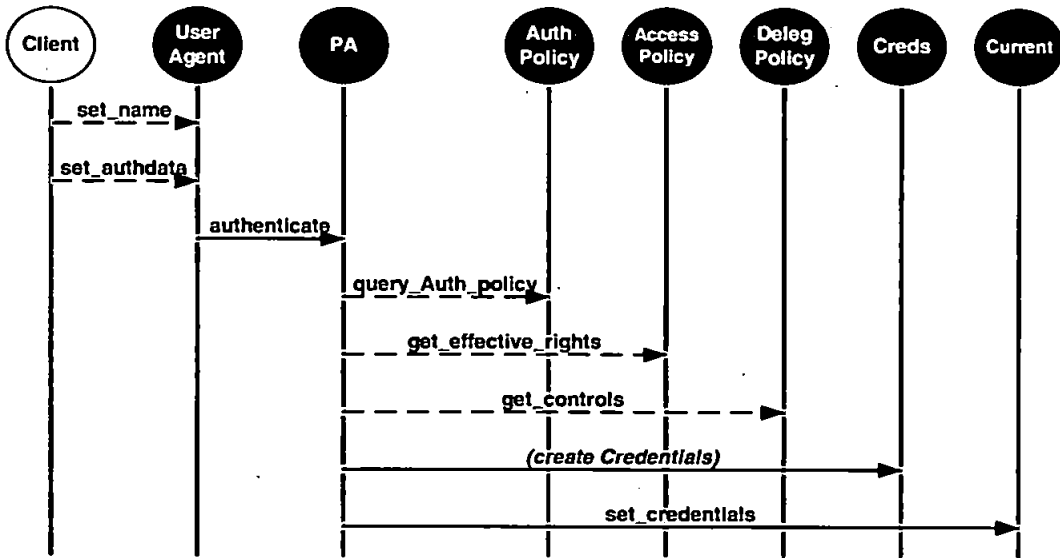


Figure 8-12 Authentication Event Sequence Chart

Access Control is considered in the next event sequence chart. It considers the scenario when a client invokes a server, and the server decides whether the client is authorized to do so (see section 7.2.4fs.2 for CCS example). Here, both CORBA Sec and the CCS utilize 4 object invocations.

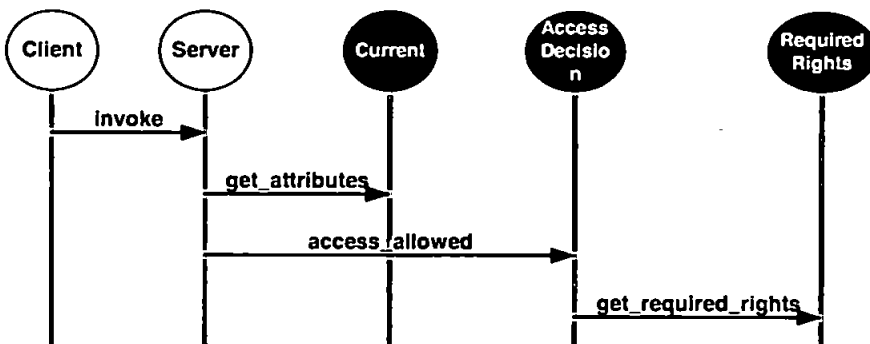


Figure 8-13 Access Control Event Sequence Chart

The next chart looks at QoP in both services. It considers the case of a client changing the QoP it requires when invoking a server (see 7.2.5.2 for CCS example). CORBASec makes 2 object invocation, while CCS makes 3 invocations.

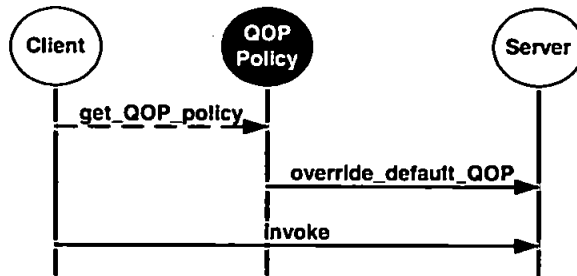


Figure 8-14 QoP Event Sequence Chart

Non-repudiation considers a scenario where the client generates evidence, e.g. proof of origin, which has to be verified by the server. The server then generates evidence to support this verification. A single call to the adjudicator to settle a dispute is also shown above (see section 7.2.6.2). CORBASec utilises only 6 invocations while CCS makes 11.

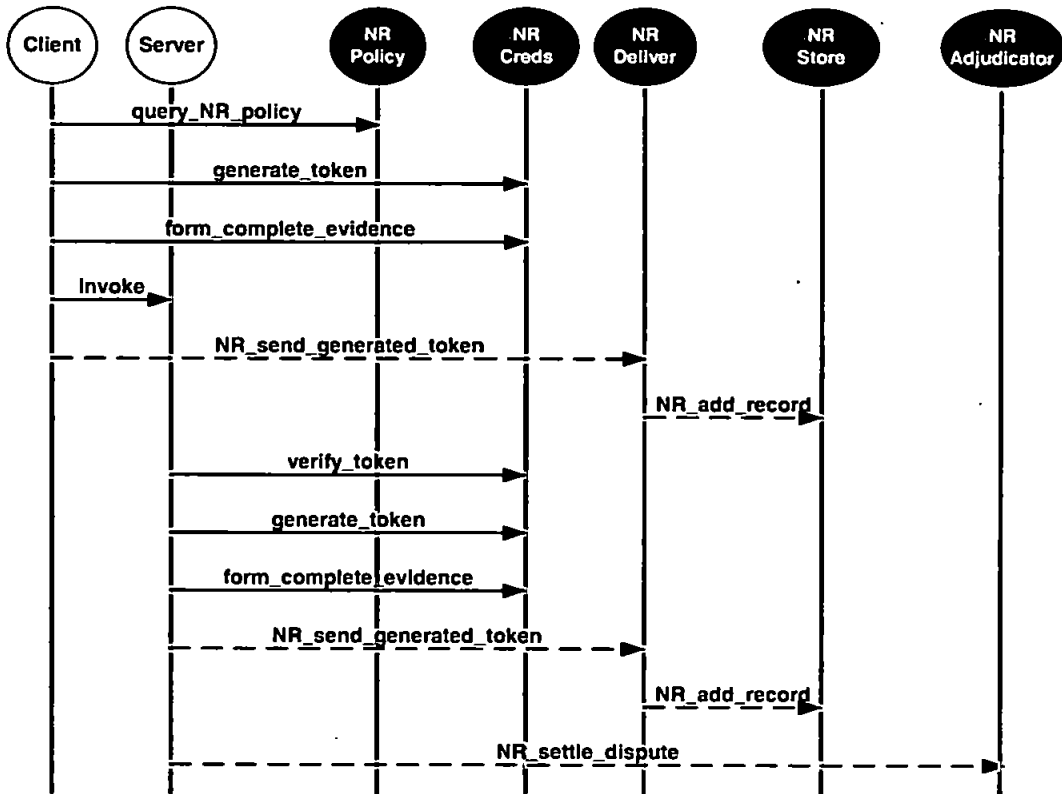


Figure 8-15 Non-repudiation Event Sequence Chart

The Audit scenario involves a client invoking a server method. The server considers whether the event should be audited and what the response should be when the event is to be audited. A record is written to the log to record the event and an alarm is raised to notify the administrator (see section 7.2.6.4 for example). CORBA Sec invokes 3 objects while CCS invokes 9 objects.

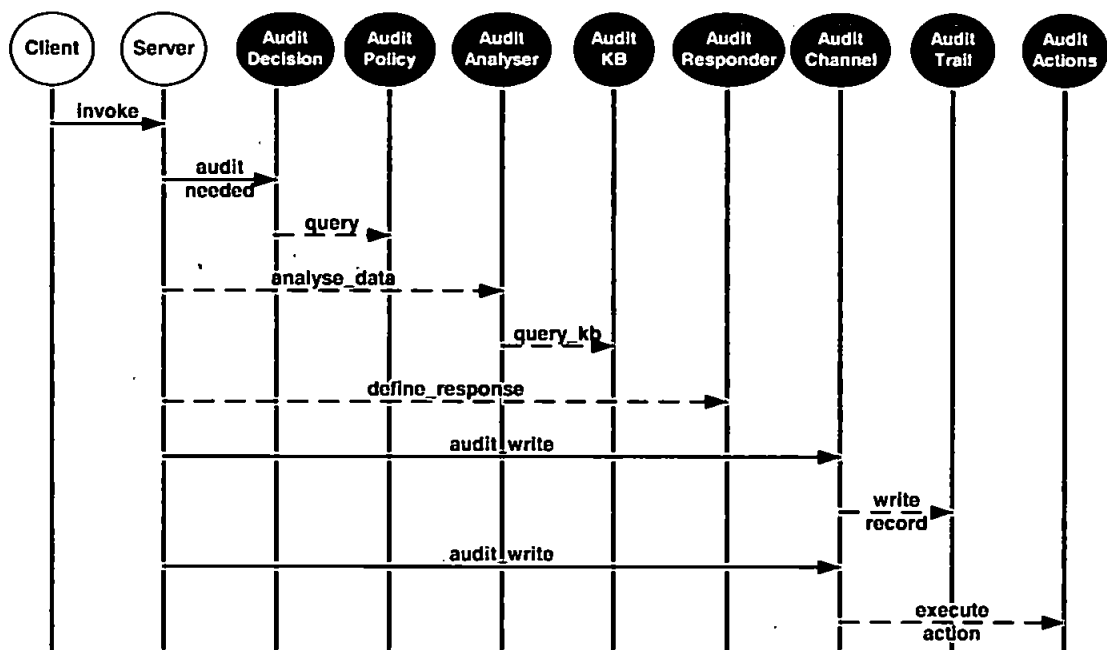


Figure 8-16 Audit Event Sequence Chart

The final chart maps a Secure Invocation between a client and server. The server requires a mapping between policy configurations, i.e. a domain mapping record. The original invocation is then sent to the client, protected by the secure association and a protected reply is returned by the server (see section 5.4.4 for example). CORBA Sec requires 7 invocations and CCS requires 11 invocations.

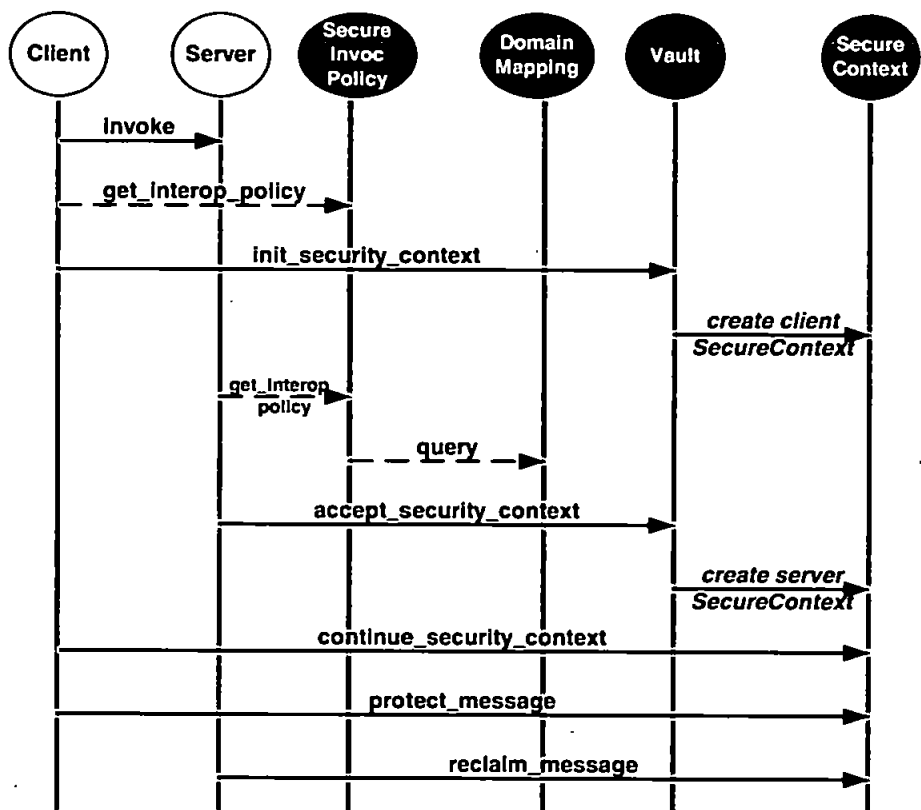


Figure 8-17 Secure Invocation Event Sequence Chart

Now that each of the security operations has been mapped on charts, the resulting number of invocations for each facility can be compared.

As can be seen from both table 8-1 and the figure 8-18 below, the CCS makes more invocations than CORBASec. However, this is to be expected as the CCS offers significantly more facilities than CORBASec. For example, the increase in both non-repudiation and audit in CCS can be accounted for because of the delivery, storage, adjudication and monitoring facilities they now have. Similarly secure invocation now offers policy level mappings and authentication is able to use the *UserAgent* and utilise a comprehensive administration structure to build the credential. There is no

increase in access control, but for QoP the number of invocations is increased by 1 (however this is a 100% increase). Even though there are increases, they average about 3.3 object invocations per facility, i.e. a 90% increase in the number of invocations required.

Service	CORBASec	CCS	Difference	% Diff.
Authentication	3	8	5	166%
Access Control	3	3	0	0%
QoP	2	3	1	50%
Non-Repudiation	6	11	5	83%
Audit	3	9	6	200%
Secure Invocation	7	10	3	42%
			Ave.=3.3	Ave. 90%

Table 8-1 Operational Object Invocation Comparison

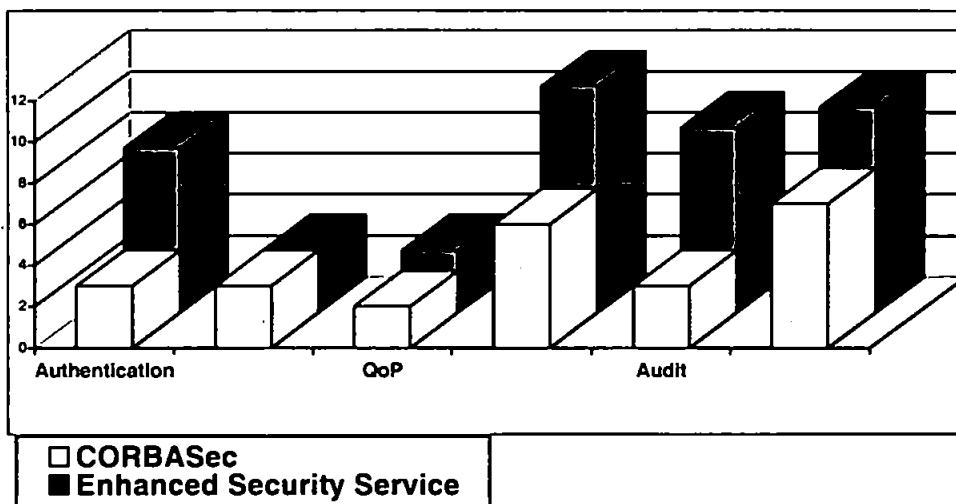


Figure 8-18 Operational Object Invocation Comparison

Therefore this overhead in object invocation is seen as minimal and bearable by the system, when one considers the new and enhanced facilities that are now available in

CCS. Also these examples are assuming that all services are used all the time, but in any large distributed object system, the administrator would tailor the policies to provide the maximum protection while minimizing the overhead.

8.3.1.2 Administration Level

Some of the administration objects, and how they are used at an operational level, have already been considered in the previous section. However, this is only a small number of the possible administrative methods available. This section will look at all of the administration objects and their methods. It will not employ sequence charts, but will simply compare object method numbers in both CCS and CORBASec.

Table 8-2 below, lists the administration objects and the number of methods available on each in CCS and CORBASec.

Service	CORBASec objects	CCS objects
Authentication	-	AuthPolicy, AuthMech
Access Control	AccessPolicy DomainAccessPolicy RequiredRights	AccessPolicy, AccessMech
Delegation	DelegationPolicy	DelegationPolicy, DelegationMech
QoP	-	QOPPolicy, QOPMech
Non-repudiation	NRPolicy	NRPolicy, NRMech
Audit	AuditPolicy	AuditPolicy, AuditMech
Secure Invocation	SecureInvocationPolicy	SecureInvocationPolicy, DomainMapping

Table 8-2 Administration Objects

This highlights the fact that CCS provides a comprehensive administration structure and so will provide more methods. However it should be noted that CCS provides policy and mechanism level administration, while CORBASec only dealt with policy

level administration. This fact is considered in table 8-3 below which compares the number of object methods.

	CORBASec	CCS		
Service	Policy	Policy	Mechanism	Total
Authentication	-	6	5	11
Access Control	6	7	5	12
Delegation	2	10	-	10
QoP	-	11	9	20
Non-Repudiation	2	6	13	19
Audit	5	6	9	15
Secure Invocation	2	15	-	15

Table 8-3 Comparison of the Numbers of Administration Object Models

As shown in figure 8-19 below, if one was to compare the total number of methods available in administrative CCS objects with the number in CORBASec, then there would be a significant overhead, approximately 12.1 object invocations per facility. However, if the comparison is made based on comparing policy administration, then the overhead becomes only 6.3 object invocations.

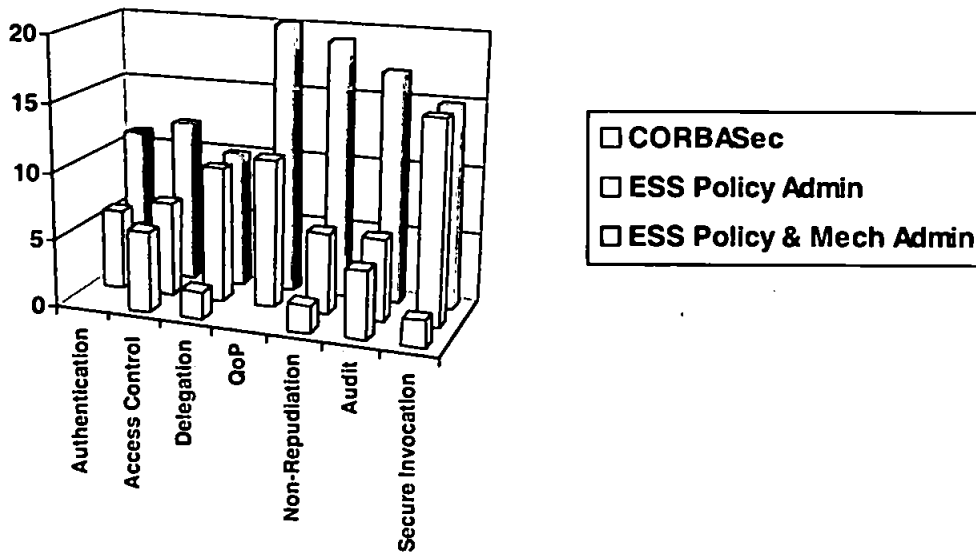


Figure 8-19 Number of Administration Object Methods

This overall increase, although significantly less than the original comparison (almost 50%), still reflects the additional facilities provided by the new administration structure. It provides a new flexibility and enables mechanism and service independence. Therefore it is again an issue of weighing up the tradeoffs between performance and security. Although the increase at administration level is double the operation level (3.1 object invocations), it can be seen as a tolerable overhead because these administration methods are generally only required at system set-up and for maintenance purposes.

8.3.1.3 Security-Aware CORBA services

The Security-Aware CORBA service that was designed by this research was the Trader. By using the figures already calculated for security invocations at an operational level, the impact by security on the Trader service can be studied.

Security Attributes	Attribute Invocations	Security Invocations	Total
Security-aware	1	-	1
Access_control_trader	1	3	9
Access_control_service_offers	1	-	1
Encrypt_stores, Integrity_check_stores	2	2	4
Encrypt_comms, Integrity_check_comms	2	10	12
NR_trade	1	11	12
Audit_trade	1	9	10

Table 8-4 Security-Aware Trader's Security Object Invocations

Table 8-4 above, lists the security attributes defined for the Trader (see section 6.5). It details the number of invocations required to get the attribute value and then defines the number of operational level object invocations required to execute the appropriate security facility. However, when calculating the average increase in the number of object invocations for each attribute that is set on, two assumptions are made:

- Although two attributes are tested, *Encrypt_stores* and *Integrity_check_stores*, the number of object invocations is kept to a single execution of a QoP facility. This is because the implementation will be executed as one function.
- Similarly for *Encrypt_comms* and *Integrity_check_comms* the number of object invocations is kept to a single execution of a Secure Invocation facility, as the implementation will execute both as a single function.

The average number of object invocations for each attribute set is then calculated as 4.9 invocations. Again trade-off is an issue, this Security-aware Trader can provide security that previously did not exist within the system and so this overhead has to be

weighed against the added protection. Also the Trader will be tailored to suit the system, with only the appropriate attributes set, so that performance and security will be considered when deciding just how 'security-aware' the Trader should be.

8.3.2 Standards Verification

Having considered verification from the practical perspective, i.e. performance modelling, it is relevant to now consider how the research relates on a more theoretical level, i.e. the DPE implementation standards. Standards in a research area do not remain static, they are constantly being revised and updated. The technologies used can become outdate and replaced by new ones. This section will look at how the standards and technologies used in the research, have been revised, and explain how the new framework is still valid even with the recent changes.

8.3.2.1 ISE and DPE standards

TINA has influenced groups such as the OMG, and it was not a surprise in September 2000, when the OMG announced that after TINA-C's decision to discontinue operation, the OMG would continue TINA-C's work under their Telecommunications Domain Task Force [113]. Therefore TINA will still remain the central ISE DPE standard even though it has now transferred to the OMG. This obviously also strengthens CORBA's position as an important standards-based DPE solution. Another such recognition is the fact that CORBA is now recognised as an international interoperability standard [114]. The International Standard's Organisation (ISO) recently adopted CORBA's Interoperability platform as ISO/IEC 19500-2. The ISO have already adopted several other specifications such as IDL (ISO/IEC 14750|ITU-T Rec. X.920), Trader (ISO/IEC 13235|ITU-T Rec. X.950) and

ODP Type Repository (ISO/IEC DIS 14769|ITU-T Rec. X.960). The OMG has also submitted CORBA's ORB specification for adoption.

8.3.2.2 CORBA Security Service Revision 1.5 and 1.7

The OMG follows a Technology Adoption Process [115], which will be outlined before discussing the version issues of CORBASec. Initially a Task Force may issues a Request for Information, which will eventually result in a Request for a Proposal (RFP). Submitters can then reply to the RFP by a submission deadline with an Initial Submission, which can later be updated as a Revised Submission. Once the OMG Architecture Board (AB) has certified a submission, i.e. that it is compliant with CORBA technology; the Task Force can then recommend to the Board of Directors (BOD) that the submission become an Adopted Specification. A Revision Task Force (RTF) can then carry out revisions on the Adopted Specification. The RTF only exists for a specified length of time and is responsible for maintenance of an adopted OMG specification, i.e. they clarify ambiguities and correct errors; they cannot extend a specification with new functionality. Once certified by the AB and implemented by one submitter, a BOD can vote to make the technology a formal Available Specification.

When the research began, the CORBA 2 security service was actually at revision 1.2. Since then, revision 1.5 was made the formal specification by the OMG in June 2000 [116] and, at the time of writing, revision 1.7 [117] is now being adopted by OMG vote but it has not been accepted as a formal available specification. Revision 1.8 is just at the RTF stage [118]. This section will examine the changes in revision 1.5 and 1.7 (it is too early to evaluate 1.8) and see whether the research is still valid with these later versions.

Two major changes occurred in version 1.5 as described below (other minor data type changes occurred, but they are of no consequence to the research).

1. New Administrative Objects: Five new policy objects are introduced:

- **MechanismPolicy:** used to request the use of one specific set of mechanisms when invoking a particular object reference;
- **EstablishTrustPolicy:** used to specify a particular policy between a client and the target object;
- **QOPPolicy:** used to specify a particular Quality of Protect for messages sent to a particular object reference;
- **DelegationDirectivePolicy:** used to specify the delegation policy used for invocations on the target object;
- **CredentialInvocationPolicy:** used to specify a particular set of Credentials to be used when invoking a target object.

There had been some previous confusion with regard to how a client would override default policies details – they were generally retrieved from and set in objects such as **Current** and **Credentials**. Therefore to alleviate confusion and provide a clear methodology for clients to specify this information when attempting to invoke a server, the above policy objects were specified. They all have a very simple structure – a single readonly attribute specifying the detail policy value, e.g. the **EstablishTrustPolicy** contains a single attribute structure that can be set to specify if trust is to be established in the client, the target or both.

2. **SSL and SECIOP:** The SSLIOP specification was previously a separate document. It is now introduced as part of the specification.

A worked example of security, using revision 1.5, is provided in a paper by Chizmadia [119].

In revision 1.7 only one major change has occurred.

1. **New security object SecurityManager.** SecurityManager does not introduce any new functionality, it merely takes the security functionality that previously existed in the Current object and places it in a separate security object.

These changes are superficial. They do not extend or add new functionality – they merely clarify procedures or move functionality to new objects. Within the specifications, the list of main objects remains unchanged, and so it none of these new objects have a substantial impact on the security service provided. Therefore all the issues and problems identified in CORBA Sec remain unchanged, and the new recommendations from the research still apply even to these later versions of the service.

8.3.2.3 CORBA 3.0

In December 1999, the OMG voted to adopt the complete CORBA 3.0 specification. CORBA 3 is actually a suite of specifications, which when taken together, adds a new dimension of capability and ease-of-use to CORBA [120, 121]. Although much discussed, the CORBA 3 is not yet adopted as the formal available specification, i.e.

version 2.4.1 is still the latest official version for vendors to reference. The new specification can be divided into three categories, which are described below:

1. Internet Integration:

- a. New *Java-to-IDL Mapping* specifications will allow developers to build distributed applications completely in Java and then generates the CORBA IDL from the Java class files [122]. CORBA 2 already provided a IDL-to-Java mapping.
- b. The *CORBA 3.0 Firewall specification* defines interfaces for passing IIOP through a firewall. It includes options for allowing the firewall to perform filtering and proxying on either side [123].
- c. The *Interoperable Name Service* [124] defines a URL-format object reference that can be typed into a program to reach defined services at a remote location, including the Naming Service.

2. Quality of Service Control

- a. The specification identifies a minimum compliance supporting CORBA ORBs. This *Minimum CORBA* specification is designed to jumpstart the use of CORBA embedded devices [125].
- b. *Real-time CORBA* extends the CORBA specification for a new type of ORB called the Real-time ORB [126].
- c. *Fault-tolerance* for CORBA is also addressed, and defines a standard based on entity redundancy and fault management control [127].
- d. The *Asynchronous Messaging* specification has two components: levels of quality of service (QoS) agreements and Interface Definition

Language changes necessary to support asynchronous invocations [128].

3. The CORBA Component Model (CCM)

The CCM will specify a framework for the development of 'plug-and-play' CORBA objects. It encapsulates the creation, lifecycle, and events for a single object and allows clients to dynamically explore an object's capabilities, methods, and events. The specification has three major parts, which cover, a container environment to provide services, integration with Enterprise JavaBeans [129] and a software distribution format that enables a CORBA component software marketplace [130].

Only one change specifically references the security aspect of CORBA, i.e. the firewall specification. However, this is addressing a specific issue and not the overall security limitations within the Security Service and CORBAServices. Therefore the research is still valid and can still be applied in a CORBA 3.0 environment. Component technology may be the driving force for CORBA 3.0, but components still require Security, and CORBAMSec still requires the new objects and functionality defined by this research.

The OMG has realised limitations of CORBAMSec. It has provided some suggested future features within the specification, but has given no detail as to how they might be accomplished. One example is the notion of an attribute mapper. It is identified in the CORBAMSec specification [36]. There is also no indication of when any further amendments would be introduced. The OMG recently drafted a roadmap [131], which

lists several areas and features, but there is no timeline for their introduction or whether they will even be completed, e.g. under policy management the OMG SECurity Special Interest Group (SECsig) has identified that negotiation of federated domains is required, while under interoperability they have listed interoperability across products. Many of the features listed in this new wish-list have already been addressed by this research, which identified the problems some time before they were acknowledged by the OMG.

8.3.2.4 Issues when implementing the standards

The previous sections have provided verification of the work by evaluation of a performance model of the research and also by viewing the research against the latest standards and technologies changes. This section looks at real-world problems that have been encountered by vendors using the current version of the DPE security

The first example revolves around the current implementations of CORBASec products that are currently available. These problems were covered in section 7.2.1. Vendors such as Concept5, Dascom and Entegrity, all have a common set of problems that they address with proprietary solutions. Firstly, they have a common set of features they all need to extend past the CORBASec specification because it is too restrictive:

- Extending the administration features through defining new interfaces;
- Using additional features to integrate with existing technologies, i.e. unitary logon, bridge technology;
- Extending the audit facilities to help secure audit records or make them available to monitoring tools.

However, there are still a number of restrictions:

- Replaceability is difficult and so they are all limited to specific sets of security technologies/mechanisms;
- Data storage is proprietary, e.g. use of LDAP;
- There is no proper monitoring/IDS integration available;
- Non-repudiation is not available;
- Interoperability is still limited to compatible domains and technologies (although most have consulting divisions that provide customised solutions);
- Multi-vendor interoperability is also not available.

All of the above issues are addressed by the research. The separation of mechanism and policy administration provides for mechanism independence and therefore releases implementers from the constraints of using the same mechanisms and technologies; it also obsoletes the notion of replaceability, because all objects should be automatically replaceable because they have been abstracted from mechanism dependencies. The new security service addresses issues such as securing stored data and extends it to persistent storage by introducing the concept of security aware services (e.g. a security aware persistent storage service could operate in conjunction with the security service). Other facilities such as non-repudiation and audit have been extended.

While the above examples illustrate the issues that CORBASec implementers encounter, the problem of inadequate DPE security is experienced in other areas.

Take, for example, Ericsson's Research and Development team who have a new product called FraudOffice [132], for Fraud Detection & Management in a telecommunications network. The FraudOffice product family offers a complete end-to-end package to network operators and service providers to combat fraud within telecommunications networks. Ericsson provides the relevant software applications, hardware platforms, systems support, fraud management services and fraud competence training so that the telecommunications operator organisation is correctly equipped to minimise the substantial financial losses and inconvenience caused by fraudsters in a network, which can be up to as much as 5% of annual billed revenue.

One of FraudOffice's main selling points is its integration flexibility [133], which is achieved through an open scalable CORBA-based platform. Because of this integration ability, FraudOffice can maximise the potential use of the operators' existing support systems (e.g. billing, data warehouse, MIS system, SS7 monitoring, Customer Care). This will become an important consideration in the next generation where the number of potential data sources for fraud detection is likely to increase dramatically (e.g. credit card transactions, log examinations, balance reconciliation, IP transactions, customer service applications, payment history, etc.) and become more diverse.

However, Ericsson encountered some obstacles when developing this system. The main problem was related to the fact that the CORBA security service was not flexible or extensive enough for their requirements. As a result, they had to build their own Security Manager and Audit and Alarm facilities [134], because they were unable to use the CORBA security technology. If the Comprehensive CORBASec had

been available, it would have saved the developers time and also provide for interoperability with other products using the same standards.

8.4 Summary

The purpose of the implementation was to prove that the new Comprehensive CORBASec, the new Secure Interoperability Service and the Security-Aware Trader are not just theoretical ideals, and that they are implementable. This chapter described the implementation of the research and how it was achieved. The research implemented all objects necessary within each of these services. It is important to note that the method of implementation is not really important to proving the concept, because the key issue is that the IDL-defined interfaces are workable. The implementation proves this even when they are implemented in the rather limited environment. Therefore details of the actual C++ implementation on a Microsoft NT are not necessary, as the same objectives should be achievable using Java on a Sun Workstation; after all this was the driving force behind CORBA distributed systems - implementation and platform independence.

The service implementations have also been applied within an application scenario, in order to illustrate clearly how the services would function together. Therefore the implementation has accomplished its objective by realising the services and using them within an operational environment. Facilities that were missing and, therefore, had to be built in-house are proposed as part of the CCS within the CORBA environment, thereby saving time and reducing the risk of error.

The chapter also evaluated the research system performance. Benchmark testing against current CORBA security products is not appropriate in the case of the research because the products could not act as an equivalent comparison to the research implementation described. However, object invocation is the advised method of performance modelling for object distributed systems, and was therefore used in the research model.

Although there is an overhead for any security operation, it was kept to a minimum and is, therefore, considered an acceptable trade-off against the extra security that is provided. As in all systems, it is the job of the administrator to tailor policies and system options to find the optimal solution, where performance and security can co-exists harmoniously. The CCS provides a comprehensive and flexible administration system to provide the administrator with this ability.

Verification of the security issues in the CORBA DPE serves to verify the methods used as generic DPE solutions. The principles used to provide new security features to the security services, secure other DPE services and enable secure interoperability between disparate domains have been proven effective and implementable in a practical DPE environment.

9. Conclusions

9.1 *Achievements of the Research*

This chapter presents concluding thoughts on the research. The following summarises the achievements of the research, which have met all of the objectives, defined in chapter 1.

1. A comprehensive analysis of the general requirements for distributed system security was conducted exceeding any previous investigation of this topic. As a result, a new set of DPE security requirements was defined and a new definition of the DPE security domain was provided. When evaluated against these requirements, the current DPE security model was shown to be inadequate on all levels.
2. A new security framework for DPEs was defined. The service components were defined at both an operational and management level. They provide all of the necessary security functions: authentication, access control, integrity, confidentiality and non-repudiation.
3. A new secure interoperability framework for DPEs was defined. A distributed system, which operates across disparate security domains, can use the new interoperability protocol to facilitate secure DPE inter-domain interactions.

4. The new and novel concept of security-aware DPE services was introduced and found to be necessary to address security vulnerabilities within the DPE services themselves. The Trader was selected as the proof-of-concept, and a new security-aware Trader architecture was defined.
5. The theoretical DPE security framework was implemented and verified by providing a mapping of it to an implementable DPE specification, the OMG's CORBA, and then building a working proof-of-concept. The work was further verified by providing object invocation analysis of the services and also through analysis of standards and existing real-world issues.

It is therefore, considered that the research has made a substantial contribution to knowledge within the domain of DPE security.

9.2 Limitations of the Research

The following sub-sections present the author's thoughts regarding the limitations of the research.

1. Although the new audit objects were defined, they were not implemented. This was decided as the audit implementation was a substantial undertaking and would not have been achievable within the research timeframe. Other areas were considered to be of greater significance to the proof of concept.

2. The issue of recovery, i.e. the system returning to a secure state, was not considered within the research. It would take several years of research to fully address this issue and, therefore, it was decided that it could not be addressed within the scope of this work.
3. Although it was never the intention of the work to examine the security-awareness of all DPE services, it was observed within the thesis that other services, such as persistence, could provide generic solutions to security problems if they were also security-aware. How this could be achieved, was not addressed within the research.

9.3 Suggestions for Future Work

There are six key areas where continuation of the research should be focused. This work was considered outside the scope of this research or was considered too complex to complete within the research timeframe.

1. Within the Audit service, there are several areas that could be further investigated in order to complete the service definition. This would include the specification of an Audit Record Format to enhance interoperability between audit services working in different security domains. A common audit record format would allow separate, and independently implemented, audit systems to easily exchange information. Similarly the specification of the Audit analysis token that is returned by the DPE AuditAnalyser needs to be defined. The definition of more Audit Event and Selector types could help provide a more flexible and configurable audit policy. Events

and Selectors could be defined within specific vertical domains such as financial or healthcare. The IDS domain is currently in the process of being standardised by the Common Intrusion Detection Framework (CIDF) [135] and the IETF [136]. This is expected to provide greater interoperability among different analysis and response systems. However, these standardisations are not complete. Therefore research in the area would significantly help the security of DPEs.

2. A standard non-repudiation token, that could be written to the NRStore object and would facilitate interoperability, needs to be defined.
3. There are many DPE supporting services, such as persistence, events and time. This research has only considered two of those services, Trader and Security, and how they can be enhanced to provide a more secure DPE. The other services also need to be studied in order to define their security vulnerabilities and solutions to these problems. In doing so it would provide a complete analysis of security for DPE services.
4. It was proposed in the research that DPE services, such as the Persistence and Query Services, could be made security-aware and then used to provide secure generic solutions to problems such as secure persistent data storage and retrieval – as would be required by the non-repudiation service's NRStore and many other objects. Another scenario could involve the use of secure Event and Transaction services in order to provide secure

recovery within a DPE. Although this would provide the ultimate generic solution for a DPE, it is recognised that there are problems. With regard to persistence, the definition has been recognised as not implementable in its current state [137]. As some DPE services are, to-date, not available in detailed specifications, work is needed to ensure that when complete, they will be able to inter-work to provide secure generic solutions to several problems.

9.4 Summary of Research Conclusions

According to the TINA consortium [138], the future of communications depends not only on individual technical or standards-based solutions but also on one universal generic software architecture solution. It also states that this approach has to be global, and it needs to involve all areas of the industry; the ultimate aim is to produce a complete set of specifications for building and managing services of any degree of complexity. However, with the rise in security breaches found by recent surveys such as that from the FBI/CSI [46], it is important to ensure security in this new open, global environment, as such an environment will only provide more opportunities to compromise a system. The current DPE security solution has been proven to be inadequate and this research has addressed the problem.

The research achieved its objectives by assessing security in a DPE, defining the current limitations and then proposing solutions to overcome these limitations. A new security framework was defined, which provides a complete set of security facilities and a comprehensive management structure. A secure interoperability service was defined which facilitated mechanism and policy level negotiations, and a security-

aware Trader was designed to prove the concept that security-aware DPE services offer a greater level of security within a DPE.

This work provides a standardised solution to increase security. DPE security vendors currently experience problems of interoperability between their products and also have to create proprietary extensions to overcome other limitations of the security service. These problems will be dissolved if vendors adopt the new security solution proposed in this research. It will provide users with greater options and, therefore, allow them to create a more secure distributed environment.

Further work can be pursued to ensure improved interoperability of the enhanced security facilities, i.e. audit and non-repudiation token definition. Also improved security for other DPE services can be achieved through the study of their security limitations and the application of security-aware interfaces.

ISE is no longer limited to just the telecommunications arena, it is supported by the data communications and processing industries. E-commerce is readily adopting the technology because of its ability to quickly provide new services and facilities in a heterogeneous, distributed environment. All of this research work will provide a more secure distributed processing environment in which a multitude of applications can be built. Whether it is finance or healthcare, education or just surfing the information highways, the data will be available, but it will be protected.

10. References

- [1] **D.I. Hopper**
"Destructive ILOVEYOU virus strikes worldwide"
CNN, 4 May, 2000, <http://www.cnn.com>

- [2] **M. Masterson**
"Love bug costs billions"
CNN Financial News, May 5, 2000, <http://www.cnnfn.com>

- [3] **D. Duffy**
"CyberInsurance: Prepare for the Worst"
Darwin Magazine, December 2000
http://www.darwinmag.com/read/120100/worst_content.html

- [4] **D. Kahn**
"The Codebreakers: The Story of Secret Writing"
Macmillan Publishing Co., 1967.

- [5] **S. Foo, P. Chor Leong, S. Cheung Hui, S. Liu**
"Security considerations in the delivery of Web-based applications – a case study"
Information Management & Computer Security, Vol 7, 1999, pg. 40-49.

- [6] **R. Orfali, D. Harkey, J. Edwards**
"The Essential Distributed Objects Survival Guide"
John Wiley & Sons, Inc., 1996.

- [7] **P.F. Linington**
"Introduction to the Open Distributed Processing Basic Reference Model"
Open Distributed Processing, Elsevier Science Publishers, 1992.

- [8] **ISO**
"Basic Reference Model of Open Distributed Processing"
ISO/IEC JTC1/ SC21/WG7 N838.

- [9] **N. Natarajan**
"Principles of a software architecture for information networks"
Bell Communications Research

- [10] **J.J. van Griethuysen**
"Enterprise modelling, a necessary basis for modern information systems"
Open Distributed Processing, Elsevier Science Publishers, IFIP, 1992.

- [11] **G.F. Coulouris, J. Dollimore, T. Kindberg**
"Distributed Systems Concepts and Design"
Addison-Wesley Publishing Company, 2nd Edition, 1994.
- [12] **J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen**
"Object-Oriented Modelling and Design"
Prentice Hall International Editions, 1991.
- [13] **W.J. Barr, T. Boyd, Y. Inoue**
"The TINA Initiative"
IEEE Communications Magazine, March 1993.
- [14] **TINA-C**
"Business Model and Reference Points for TINA"
TINA-C, 2000, <http://www.tinac.com>
- [15] **TINA-C**
"Overall Concepts and Principles of TINA"
TINA-C, Deliverable TB-MDC.018_1.0_94, Version 1.0, 17 Feb 1995.
- [16] **TINA-C**
"Service Architecture"
TINA-C, Version 5, 31 March 1997.
- [17] **R.H. Glitho, S. Hayes**
"Telecommunications Management Network: Vision Vs. Reality"
IEEE Communications Magazine, p47-52, March 1995.
- [18] **M. Knackelmans, E. de Jong**
"Overview of IN and TMN Harmonisation"
IEEE Communications Magazine, p62-66, March 1995
- [19] **TINA-C**
"Principles of TINA"
TINA-C website, 2000, <http://www.tinac.com>
- [20] **TINA**
"Domain Types and Basic Reference Points in TINA"
TINA-C, May 1995, <http://www.tinac.com>
- [21] **N.D.Hoa**
"Distributed Object Computing with TINA and CORBA"
N.D.Hoa, June 1997, <http://www.nenya.ms.mff.cuni.cz/~hoa/tina/tina.html>

- [22] **TINA**
"TINA Object Definition Language (TINA-ODL) Manual"
TINA, Version 1.3, 20 June 1994, <http://www.tinac.com>
- [23] **ITU**
"ITU X.700 Series – System Management"
ITU, <http://www.itu.int>
- [24] **R. Orfali, D. Harkey, J. Edwards**
"Instant CORBA"
J. Wiley & Sons, 1997.
- [25] **OMG**
"C++ Language Mapping Specification"
OMG Document 99-07-36, July, 1999.
- [26] **OMG**
"IDL to Java Language Mapping Specification"
OMG Document 99-07-53, July, 1999.
- [27] **J. Siegel**
"CORBA Fundamentals and Programming"
John Wiley & Sons, Inc., 1996.
- [28] **D. Rodgers**
"Developing Secure, Web-Based Applications"
Software Development Journal, May 1998,
<http://www.sdmagazine.com/supplement/ss/feature/s985f2c.shtm>
- [29] **The Australian**
"Mobile fraud runs riot"
The Australian, 22 September, 1998.
- [30] **E. Leahy**
"Ericsson Fraud Management Solution – FraudOffice"
Ericsson, Business Evolution and Components Seminar, 12 March, 1999.
- [31] **ISO**
"ODP Trading Function"
ISO/IEC JTC1/SC21, 20 June 1995.
- [32] **IONA Technologies PLC**
"OrbixTrader – White Paper"
IONA web site, (<http://www.iona.com>)

- [33] **OMG**
"Trading Object Service"
OMG Document 96-05-06, May 1996.
- [34] **R. Resnick**
"Intergalactic Distributed Objects"
Dr.Dobb's SourceBook, January/February 1997.
- [35] **M. Bearman**
"Tutorial on ODP Trading Function"
DSTC, University of Canberra, Australia, <http://www.dstc.edu.au>
- [36] **OMG**
"CORBA Security Service Version 1.2"
OMG, June 2000, <http://www.omg.org>
- [37] **OMG**
"Common Secure Interoperability"
OMG Document, orbos/96-06-20, June 1996.
- [38] **Netscape**
"SSL3.0 Specification"
<http://home.netscape.com/eng/ssl3/3-SPEC.HTM#2>
- [39] **NUA**
"Viruses, Hackers to Cost Businesses USD1.6 bn"
NUA Internet Survey, September 1999, <http://www.nua.ie/surveys/>
- [40] **CERT/CC**
"CERT/CC Statistics 1988-2000"
CERT, December 2000, http://www.cert.org/stats/cert_stats.html
- [41] **ISO**
"Information Processing Systems OSI RM, Part 2: Security Architecture"
ISO/TC 97 7498-2, 1988.
- [42] **ISO**
"Information Processing Systems OSI RM"
ISO/TC 97 7498, 1998.
- [43] **ISO CD 10181-3**
"Access Control Framework"
ISO, Geneva, Switzerland, 1991.

- [44] **F.M. Avolio**
"Network Security Building Internetwork Firewalls"
Business Communications Review, January 1994.
- [45] **R Kay**
"Top Security Threats"
BYTE, April 1995.
- [46] **CSI/FBI**
"Issues & Trends: 2000 CSI/FBI Computer Crime and Security Survey"
CSI/FBI, 2000, <http://www.gosci.org>
- [47] **D. Gollmann**
"Computer Security"
Wiley & Sons, 1999.
- [48] **D. Bell, L. La Padula**
"Mitre Technical Report 2547 (Secure Computer System): Volume II"
Journal of Computer Security, 1996.
- [49] **E.I. Organick**
"The Multics System: An Examination of Its Structure"
MIT Press, Cambridge, MA, 1972.
- [50] **C. Laferriere, R. Charland**
"Authentication and Authorization Techniques in Distributed Systems"
IEEE Journal, 1993.
- [51] **ISO DIS 10181-2**
"Authentication Framework"
ISO, Geneva, Switzerland, 1991.
- [52] **National Institute of Standards and Technology**
"Digital Signature Standard"
NIST FIPS PUB 186, U.S. Department of Commerce, Feb 1994.
- [53] **M. Ganley**
"Digital Signatures and their uses"
Computers & Security, 13, 1994.
- [54] **J.R. Vacca**
"Stop Impersonators from Entering Your Network"
Internet Security Advisor, pg. 8-18, February 2000.

- [55] **A.P. Conn, J.H. Parodi, M. Taylor**
"The Place of Biometrics in a User Authentication Taxonomy"
Digital Equipment Corporation, June 1990.
- [56] **R. Bright**
"Smart Cards: Principles, Practice, Applications"
Ellis Horwood, 1988.
- [57] **D. Bicknell**
"The key question"
Computer Weekly, 13 February, 1997.
- [58] **P. Wayner**
"Who goes there?"
BYTE, volume 22, no. 6, June 1997.
- [59] **CCITT**
Recommendation X.509 "The Directory-Authentication Framework"
Consultation Committee, International Telephone and Telegraph, International Telecommunications Union, Geneva, 1989.
- [60] **National Bureau of Standards**
"Data Encryption Standard"
NBS FIPS PUB 46-1, U.S. Department of Commerce, Jan 1988.
- [61] **R.L. Rivest, A. Shamir, L.M. AdleMan**
"A Method for Obtaining Digital Signatures and Public-Key Cryptosystems"
Communications of the ACM, v. 21, n. 2 Feb 1978.
- [62] **R.L. Rivest, A. Shamir, L.M. AdleMan**
"On Digital Signatures and Public Key Cryptosystems"
MIT Laboratory for Computer Science, Technical Report, MIT/LCS/TR-212, January 1979.
- [63] **Electronic Privacy Information Centre**
"Cryptography and Liberty 1999: An International Survey of Encryption Policy"
EPIC, Washington DC, <http://wwwz.epic.org/reports/crypto1999.html>, 1999.
- [64] **Bureau of Export Administration**
"Revisions to Encryption Items; Interim Final Rule"
Department of Commerce, 15 CFR parts 734, 740, et al. , January 14, 2000.

- [65] **D.D. Chamberlin, J.N. Gray, I.L. Traiger**
"Views, Authorisation and locking in Relational DataBase Systems"
Proceedings from the AFIPS NCC, USA, Vol. 44, 1975.
- [66] **F. Cohen**
"Eliminating Common Software Security Faults"
Software Development, May 1998 (<http://www.sdmagazine.com>).
- [67] **Department of Defense**
"Department of Defense Trusted Computer System Evaluation and Criteria" –The Orange Book;
DOD 5200.28-STD, Dec 1985.
- [68] **S. Muftic**
"Security Mechanisms for Computer Networks"
Ellis Horwood, Ltd., Chichester, England, Dec 1988.
- [69] **National Institute of Standards and Technology**
"Secure Hash Standard"
NIST FIPS PUB 180, U.S. Department of Commerce, May 1993.
- [70] **R.L. Rivest**
"The MD5 Message Digest Algorithm"
RFC 1321, April 1992.
- [71] **G. Tsudik**
"Message Authentication with One-Way Hash Functions"
ACM Computer Communications Review, Volume 22, Number 4, pp. 29-38, 1992.
- [72] **S. Herda**
"Non-repudiation: Constituting evidence and proof in digital cooperation"
Computer Standards & Interfaces, Volume 17, 1995.
- [73] **ISO**
"Non-Repudiation Framework"
SC21 N6167, July 1991.
- [74] **D. Anderson, T. Frivold, A. Valdes**
"Next-generation Intrusion Detection System – A Summary"
Computer Science Laboratory, SRI-CSL-95-07, May 1995.

- [75] P.G. Neumann, P.A. Porras,
"Experience with EMERALD to DATE"
Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network
Monitoring, CA, pg. 73-80, April 1999.
- [76] ISSO/NSA
"ID-Secure Intrusion Detection Tools Database"
Information System Security Organisation (ISSO)/NSA,
<http://www.nswc.navy.mil/ISSEC/CID>, 1999.
- [77] J. Morrissey, P.W. Sanders, C.T. Stockel
"Increased domain security through application of local security and monitoring"
Expert Systems Journal, Vol. 13, No. 4, November 1996, pp. 296-305.
- [78] H Hamashige
"Cybercrime can kill venture: Small businesses can be destroyed by hackers, but
security measure can help"
CNNfn, 10 March 2000, <http://www.cnnfn.com>
- [79] Internet Engineering Task Force (IETF)
"Secure Sockets Layer Version 3.0 (SSL V3.0)"
IETF, <http://home.netscape.com/eng/ssl3/ssl-toc.html>
- [80] IBM
"RACF Version 2 Release 2 Technical Presentation Guide"
IBM ITSO Redbooks Publications, 2000,
<http://www.redbooks.ibm.com/abstracts/gg242539.html>
- [81] D. Pinkas, T. Parker, P. Kaijser
"SESAME: An Introduction"
Issue 1.2, Bull, ICL, and SNI, September 1993.
- [82] MIT
"RFC1510 – The Kerberos Network Authentication Service V5"
RFC index, <http://dsd.ds.internic.net/ds/rfc-index.html>
- [83] National Information System Security (INFOSEC)
"National Information System Security Glossary"
INFOSEC, NSTISSI No. 4009, June 5 1992.
- [84] WAP Forum
"Wireless Application Protocol White Paper"
WAP Forum, <http://www.wapforum.org>, June 2000.

- [85] **Network Working Group**
"Generic Security Service Application Program Interface, Version 2"
RFC 2078, January 1997.
- [86] **S. Muftic, A. Patel, P. Sanders, R. Colon, J. Heijnsdijk, U. Pulkkinen**
"Security Architecture for Open Distributed Systems"
J. Wiley & Sons, 1994.
- [87] **D. Brown, S. Montesi**
"Requirements upon TINA-C architecture"
TINAC, TB_MH.002_2.0_94, 17, February 1995.
- [88] **OMG**
"OMG White Paper on Security"
OMG, Issue: 1.0, April 1994.
- [89] **C. Cavanagh**
"CORBA Security White Paper"
Advanced Software, May 2001.
- [90] **ISO/IEC**
"Specification of Abstract Syntax Notation One (ASN.1)"
ISO/IEC 8824, 1990.
- [91] **ISO/IEC**
"Specification of the Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)"
ISO/IEC 8825, 1990.
- [92] **MSDN Magazine**
"A Young Person's Guide to the Simple Object Access Protocol: SOAP Increases Interoperability Across Platforms and Languages"
MSDN, <http://msdn.microsoft.com/masdmag/issues/0300/soap/soap.asp>, March 2000.
- [93] **OMG**
"COM / CORBA Mappings"
OMG, Document orbos/97-09-07, September 1997.
- [94] **M.J. Foley**
"MS middleware: An untapped goldmine"
Zdnet, <http://www.zdnet.com>, 22 April 1999.

- [95] **Microsoft**
"Windows 2000 Interoperability Solutions for Business"
Microsoft, <http://www.microsoft.com/windows2000/guide/server/solutions>,
September 2000.
- [96] **OMG**
"Common Secure Interoperability"
OMG Document, orbos/96-06-20, June 1998.
- [97] **W. Rosenberry, D. Kenney, G. Fisher**
"Understanding DCE"
O'Reilly & Associates, May 1993.
- [98] **W. Hu**
"DCE Security Programming"
O'Reilly & Associates, July 1995.
- [99] **P. Graubmann, W. Hwang, M. Kudela, K. MacKinnon, N. Mercouroff, N. Watanabe**
"Engineering Modelling Concept (DPE Architecture), version 2.0"
TINA-C, December 1994
- [100] **P. Kaijser**
"Security Protection for parts of a data structure"
Computer Communications, volume 17, number 7, July 1994.
- [101] **L. Bruno**
"Internet Security: How much is enough?"
Data Communications, April 1996.
- [102] **S. Castano, M. Fugini, G. Martella, P. Samarati**
"Database Security"
Addison Wesley, 1994.
- [103] **OMG**
"Persistence Object Service Specification"
OMG Document: orbos /97-12-12, December 1997
- [104] **IONA**
"OrbixSecurity Whitepaper".
IONA, <http://www.iona.com>, August 1999.

- [105] **D. Sullivan**
“*Beyond CORBA Security: Providing Network Authorisation Services*”
Internet Security Advisor, Volume 2, Number 2, 1999.
- [106] **Entegrity Solutions**
“*NetCrusader/CORBA Security Services for CORBA Applications*”
Entegrity Solutions Whitepaper v. 1.1, <http://www.entegrity.com>, January 2000.
- [107] **E. Gamma, R. Helm, R. Johnson, J. Vlissides**
“*Design Patterns – Elements of Reusable Object-Oriented Software*”
Addison Wesley, 1995.
- [108] **C.J. Date**
“*An Introduction to Database Systems – Volume II*”
Addison Wesley, ISBN 0-201-14474-3, 1985.
- [109] **OMG**
“*OMG RFP5 Submission: Trading Object Service*”
OMG Document orbos/96-05-06, Version 1.0.0, May 19 1996
- [110] **OMG**
“*Issues for CORBA Security 1.8 Revision Task Force*”
OMG, <http://cgi.omg.org/issues/sec-rev.html>, September 2000.
- [111] **OMG**
“*Security Special Interest Group Roadmap*”
OMG, <http://www.omg.org/homepage/sCCSig/security-sig-roadmap.htm>, September 2000.
- [112] **R. Sessions**
“*Ten Rules for Distributed Object Systems*”
OS/2 Magazine, Miller Freeman Inc. 1996.
- [113] **OMG**
“*TINA Standardization to Continue at the OMG*”
OMG, September 2000, <http://www.omg.org>
- [114] **OMG**
“*CORBA Interoperability approved as ISO Standard*”
OMG, October 2000, <http://www.omg.org>
- [115] **OMG**
“*Object Management Group Terms and Acronyms*”
OMG, 2000, http://www.omg.org/gettingstarted/groupterms_adn_acronyms.htm

- [116] **OMG**
"CORBA Security Service Version 1.5"
OMG, June 2000, <http://www.omg.org/cgi-bin/doc?orbos/2000-06-25>
- [117] **OMG**
"Security Service Revision 1.7"
OMG, 2000, <http://www.omg.org/cgi-bin/doc?security/99-12-02>
- [118] **OMG**
"Security 1.8 RTF"
OMG, 2000, <http://www.omg.org/technology/documents/formal/security-service.htm>
- [119] **D. Chizmadia**
"CORBASec: Securing Distributed Systems"
Software Development, June 1999.
<http://www.sdmagazine.com/supplement/ss/features/s996f1.shtm>
- [120] **OMG**
"OBJECT MANAGEMENT GROUP OUTLINES CORBA 3.0@ FEATURES"
Comdex Enterprise '98/Object World Conference, September 9, 1998
<http://www.omg.com/>
- [121] **J. Siegel**
"What's Coming in CORBA 3"
OMG 1999-2000, <http://www.omg.org>
- [122] **OMG**
"Java-to-IDL Mapping Specification"
OMG, 1999, <http://www.omg.org/cgi-bin/doc?orbos/99-03-09>
- [123] **OMG**
"CORBA 3 Firewall Specification"
OMG, 1998, <http://www.omg.org/cgi-bin/doc?orbos/98-05-04>
- [124] **OMG**
"The Interoperable Name Service"
OMG, 1998, <http://www.omg.org/cgi-bin/doc?orbos/98-10-11>
- [125] **OMG**
"Minimum CORBA Specification"
OMG, 1998, <http://www.omg.org/cgi-bin/doc?orbos/98-08-04>

- [126] **OMG**
"Real-time CORBA Specification"
OMG, 1999, <http://www.omg.org/cgi-bin/doc?orbos/99-02-012>
- [127] **OMG**
"Fault Tolerance RFP"
OMG, 1999,
[http://cgi.omg.org/techprocess/meetings/schedule/Fault Tolerance RFP.html](http://cgi.omg.org/techprocess/meetings/schedule/Fault_Tolerance_RFP.html)
- [128] **OMG**
"Messaging Specification"
OMG, 1998, <http://www.omg.org/cgi-bin/doc?orbos/98-05-05>
- [129] **Sun Microsystems**
"Enterprise JavaBeans Specification"
Sun Microsystems, 1998, <http://java.sun.com>
- [130] **OMG**
"The CORBA Component Model"
OMG, 1999,
<http://www.omg.org/cgi-bin/doc?orbos/99-07-01>
<http://www.omg.org/cgi-bin/doc?orbos/99-07-02>
<http://www.omg.org/cgi-bin/doc?orbos/99-07-03>
- [131] **OMG SECsig**
"OMG Security SIG (SECsig) Roadmap"
OMG, 2000, http://www.omg.org/homepages/secsig/security_sig_roadmap.htm
- [132] **Ericsson**
"FraudOffice Overview"
Ericsson, 2000, <http://www.ericsson.com/fraudoffice/overview>
- [133] **Ericsson**
"FraudOffice Benefits"
Ericsson, 2000, <http://www.ericsson.com/fraudoffice/benefits>
- [134] **Ericsson**
"Ericsson Fraud Management Solution FraudOffice"
Proceedings of Business Evolution & Components Seminar, Dublin, March 12 1999.
- [135] **CIDF**
"Common Intrusion Detection Framework"
CIDF, 2000, <http://www.gidos.org>

[136] IETF

"Intrusion Detection Work Group"

IETF, 2000, <http://www.ietf.org>

[137] OMG

"Persistence service problems"

OMG web site (<http://www.omg.org>).

[138] TINA-C

"TINA FAQ"

TINA-C, 2000, <http://www.tinac.com/faq/faq.htm#5>

Appendix A - IDL for Comprehensive CORBASec

This appendix will present the IDL for the Comprehensive CORBA Security Service (CORBASec). It has several structural changes from the original CORBA 2.0 Security Service, along with the modification and addition of new object interfaces.

Module Structure

The modules used in the IDL are now:

- Security
- SecurityLevel1
- SecurityLevel2
- SecurityAdmin
- SECIOP

The NRService and SecurityReplaceable modules have had their interfaces now included in the SecurityLevel2 and SecurityAdmin modules. There are two reasons for this. Non-repudiation is no longer an optional service and therefore is now included in the main modules. Also security replaceability is no longer required as mechanism and service independence is now built into the CORBA security structure.

Object Interfaces

Comments through out the IDL code will explain the modifications to and addition of new object interfaces. The new IDL code will be highlighted in **bold**.

IDL

```
// *****
// *           Module:      Security           *
// *           Function:    Defines data types etc. *
// *****
#include <orb.idl>

module Security {

    typedef string SecurityName;
    typedef sequence <octet> Opaque;

    // Used to define the Policy type
    enum PolicyType {
        ClientInvocationAccess,
        TargetInvocationAccess,
        ApplicationAccess,
        ClientInvocationAuthentication,
        TargetInvocationAuthentication,
        ApplicationAuthentication,
        ClientInvocationQoP,
        TargetInvocationQoP,
        ApplicationQoP,
        StoredDataQoP,
        ClientInvocationAudit,
        TargetInvocationAudit,
        ApplicationAudit,
        ClientInvocationNonRepudiation,
        TargetInvocationNonRepudiation,
        ApplicationNonRepudiation,
        ClientInvocationDelegation,
        TargetInvocationDelegation,
        ApplicationDelegation,
        ClientSecureInvocation,
        TargetSecureInvocation,
        ApplicationSecureInvocation,
        Construction
    };

    // Used to define the Principal type for the Policy
    // NOTE: UA represents the entry point for a user to the system.
    //       User has not yet obtained any id,attributes etc.
    enum PrincipalType {
        Principal,
        UA
    };
};
```

```
// extensible families for standard data types
struct ExtensibleFamily {
    unsigned short    family_definer;
    unsigned short    family;
};

// security association mechanism type
typedef string MechanismType;
typedef sequence <MechanismType> MechanismTypeList;

struct SecurityMechandName {
    MechanismType      mech_type;
    SecurityName       security_name;
};
typedef sequence <SecurityMechandName> SecurityMechandNameList;

// security attributes
typedef unsigned long SecurityAttributeType;

// identity attributes; family=0
const SecurityAttributeType AuditId = 1;
const SecurityAttributeType AccountingID = 2;
const SecurityAttributeType NonRepudiationId = 3;

// privilege attributes; family = 1
const SecurityAttributeType Public = 1;
const SecurityAttributeType AccessID = 2;
const SecurityAttributeType PrimaryGroupID = 3;
const SecurityAttributeType GroupId = 4;
const SecurityAttributeType Role = 5;
const SecurityAttributeType AttributeSet = 6;
const SecurityAttributeType Clearance = 7;
const SecurityAttributeType Capability = 8;

struct AttributeType {
    ExtensibleFamily      attribute_family;
    SecurityAttributeType attribute_type;
};

typedef sequence<AttributeType> AttributeTypeList;

struct Attribute {
    AttributeType      attribute_type;
    sequence <octet>    defining_authority;
    Opaque             value;
};
```

```
typedef sequence<Attribute>      AttributeList;

typedef sequence<octet>          def_authority;

// Authentication return status
enum AuthenticationStatus {
    Success,
    Failure,
    Continue,
    Expired
};

// Association return status
enum AssociationStatus {
    AoSuccess,
    AoFailure,
    AoContinue
};

//Authentication method
typedef unsigned long AuthenticationMethod;

//Access Control method
typedef unsigned long AccessMethod;

// Authentication Types
enum AuthenticationType {
    Client,
    Server,
    Mutual
};

// Credential types which can be set as Current default
enum CredentialType {
    InvocationCredentails,
    OwnCredentials,
    NRCredentials
};

// Declarations related to Rights
struct Right {
    ExtensibleFamily    rights_family;
    string              right;
};

typedef sequence <Right> RightsList;

enum RightsCombinator {
```

```
        AllRights,
        AnyRights
};

// Delegation related
enum DelegationState {
    Initiator,
    Delegate
};

//pick up from TimeBase
typedef TimeBase::UtcT          UtcT;
typedef TimeBase::IntervalT    IntervalT;
typedef TimeBase::TimeT        TimeT;


// Security features available on credentials
enum SecurityFeature {
    NoDelegation,
    SimpleDelegation,
    CompositeDelegation,
    NoProtection,
    Integrity,
    Confidentiality,
    IntegrityAndConfidentiality,
    DetectReplay,
    DetectMisordering,
    EstablishTrustInTarget,
    Anonymity
};

struct SecurityFeatureValue {
    SecurityFeature    feature;
    boolean            value;
};

typedef sequence<SecurityFeatureValue> SecurityFeatureValueList;

// Quality of protection which can be specified
// for an object ref and used to protect messages
enum QOP {
    QOPNoProtection,
    QOPIntegrity,
    QOPConfidentiality,
    QOPIntegrityAndConfidentiality
};
```

Appendix A: IDL for Comprehensive CORBA Security

```
// Association options which can be administered on secure invocation
// policy and used to initialise security context
typedef unsigned short      AssociationOption;

const AssociationOption AONoProtection = 1;
const AssociationOption AOIntegrity = 2;
const AssociationOption AOConfidentiality = 4;
const AssociationOption AODetectReplay = 8;
const AssociationOption AODetectMisordering = 16;
const AssociationOption AOEstablishTrustInClient = 32;
const AssociationOption AOEstablishTrustInTarget = 64;
const AssociationOption AOAnonymity = 128;

typedef sequence <AssociationOption> AssociationOptions;

// Flag to indicate whether association options being administered
// are the "required" or "supported" set
enum RequiresSupports {
    Requires,
    Supports
};

// Direction of communication for which secure invocation
// policy applies
enum CommunicationDirection {
    Both,
    Request,
    Reply
};

// AssociationOptions-Direction pair
struct OptionsDirectionPair {
    AssociationOptions  options;
    CommunicationDirection  direction;
};

typedef sequence<OptionsDirectionPair> OptionsDirectionPairList;

// Delegation mode which can be administered
enum DelegationMode {
    DNoDelegation,
    DsimpleDelegation,
    DCompositeDelegation
};

// Association options supported by a given mech type
struct MechandOptions {
    MechanismType      mechanism_type;
```

```
        AssociationOptions        options_supported;
};
typedef sequence <MechandOptions> MechandOptionsList;

//Audit
struct AuditEventType {
    ExtensibleFamily        event_family;
    unsigned short          event_type;
};
typedef sequence <AuditEventType> AuditEventTypeList;

typedef unsigned long SelectorType;
// family = 1, System event selectors
const SelectorType    Intface = 1;
const SelectorType    Obj = 2;
const SelectorType    Operation = 3;
const SelectorType    SelInitiator = 4;
const SelectorType    SuccessFailure = 5;
const SelectorType    Time = 6;

typedef sequence<SelectorType> SelectorTypeList;

struct SelectorValue {
    SelectorType selector;
    any          value;
};
typedef sequence <SelectorValue> SelectorValueList;

// used by AuditAnalyser in analyse_data
enum AnalyserResult {
    0=no_violation,
    1=raise_suspicion,
    2=violation
}

// used by AuditAnalyser when justifying audit analysis
struct AuditJustify {
    string        justification_message;
    Opaque        justification_data;
};

// Msg_part used by QOP to specify how much of the message should
// have integrity/confidentiality mechanisms applied
enum msg_part {
    parameters,
    parameters_operations,
}
```

Appendix A: IDL for Comprehensive CORBA Security

```
        parameters_operations_targetId,
        parameters_operations_targetId_serviceInfo
};
typedef sequence <msg_part> MsgPartList;

// Used in the Interoperability Interface
enum InteropPolicyType {
    Std_mechs,
    Translator
};

// Used to define operator types used in access decisions
enum OperatorType {
    GT,
    LT,
    EQ,
    GE,
    LE,
    NE
};

// Used in securitylevel2
typedef unsigned short minor_status;
typedef unsigned short major_status;
typedef-Opaque errormsg;

typedef Security::MechanismType NRmech;
typedef Security::ExtensibleFamily    NRPolicyId;

enum NRVerificationResult {
    invalid,
    valid,
    ConditionallyValid
};

// The following are used for evidence validity duration
// month = 30 days; year = 365 days

typedef unsigned long duration_in_minutes;

const duration_in_minutes  DURATION_HOUR = 60;
const duration_in_minutes  DURATION_DAY  = 1440;
const duration_in_minutes  DURATION_WEEK = 10080;
const duration_in_minutes  DURATION_MONTH= 43200;
const duration_in_minutes  DURATION_YEAR = 525600;
```



```
typedef long time_offset_in_minutes;

// last_revocation_check_offset may be >0 or <0; add this to evidence
// generation time to get latest time at which mech will check to
// see if this authority's key has been revoked.
struct authorityDescriptor {
    string                authority_name;
    string                authority_role;
    time_offset_in_minutes last_revocation_check_offset;
};
typedef sequence <authorityDescriptor> authorityDescriptorList;

// max_time_skey is max permissible difference between evidence
// generated time and time of service countersignature
// ignored if trusted time not required.
struct mechanismDescriptor {
    NRmech                mech_type;
    authorityDescriptorList authority_list;
    time_offset_in_minutes max_time_skew;
};
typedef sequence <mechanismDescriptor> mechanismDescriptorList;

enum EvidenceType {
    ProofofCreation,
    ProofofSubmission,
    ProofofReceipt,
    ProofofApproval,
    ProofofRetrieval,
    ProofofOrigin,
    ProofofDelivery,
    NoEvidence
};

enum EvidenceDirection {
    Evidence,
    RequestedEvidence
};

struct evidenceDescriptor {
    EvidenceType    evidence_type;
    duration_in_minutes evidence_validity_duration;
    boolean         must_use_trusted_time;
};
typedef sequence <evidenceDescriptor> evidenceDescriptorList;
```

```
struct NRPolicyFeatures {
    NRPolicyId          policy_id;
    unsigned long       policy_version;
    NRmech              mechanism;
};
typedef sequence<NRPolicyFeatures> NRPolicyFeaturesList;

// features used when generating requests
struct requestFeatures {
    NRPolicyFeatures    requested_policy;
    Security::EvidenceType requested_evidence;
    string              requested_evidence_generators;
    string              requested_evidence_recipients;
    boolean              include_this_token_in_evidence;
};

// Used in the NRAdjudicator and NRStore
enum DecisionType {
    originator,
    target,
    undecided,
    neither
};

enum ServiceType {
    Authentication,
    AccessControl,
    Delegation,
    QofP,
    Audit,
    NonRepudiation
};
typedef string Constraint;
typedef string Preference;
typedef unsigned long MappingId;
typedef sequence<MappingId> MappingIdSeq;

struct domain_values{
    sequence<octet> domain1_value;
    sequence<octet> domain2_value;
};
typedef sequence<domain_values> domain_values_list;

// Domain Mapping structure
struct Mapping {
    MappingId          mappingId;
    ServiceType        serviceType;
    string policy       Classification;
};
```

```

        unsigned short          family_definer1;
        unsigned short          family_id1;
        sequence<octet>         attribute_type1;
        sequence<octet>         defining_authority1;
        unsigned short          family_definer2;
        unsigned short          family_id2;
        sequence<octet>         attribute_type2;
        sequence<octet>         defining_authority2;
        sequence<domain_values> mapped_values;
        Security::UtcT          timeStamp;
        Security::Opaque         remoteDomainId;
        Security::Opaque         remoteDomainAuthority;
};
typedef sequence<Mapping> MappingSeq;

//Interoperability Policy Structures
struct MechRequiresSupports {
    sequence<string>             mech_required;
    sequence<Security::CommunicationDirection>
    mech_required_direction;
    sequence<string>             mechs_supported;
    sequence<Security::CommunicationDirection>
    mechs_supported_direction;
};

struct AuthPolicyRequiresSupports{
    AuthenticationType           auth_type_required;
    CommunicationDirection       auth_type_required_direction;
    sequence<AuthenticationType> auth_type_supported;
    sequence<CommunicationDirection>
    auth_type_supported_direction;
};

struct SecureInvocationFamily {
    string                       policy_classification;
    ExtensibleFamily             event_family;
};
typedef sequence <SecureInvocationFamily> SecureInvocationFamilyList;

struct DelegationPolicyRequiresSupports{
// type = none,simple, composite
    Security::DelegationMode     mode_required;
    Security::CommunicationDirection
    mode_required_direction;
    sequence<Security::DelegationMode> mode_supported;
    sequence<Security::CommunicationDirection>
    mode_supported_direction;
};

```

```

struct QOPPolicyRequiresSupports{
    AssociationOptions                qop_type_required;
    sequence<CommunicationDirection> qop_type_required_direction;
    AssociationOptions                qop_type_supported;
    sequence<CommunicationDirection> qop_type_supported_direction;
    MsgPartList                      integ_msg_part_required;
    sequence<CommunicationDirection> integ_msg_part_supported;
    integ_msg_part_supported_direction;
    MsgPartList                      conf_msg_part_required;
    sequence<CommunicationDirection> conf_msg_part_supported;
    conf_msg_part_supported_direction;
};

struct NRPolicyRequiresSupports{
    Security::evidenceDescriptorList evidence_required;
    sequence<Security::CommunicationDirection>
    evidence_required_direction;
    Security::evidenceDescriptorList evidence_supported;
    sequence<Security::CommunicationDirection>
    evidence_supported_direction;
    Security::authorityDescriptorList authorities;
};

struct AuditPolicyRequiresSupports{
    Security::AuditEventTypeList event_required;
    sequence<Security::CommunicationDirection>
    event_required_direction;
    Security::AuditEventTypeList event_supported;
    sequence<Security::CommunicationDirection>
    event_supported_direction;
    Security::SelectorTypeList selector_required;
    Security::SelectorTypeList selector_supported;
};

// END OF SECURITY DATA MODULE
};

// *****
// *      Module:      1      *
// *      Function:    Security Level 1 Interfaces.  *
// *****

```

```
//module securitylevel1 {

//interface Current : CORBA1::Current {
interface Current {
    Security::AttributeList get_attributes (
        in Security::AttributeList    attributes
    );
};

// END OF securitylevel1 MODULE
};

// *****
// *          Module:          securitylevel21          *
// *          Function:        Security Level 2 Interfaces (SL2)      *
// *****
module securitylevel2 {
typedef string Identifier;
typedef string InterfaceName;

// moved RequiredRights because SL2 interfaces refer to RequiredRights.
// Previously these interfaces were in SecurityReplaceability module but they
// are all now part of SL2.
interface RequiredRights;
interface UserAgent;
interface PrincipalAuthenticator;
interface Credentials;
interface Object2;
interface Current;

// RequiredRights Interface
interface RequiredRights {
    void get_required_rights(
        in Object                object,
        in Identifier            operation_name,
        in InterfaceName        interface_name,
        in string                parameter_name,
        in Security::OperatorType operator,
        in Security::Opaque      parameter_value,
        out Security::RightsList rights,
        out Security::RightsCombinator rights_combinator
    );

    void set_required_rights (
        in string                operation_name,
        in InterfaceName        interface_name,
        in string                parameter_name,
        in Security::OperatorType operator,

```

```

        in Security::Opaque          parameter_value,
        in Security::RightsList      rights,
        in Security::RightsCombinator rights_combinator
    );
};

interface PrincipalAuthenticator {
    Security::AuthenticationStatus authenticate (
        in Security::AuthenticationMethod method,
        in string security_name,
        in Security::Opaque auth_data,
        in Security::AttributeList privileges,
        out Credentials creds,
        out Security::Opaque continuation_data,
        out Security::Opaque auth_specific_data
    );

    Security::AuthenticationStatus continue_authentication (
        in Security::Opaque response_data,
        inout Credentials creds,
        out Security::Opaque continuation_data,
        out Security::Opaque auth_specific_data
    );
};

// Interface Credentials
interface Credentials {
    void set_security_features (
        in Security::CommunicationDirection direction,
        in Security::SecurityFeatureValueList security_features,
        out Security::errormsg error,
        out Security::major_status major_error,
        out Security::minor_status minor_error
    );

    Security::SecurityFeatureValueList get_security_features (
        in Security::CommunicationDirection direction,
        out Security::errormsg error,
        out Security::major_status major_error,
        out Security::minor_status minor_error
    );

    boolean set_privileges (

```

```
        in boolean
        in Security::AttributeList
        out Security::AttributeList
        out Security::errormsg
        out Security::major_status
        out Security::minor_status

        force_commit,
        requested_privileges,
        actual_privileges,
        error,
        major_error,
        minor_error

    );

    Security::AttributeList get_attributes (
        in Security::AttributeTypeList
        out Security::errormsg
        out Security::major_status
        out Security::minor_status

        attributes,
        error,
        major_error,
        minor_error

    );

    boolean set_controls (
        in boolean
        in Security::AttributeList
        in Security::DelegationMode
        in Security::UtcT
        in Security::AttributeList
        in long
        out Security::errormsg
        out Security::major_status
        out Security::minor_status

        force_commit,
        required_attributes,
        delegation_mode,
        expiry_time,
        privileges_delegated,
        no_of_invocations,
        error,
        major_error,
        minor_error

    );

    boolean get_controls (
        in Security::AttributeList
        out boolean
        out Security::DelegationMode
        out Security::UtcT
        out Security::AttributeList
        out long
        out Security::errormsg
        out Security::major_status
        out Security::minor_status

        required_attributes,
        force_commit,
        delegation_mode,
        expiry_time,
        privileges_delegated,
        no_of_invocations,
        error,
        major_error,
        minor_error

    );

    boolean is_valid (
        out Security::UtcT
        out Security::errormsg
        out Security::major_status
        out Security::minor_status

        expiry_time,
        error,
        major_error,
        minor_error

    );
```

```

        boolean refresh();
};

typedef sequence<Credentials> CredentialsList;

// Interface object derived from Object
// providing additional operations on objref at this security level.
interface Object : CORBA::Object{
    void override_default_credentials (
        in Credentials                creds
    );

    void override_default_QOP (
        in Security::QOP              qop
    );

    Security::SecurityFeatureValueList get_security_features (
        in Security::CommunicationDirection direction
    );

    Credentials get_active_credentials();

    CORBA::Policy get_policy (
        long get_policy (
            in Security::PolicyType    policy_type
        );

    Security::MechanismType get_security_mechanism();

    void override_default_mechanism (
        in Security::MechanismType    mechanism_type
    );

    Security::SecurityMechandName get_security_names();
};

// Interface Current derived from securitylevel11::Current
// providing additional operations on Current at this security
// level. This is implemented by the ORB.

interface Current {
    Security::AttributeList                get_attributes (
        in Security::AttributeTypeList    attributes
    );

    void set_credentials (
        in Security::CredentialType        cred_type,

```


Appendix A: IDL for Comprehensive CORBA Security

```
        in Credentials                creds
    );

    readonly attribute CredentialsList    received_credentials;

    readonly attribute Security::SecurityFeatureValueList
        received_security_features;

    CORBA::Policy get_policy(
        in Security::PolicyType          policy_type
    );

    readonly attribute RequiredRights required_rights_object;

};

// AUDIT OBJECTS

// Interface for AuditDecision
interface AuditDecision {
    boolean audit_needed (
        in Security::AuditEventType          event_type,
        in Security::SelectorValueseq        value_list
    );
};

// Interface for AuditAnalyser
interface AuditAnalyser {
    boolean analyse_data (
        in Security::AuditEventType          event_type,
        in CredentialsList                   creds,
        in Security::UtcT                    time,
        in Security::SelectorSequence        descriptors,
        in Security::Opaque                  event_specific_data,
        out Security::AnalyserResult         result,
        out Security::Opaque                  analysis_token
    );

    boolean justify (
        in Security::Opaque                  analysis_token,
        out sequence<AuditJustify>           justification
    );
};

// Interface for AuditResponder
interface AuditResponder {
    boolean define_response (
        in Security::AnalyserResult         result,
```

```

        in Security::Opaque
        out sequence<Security::Opaque>
        out sequence<Object>
        audit_token,
        audit_data,
        audit_channels
    );

};

// Interface AuditChannel
interface AuditChannel {
    readonly attribute Object
        linked_object,

    boolean audit_write (
        in Security::Opaque
        out Security::errormsg
        out Security::major_status
        out Security::minor_status
        audit_data
        error,
        major_error,
        minor_error
    );
};

// Interface AuditTrail
interface AuditTrail {
    boolean read_record (
        in long
        out Security::AuditEventType
        out CredentialsList
        out Security::UtcT
        out Security::SelectorSequence
        out Security::Opaque
        out Security::AnalyserResult
        out Security::Opaque
        out Security::errormsg
        out Security::major_status
        out Security::minor_status
        id,
        event_type,
        creds,
        time,
        descriptors,
        event_specific_data,
        result,
        analysis_token
        error,
        major_error,
        minor_error
    );

    boolean write_record (
        in long
        in Security::AuditEventType
        in CredentialsList
        in Security::UtcT
        in Security::SelectorSequence
        in Security::Opaque
        in Security::AnalyserResult
        in Security::Opaque
        out Security::errormsg
        out Security::major_status
        out Security::minor_status
        id,
        event_type,
        creds,
        time,
        descriptors,
        event_specific_data,
        result,
        analysis_token
        error,
        major_error,
        minor_error
    );
};

```

```

    );

    boolean query_record (
        inout sequence<long>                                id,
        inout sequence<Security::AuditEventType>            event_type,
        inout sequence<CredentialsList>                     creds,
        inout sequence<Security::UtcT>                      time,
        inout sequence<Security::SelectorSequence>          descriptors,
        inout sequence<Security::Opaque>                    event_specific_data,
        inout sequence<Security::AnalyserResult>            result,
        inout sequence<Security::Opaque>                   analysis_token,
        out Security::errormsg                               error,
        out Security::major_status                           major_error,
        out Security::minor_status                           minor_error
    );
};

// Interface AuditAction
interface AuditAction {
    boolean get_action_info (
        in long                                              id,
        out Security::Opaque                                action_data
    );

    boolean execute_action (
        in Security::Opaque                                action_data
        out Security::errormsg                               error,
        out Security::major_status                           major_error,
        out Security::minor_status                           minor_error
    );
};

// *****
// *          Module:      NRservice                      *
// *          Function:    Non-Repudiation interfaces      *
// *****

//Interface NRCredentials
interface NRCredentials {
    boolean set_NR_features (
        in Security::NRPolicyFeaturesList                  requested_features,
        in Security::NRPolicyFeaturesList                  actual_features
    );

    Security::NRPolicyFeaturesList get_NR_features();

    void generate_token (
//          in sequence <octet>                             input_buffer,

```

Appendix A: IDL for Comprehensive CORBA Security

```

        in Security::Opaque
        in Security::EvidenceType
        in boolean
        in boolean
        in Security::requestFeatures
        in boolean
        out Security::Opaque
        out Security::Opaque
        out Security::errormsg
        out Security::major_status
        out Security::minor_status

        input_buffer,
        generate_evidence_type,
        include_data_in_token,
        generate_request,
        request_features,
        input_buffer_complete,
        nr_token,
        evidence_check,
        error,
        major_error,
        minor_error

    );

    Security::NRVerificationResult verify_evidence (
        in Security::Opaque
        in Security::Opaque
        in boolean
        in boolean
        out Security::Opaque
        out Security::Opaque
        out sequence <octet>
        out boolean
        out boolean
        out Security::TimeT
        out Security::TimeT
        out Security::errormsg
        out Security::major_status
        out Security::minor_status

        input_token_buffer,
        evidence_check,
        form_complete_evidence,
        token_buffer_complete,
        output_token,
        data_included_in_token,
        data_included_in_token,
        evidence_is_complete,
        trusted_time_used,
        complete_evidence_before,
        complete_evidence_after,
        error,
        major_error,
        minor_error

    );

    void get_token_details (
        in Security::Opaque
        in boolean
        out string
        out Security::NRPolicyFeatures
        out Security::EvidenceType
        out Security::UtcT
        out Security::UtcT
        out Security::duration_in_minutes
        out boolean
        out boolean
        out Security::requestFeatures
        out Security::errormsg
        out Security::major_status
        out Security::minor_status

        token_buffer,
        token_buffer_complete,
        token_generator_name,
        policy_features,
        evidence_type,
        evidence_generation_time,
        evidence_valid_start_time,
        evidence_validity_duration,
        data_included_in_token,
        request_included_in_token,
        request_features,
        error,
        major_error,
        minor_error
    )

```

```

);

boolean form_complete_evidence (
    in Security::Opaque
    out Security::Opaque
    out boolean
    out Security::TimeT
    out Security::TimeT
    out Security::errormsg
    out Security::major_status
    out Security::minor_status
    input_token,
    output_token,
    trusted_time_used,
    complete_evidence_before,
    complete_evidence_after,
    error,
    major_error,
    minor_error
);

};

// interface NRDeliver
interface NRDeliver {
//NR_send_generated_token will send a token and its input data to the
//target object specified.
    boolean NR_deliver_token(
        in Security::Evidence Direction
        in Security::EvidenceType
        in Security::Opaque
        in Security::Opaque
        in boolean
        in Object
        in Object
        out Security::errormsg
        out Security::major_status
        out Security::minor_status
        evidence_direction,
        evidence_type,
        nr_token,
        evidence_check,
        data_in_token,
        originator,
        target,
        error,
        major_error,
        minor_error
    );

// interface NRStore
interface NRStore {
//The NR_record_add method returns a value of True/False depending on
//whether the record was added successfully. If False, errormsg will
//contain a systems message, explaining the problem, or the minor_error
//will contain a mechanism dspecific message (GSS-API compliance).
//Otherwise the error parameters will be null.
    boolean NR_record_add (
        in Security::Opaque
        in CredentialsList
        in Security::EvidenceDirection
        in Security::EvidenceType
        in boolean
        in Security::Opaque
        nr_token,
        nr_creds,
        evidence_direction,
        evidence_type,
        data_in_token,
        evidence_check,

```

```

        in Security::UtcT                nr_store_time,
        out Security::Opaque             nr_index,
        out Security::errormsg           error,
        out Security::major_status       major_error,
        out Security::minor_status       minor_error
    );

//An index is supplied to retrieve the appropriate key. This can be the
//result of a query or iterator operation (Query and Collection Service).
//The NR_record_get method returns a value of True/False depending on
//whether the record was successfully retrieved. If False, errormsg will
//contain a systems message, explaining the problem, or the minor_error
//will contain a mechanism specific message (GSS-API compliance).
//Otherwise the error parameters will be null.
    boolean NR_record_get (
        in Security::Opaque             nr_index,
        out Security::EvidenceDirection evidence_direction,
        out Security::EvidenceType      evidence_type,
        out boolean                     data_in_token,
        out CredentialsList             nr_creds,
        out Security::Opaque            nr_token,
        out Security::Opaque            evidence_check,
        out Security::UtcT              nr_store_time,
        out Security::errormsg           error,
        out Security::major_status       major_error,
        out Security::minor_status       minor_error
    );

    boolean NR_record_query (
        inout sequence<Security::Opaque>      nr_index,
        inout sequence<Security::EvidenceDirection>
        evidence_direction,
        inout sequence<Security::EvidenceType> evidence_type,
        inout sequence<boolean>               data_in_token,
        inout sequence<CredentialsList>       nr_creds,
        inout sequence<Security::Opaque>      nr_token,
        inout sequence<Security::Opaque>      evidence_check,
        inout sequence<Security::UtcT>        nr_store_time,
        out Security::errormsg                 error,
        out Security::major_status             major_error,
        out Security::minor_status             minor_error
    );
};

//interface NRAdjudicator
interface NRAdjudicator{
    boolean NR_settle_dispute (

```

```

        in Object                                originator,
        in Security::Opaque                      originator_nr_token,
        in Object                                target,
        in Security::Opaque                      target_nr_token,
        out Security::Opaque                     nr_decision_token,
        out Security::DecisionType               decision,
        out Security::errormsg                   error,
        out Security::major_status               major_error,
        out Security::minor_status               minor_error
    );

};

// The NRPolicy has been removed from this module and placed in the
// SecurityAdmin module.

// *****
// *      Module:      SecurityReplacable      *
// *      Function:    Allows replacability    *
// *****

interface SecurityContext;

// INTERFACE VAULT
interface Vault {
    Security::AssociationStatus init_security_context (
        in CredentialsList          creds_list,
        in Security::SecurityName   target_security_name,
        in Object                    target,
        in Security::OptionsDirectionPairList association_options,
        in Security::MechanismType  mechanism,
        in Security::Opaque          mech_data,
        in Security::Opaque          chan_binding,
        inout short                   lifetime_rec,
        out Security::MechanismType  out_mechanism,
        out boolean                   deleg_state,
        out boolean                   mutual_state,
        out boolean                   replay_det_state,
        out boolean                   sequence_state,
        out boolean                   anon_state,
        out boolean                   trans_state,
        out boolean                   prot_ready_state,
        out boolean                   conf_avail,
        out boolean                   integ_avail,
        out Security::Opaque          security_token,
        out SecurityContext           security_context,
        out Security::errormsg        error,
    );

```

Appendix A: IDL for Comprehensive CORBA Security

```

        out Security::major_status      major_error,
        out Security::minor_status      minor_error
    );

    Security::AssociationStatus accept_security_context (
        in CredentialsList              creds_list,
        in Security::Opaque              chan_bindings,
        in Security::Opaque              in_token,
        out boolean                      deleg_state,
        out boolean                      mutual_state,
        out boolean                      replay_det_state,
        out boolean                      sequence_state,
        out boolean                      anon_state,
        out boolean                      trans_state,
        out boolean                      prot_ready_state,
        out boolean                      conf_avail,
        out boolean                      integ_avail,
        out CredentialsList              delegated_creds_list,
        out Security::Opaque              out_token,
        out short                        lifetime_rec,
        out SecurityContext               security_context,
        out Security::errormsg            error,
        out Security::major_status        major_error,
        out Security::minor_status        minor_error
    );

    Security::MechandOptionsList         get_supported_mechs();
};

// Interface SecurityContext
interface SecurityContext {
    readonly attribute CredentialsList received_credentials;

    readonly attribute Security::SecurityFeatureValueList
    received_security_features;

    Security::AssociationStatus continue_security_context (
        in Security::Opaque              in_token,
        out Security::Opaque              out_token,
        out Security::errormsg            error,
        out Security::major_status        major_error,
        out Security::minor_status        minor_error
    );

    void protect_message (
        in Security::Opaque              message,

```



```

        in Security::QOP
        inout boolean
        out Security::Opaque
        out Security::Opaque
        out Security::errmsg
        out Security::major_status
        out Security::minor_status
    );

    boolean reclaim_message (
        in Security::Opaque
        in Security::Opaque
        out Security::QOP
        out boolean
        out Security::Opaque
        out Security::errmsg
        out Security::major_status
        out Security::minor_status
    );

    boolean is_valid (
        out Security::UtcT
        out Security::errmsg
        out Security::major_status
        out Security::minor_status
    );

    boolean refresh();
};

//Interface AccessDecision
interface AccessDecision {
    boolean access_allowed (
        in CredentialsList
        in Object
        in Identifier
        in string
        in Security::OperatorType
        in Security::Opaque
        in Identifier
        cred_list,
        target,
        operationName,
        parameter_name,
        operator,
        parameter_value,
        targetInterfaceName
    );
};

// END OF SECURITYLEVEL2 MODULE
};

// *****

```

```
// *           Module:           SecurityAdmin           *
// *           Function:        Administration Interfaces   *
// *****
module SecurityAdmin {

interface MappingIterator;

interface UserAgent{
    void set_security_name (
        in string                security_name,
        out Security::errormsg    error,
        out Security::major_status major_error,
        out Security::minor_status minor_error
    );

    void set_auth_data (
        in Security::Opaque       auth_data,
        out Security::errormsg    error,
        out Security::major_status major_error,
        out Security::minor_status minor_error
    );

    void set_privileges (
        in Security::AttributeList privileges,
        out Security::errormsg    error,
        out Security::major_status minor_error
    );

    void set_name (
        in Security::Opaque       security_name,
        out Security::errormsg    error,
        out Security::major_status major_error,
        out Security::minor_status minor_error
    );

    Security::AuthenticationStatus authenticate (
        out Security::Opaque       continuation_data,
        out Security::Opaque       auth_specific_data,
        out Security::errormsg    error,
        out Security::major_status major_error,
        out Security::minor_status minor_error
    );

//The above methods are used prior to authenticate. The following method
//is used after the authenticate and with continue_authentication.

    Security::AuthenticationStatus reply_to_challenge (
        in Security::Opaque       response_data,
```

```

        out Security::errormsg          error,
        out Security::major_status      major_error,
        out Security::minor_status      minor_error
    );

};

//Interface QOPPolicy
interface QOPPolicy {
    readonly attribute Security::PolicyType  policy_type;

    void set_QOP_policy(
        in long                                policy_id,
        in Security::InterfaceDefInfo          object_type,
        in Security::QOP                      QOP_type,
        in long                                integrity_mech,
        in Security::msg_part                  integrity_msg_part,
        in long                                confidentiality_mech,
        in Security::msg_part                  confidentiality_msg_part,
        in Security::CommunicationDirection    direction,
        in Security::UtcT                      expiry_time
    );

    void get_QOP_policy(
        inout long                            policy_id,
        inout Security::InterfaceDefInfo      object_type,
        out Security::QOP                    QOP_type,
        out long                             integrity_mech,
        out Security::msg_part                integrity_msg_part,
        out long                             confidentiality_mech,
        out Security::msg_part                confidentiality_msg_part,
        out Security::CommunicationDirection direction,
        out Security::UtcT                    expiry_time
    );

    void query_QOP_policy(
        inout long                            policy_id,
        inout sequence<Security::InterfaceDefInfo> object_type,
        inout sequence<Security::QOP>         QOP_type,
        inout sequence<long>                  integrity_mech,
        inout sequence<Security::msg_part>    integrity_msg_part,
        inout sequence<long>                  confidentiality_mech,
        inout sequence<Security::msg_part>    confidentiality_msg_part,
        inout sequence<Security::CommunicationDirection> direction,
        inout sequence<Security::UtcT>        expiry_time
    );
};

```

```

void update_QOP_policy(
    in long
    inout Security::InterfaceDefInfo
    inout Security::QOP
    inout long
    inout Security::msg_part
    inout long
    inout Security::msg_part
    inout Security::CommunicationDirection
    inout Security::UtcT
    policy_id,
    object_type,
    QOP_type,
    integrity_mech,
    integrity_msg_part,
    confidentiality_mech,
    confidentiality_msg_part,
    direction,
    expiry_time
);

void delete_QOP_policy(
    in long
    inout Security::InterfaceDefInfo
    policy_id,
    object_type,
);

void set_stored_QOP_policy(
    in long
    in Security::InterfaceDefInfo
    in Security::QOP
    in long
    in long
    in Security::UtcT
    policy_id,
    object_type,
    QOP_type,
    integrity_mech,
    confidentiality_mech,
    expiry_time
);

void get_stored_QOP_policy(
    inout long
    inout Security::InterfaceDefInfo
    out Security::QOP
    out long
    out long
    out Security::UtcT
    policy_id,
    object_type,
    QOP_type,
    integrity_mech,
    confidentiality_mech,
    expiry_time
);

void query_stored_QOP_policy(
    inout sequence<long>
    inout sequence<Security::InterfaceDefInfo>
    inout sequence<Security::QOP>
    inout sequence<long>
    inout sequence<long>
    inout sequence<Security::UtcT>
    policy_id,
    object_type,
    QOP_type,
    integrity_mech,
    confidentiality_mech,
    expiry_time
);

void update_stored_QOP_policy(
    in long
    inout Security::InterfaceDefInfo
    inout Security::QOP
    policy_id,
    object_type,
    QOP_type,

```

```

        inout long                integrity_mech,
        inout long                confidentiality_mech,
//      inout Security::RequiresSupports requires_supports,
        inout Security::UtcT      expiry_time
    );

    void delete_stored_QOP_policy(
        in long                    policy_id,
        inout Security::InterfaceDefInfo object_type
    );
};

//Interface QOPMechanism
interface QOPMechanism {

    readonly attribute Security::PolicyType policy_type;

// Integrity operations
    void set_Integrity_mech (
        in long                    integrity_mech,
        in string                  integrity_mech_name,
        in Security::Opaque        parameters,
        in Security::Opaque        remote_parameters,
        in boolean                  Standard_mechanism,
        in Security::Opaque        interface_details,
        in Security::UtcT          expiry_time
    );

    void get_Integrity_mech (
        in long                    integrity_mech,
        in string                  integrity_mech_name,
        inout Security::Opaque      parameters,
        inout Security::Opaque      remote_parameters,
        inout boolean                Standard_mechanism,
        inout Security::Opaque      interface_details,
        inout Security::UtcT        expiry_time
    );

    void query_Integrity_mech (
        inout sequence<long>        integrity_mech,
        inout sequence<string>      integrity_mech_name,
        inout sequence<Security::Opaque> parameters,
        inout sequence<Security::Opaque> remote_parameters,
        inout sequence<boolean>     Standard_mechanism,
        inout sequence<Security::Opaque> interface_details,
        inout sequence<Security::UtcT> expiry_time
    );
};

```

```

    void delete_Integrity_mech (
        in long                                integrity_mech
    );

// Confidentiality operations
    void set_Confidentiality_mech (
        in long                                confidentiality_mech,
        in string
        confidentiality_mech_name,
        in Security::Opaque                    parameters,
        in Security::Opaque                    remote_parameters,
        in boolean                             Standard_mechanism,
        in Security::Opaque                    interface_details,
        in Security::UtcT                      expiry_time
    );

    void get_Confidentiality_mech (
        in long                                confidentiality_mech,
        in string
        confidentiality_mech_name,
        inout Security::Opaque                 parameters,
        inout Security::Opaque                 remote_parameters,
        inout boolean                          Standard_mechanism,
        inout Security::Opaque                 interface_details,
        inout Security::UtcT                   expiry_time
    );

    void query_Confidentiality_mech (
        inout sequence<long>                   confidentiality_mech,
        inout sequence<string>                 confidentiality_mech_name,
        inout sequence<Security::Opaque>       parameters,
        inout sequence<Security::Opaque>       remote_parameters,
        inout sequence<boolean>                Standard_mechanism,
        inout sequence<Security::Opaque>       interface_details,
        inout sequence<Security::UtcT>         expiry_time
    );

    void delete_Confidentiality_mech (
        in long                                confidentiality_mech
    );

};

//Interface AuthPolicy
interface AuthPolicy {

    readonly attribute Security::PolicyType    policy_type;

```

```
void set_Auth_policy (
    in long                                policy_id,
    in Security::PolicyType                type,
    in Security::PrincipalType              principal_type,
    in string                              security_name,
    in Security::UtcT                      expiry_time,
    in Security::AuthenticationMethod       method,
    in Security::Opaque                    auth_data,
    in Security::AttributeList              privileges
);

void get_Auth_policy (
    in long                                policy_id,
    in Security::PolicyType                type,
    in Security::PrincipalType              principal_type,
    in string                              security_name,
    out Security::UtcT                     expiry_time,
    out Security::AuthenticationMethod       method,
    out Security::Opaque                    auth_data,
    out Security::AttributeList              privileges
);

void query_Auth_policy (
    inout sequence<long>                   policy_id,
    inout sequence<Security::PolicyType>   type,
    inout sequence<Security::PrincipalType> principal_type,
    inout sequence<string>                  security_name,
    inout sequence<Security::UtcT>          expiry_time,
    inout sequence<Security::AuthenticationMethod> method,
    inout sequence<Security::Opaque>        auth_data,
    inout sequence<Security::AttributeList> privileges
);

void update_Auth_policy (
    in long                                policy_id,
    inout Security::PolicyType              type,
    inout Security::PrincipalType            principal_type,
    inout string                            security_name,
    inout Security::UtcT                    expiry_time,
    inout Security::AuthenticationMethod     method,
    inout Security::Opaque                  auth_data,
    inout Security::AttributeList             privileges
);

void delete_Auth_policy (
    in long                                policy_id
);

};
```

```

//Interface AuthMechanism
interface AuthMechanism {

    readonly attribute Security::PolicyType        policy_type;

    void set_Auth_mech (
        in Security::AuthenticationMethod        method,
        in string                                mech_name,
        in Security::Opaque                       parameters,
        in Security::Opaque                       remote_parameters,
        in boolean                                standard_mechanism,
        in Security::Opaque                       interface_details,
        in Security::UtcT                         expiry_time
    );

    void get_Auth_mech (
        in Security::AuthenticationMethod method,
        in string                                mech_name,
        inout Security::Opaque                  parameters,
        inout Security::Opaque                  remote_parameters,
        inout boolean                           Standard_mechanism,
        inout Security::Opaque                  interface_details,
        inout Security::UtcT                    expiry_time
    );

    void query_Auth_mech (
        inout sequence<Security::AuthenticationMethod> method,
        inout sequence<string>                        mech_name,
        inout sequence<Security::Opaque>              parameters,
        inout sequence<Security::Opaque>              remote_parameters,
        inout sequence<boolean>                       Standard_mechanism,
        inout sequence<Security::Opaque>              interface_details,
        inout sequence<Security::UtcT>                expiry_time
    );

    void delete_Auth_mech (
        inout Security::AuthenticationMethod        method
    );
};

//Interface DelegationPolicy
// The get/set_delegation_mode operations are taken from the original
// Delegation Policy. The query and get/set_control operations are
// newly defined.
interface DelegationPolicy {

    readonly attribute Security::PolicyType        policy_type;

```



```

void set_delegation_mode (
    in long                                policy_id,
    in Security::InterfaceDefInfo          object_type,
    in Security::DelegationMode            mode
);

Security::DelegationMode get_delegation_mode (
    in long                                policy_id,
    in Security::InterfaceDefInfo          object_type,
    out Security::DelegationMode          mode
);

void query_delegation_mode (
    inout sequence<long>                   policy_id,
    inout sequence<Security::InterfaceDefInfo> object_type,
    inout sequence<Security::DelegationMode> mode
);

void update_delegation_mode (
    in long                                policy_id,
    inout Security::InterfaceDefInfo        object_type,
    inout Security::DelegationMode          mode
);

```

//set_controls is used to specify restrictions on where and when
//attributes/credentials can be delegated/used. object_type specifies
//the object delegating. force_commit, if true, means that the
//restrictions should be applied immediately. required_attributes
//identifies the attributes the intermediate/target object should
//have so that this client can use a delegation_mode before the
//specified expiry_time. privileges_delegated lists the
//privileges that can be delegated (in a composite only some
//might be delegated), while no_of_invocations specifies the
//maximum number of delegations allowed. The out parameters
//specify error messages if the method fails.

```

boolean set_controls (
    in long                                policy_id,
    in Security::InterfaceDefInfo          object_type,
    in boolean                             force_commit,
    in Security::AttributeList             required_attributes,
    in Security::DelegationMode            delegation_mode,
    in Security::UtcT                      expiry_time,
    in Security::AttributeList             privileges_delegated,
    in long                                no_of_invocations
);

```

//get_controls will return the restriction controls for the
 //initiating object "object_type" or for a target object with the
 //specified required_attributes.

```

    boolean get_controls (
        in long
        in Security::InterfaceDefInfo
        in Security::AttributeList
        out boolean
        out Security::DelegationMode
        out Security::UtcT
        out Security::AttributeList
        out long
        policy_id,
        object_type,
        required_attributes,
        force_commit,
        delegation_mode,
        expiry_time,
        privileges_delegated,
        no_of_invocations
    );
    
```

```

    boolean query_controls (
        inout sequence<long>
        inout sequence<Security::InterfaceDefInfo>
        inout sequence<Security::AttributeList>
        inout sequence<boolean>
        inout sequence<Security::DelegationMode>
        inout sequence<Security::UtcT>
        inout sequence<Security::AttributeList>
        inout sequence<long>
        policy_id,
        object_type,
        required_attributes,
        force_commit,
        delegation_mode,
        expiry_time,
        privileges_delegated,
        no_of_invocations
    );
    
```

```

    boolean update_controls (
        in long
        inout Security::InterfaceDefInfo
        inout Security::AttributeList
        inout boolean
        inout Security::DelegationMode
        inout Security::UtcT
        inout Security::AttributeList
        inout long
        policy_id,
        object_type,
        required_attributes,
        force_commit,
        delegation_mode,
        expiry_time,
        privileges_delegated,
        no_of_invocations
    );
    
```

```

    boolean remove_controls (
        in long
        inout Security::InterfaceDefInfo
        policy_id,
        object_type,
    );
    
```

};

// The operations used in ACCESSPOLICY below are taken from the
 // original AccessPolicy and DomainAccessPolicy. The operation names
 // have been preserved for compatability i.e. they are not using the
 // usual get/set/query names.

```
//Interface AccessPolicy
interface AccessPolicy {
```

```
    readonly attribute Security::PolicyType        policy_type;

    Security::RightsList get_effective_rights(
        in securitylevel2::CredentialsList        cred_list,
        in Security::ExtensibleFamily              rights_family
    );

    void grant_rights (
        in Security::AccessMethod                  method,
        in Security::Attribute                     priv_attr,
        in Security::DelegationState               del_state,
        in Security::ExtensibleFamily              rights_family,
        in Security::RightsList                    rights
    );

    void revoke_rights (
        in Security::AccessMethod                  method,
        in Security::Attribute                     priv_attr,
        in Security::DelegationState               del_state,
        in Security::ExtensibleFamily              rights_family,
        in Security::RightsList                    rights
    );

    void replace_rights (
        in Security::AccessMethod                  method,
        in Security::Attribute                     priv_attr,
        in Security::DelegationState               del_state,
        in Security::ExtensibleFamily              rights_family,
        in Security::RightsList                    rights
    );

    Security::RightsList get_rights (
        in Security::AccessMethod                  method,
        in Security::Attribute                     priv_attr,
        in Security::DelegationState               del_state,
        in Security::ExtensibleFamily              rights_family
    );

    void query_rights (
        inout sequence<Security::AccessMethod>    method,
        inout sequence<Security::Attribute>        priv_attr,
        inout sequence<Security::DelegationState> del_state,
        inout sequence<Security::ExtensibleFamily> rights_family,
        inout sequence<Security::RightsList>       rights
    );
```

```

};

//Interface AccessMechanism
interface AccessMechanism {

    readonly attribute Security::PolicyType        policy_type;

    void set_Access_mech (
        in Security::AccessMethod                method,
        in string                                mech_name,
        in Security::Opaque                       parameters,
        in Security::Opaque                       remote_parameters,
        in boolean                                Standard_mechanism,
        in Security::Opaque                       interface_details,
        in Security::UtcT                         expiry_time
    );

    void get_Access_mech (
        in Security::AccessMethod                method,
        in string                                mech_name,
        inout Security::Opaque                   parameters,
        inout Security::Opaque                   remote_parameters,
        inout boolean                            Standard_mechanism,
        inout Security::Opaque                   interface_details,
        inout Security::UtcT                     expiry_time
    );

    void query_Access_mech(
        inout sequence<Security::AccessMethod>    method,
        inout sequence<string>                    mech_name,
        inout sequence<Security::Opaque>          parameters,
        inout sequence<Security::Opaque>          remote_parameters,
        inout sequence<boolean>                   Standard_mechanism,
        inout sequence<Security::Opaque>          interface_details,
        inout sequence<Security::UtcT>            expiry_time
    );

    void delete_Access_mech(
        in Security::AccessMethod                method
    );

};

```

```

//Interface AuditPolicy
interface AuditPolicy {

```

```

    readonly attribute Security::PolicyType        policy_type;

```

```
void set_audit_selectors (
    in long
    in unsigned long
    in Security::UtcT
    in Security::TimeT
    in boolean
    in Security::Opaque
    in InterfaceDef
    in Security::AuditEventTypeList
    in Security::SelectorValueList
    in Object
    in Object
    in Security::authorityDescriptor
    policy_id,
    policy_version,
    expiry_time,
    effective_time,
    revoked,
    revocation_details,
    object_type,
    events,
    selectors,
    audit_analyser,
    audit_responder,
    accepted_authorities
);
```

```
void clear_audit_selectors (
    in long
    in unsigned long
    in Security::UtcT
    in Security::TimeT
    in boolean
    in Security::Opaque
    in InterfaceDef
    in Security::AuditEventTypeList
    policy_id,
    policy_version,
    expiry_time,
    effective_time,
    revoked,
    revocation_details,
    object_type,
    events,
);
```

```
void replace_audit_selectors (
    in InterfaceDef
    in Security::AuditEventTypeList
    in Security::SelectorValueList
    in Object
    in Object
    in Security::authorityDescriptor
    object_type,
    events,
    selectors,
    audit_analyser,
    audit_responder,
    accepted_authorities
);
```

```
Security::SelectorValueList get_audit_selectors (
    inout long
    inout unsigned long
    inout Security::UtcT
    inout Security::TimeT
    inout boolean
    inout Security::Opaque
    inout InterfaceDef
    inout Security::AuditEventTypeList
    out Security::SelectorValueList
    out Object
    out Object
    policy_id,
    policy_version,
    expiry_time,
    effective_time,
    revoked,
    revocation_details,
    object_type,
    events,
    selectors,
    audit_analyser,
    audit_responder,
);
```

```

        out Security::authorityDescriptor    accepted_authorities
    );

    boolean set_audit_channel (
        in SecurityLevel2::AuditChannel    audit_channel,
        in Object                            response_event
    );

};

//Interface AuditMechanism
interface AuditMechanism {

    readonly attribute Security::PolicyType    policy_type;

//Audit_mechanism operations
    void set_audit_mech (
        in Security::MechanismType          method,
        in string                            mechanism_type,
        in Security::Opaque                  parameters,
        in Security::Opaque                  remote_parameters,
        in boolean                           Standard_mechanism,
        in Security::Opaque                  interface_details,
        in Security::UtcT                    expiry_time
    );

    void get_audit_mech (
        in Security::MechanismType          method,
        in string                            mechanism_type,
        inout Security::Opaque               parameters,
        inout Security::Opaque               remote_parameters,
        inout boolean                        Standard_mechanism,
        inout Security::Opaque               interface_details,
        inout Security::UtcT                 expiry_time
    );

    void query_audit_mech(
        inout sequence<Security::MechanismType>    method,
        inout sequence<string>                      mechanism_type,
        inout sequence<Security::Opaque>            parameters,
        inout sequence<Security::Opaque>            remote_parameters,
        inout sequence<boolean>                     Standard_mechanism,
        inout sequence<Security::Opaque>            interface_details,
        inout sequence<Security::UtcT>              expiry_time
    );

    void delete_audit_mech(
        inout Security::MechanismType              method

```

```

);

void set_audit_authority (
    in Security::authorityDescriptor    authority,
    in Security::Opaque                 parameters,
    in boolean                           Standard_mechanism,
    in Security::Opaque                 interface_details
);

void get_audit_authority (
    in Security::authorityDescriptor    authority,
    inout Security::Opaque              parameters,
    inout boolean                       Standard_mechanism,
    inout Security::Opaque              interface_details
);

void query_audit_authority(
    inout sequence<Security::authorityDescriptor> authority,
    inout sequence<Security::Opaque>              parameters,
    inout sequence<boolean>                       Standard_mechanism,
    inout sequence<Security::Opaque>              interface_details
);

void remove_audit_authority(
    inout Security::authorityDescriptor    authority
);

};

//Interface NRPolicy
interface NRPolicy {

    readonly attribute Security::PolicyType    policy_type;

    void set_NR_policy_info (
        in Security::ExtensibleFamily          NR_policy_id,
        in unsigned long                       policy_version,
        in Security::InterfaceDefInfo          object_type,
        in Security::TimeT                     policy_effective_time,
        in Security::TimeT                     policy_expiry_time,
        in boolean                             revoked,
        in Security::Opaque                    revocation_details,
        in Security::evidenceDescriptorList    supported_evidence_types,
        in Security::mechanismDescriptorList   supported_mechanisms,
        in Security::authorityDescriptorList   accepted_authorities
    );

    void get_NR_policy_info (
        out Security::ExtensibleFamily          NR_policy_id,

```

```

out unsigned long                policy_version,
out Security::InterfaceDefInfo    object_type,
out Security::TimeT              policy_effective_time,
out Security::TimeT              policy_expiry_time,
out boolean                      revoked,
out Security::Opaque             revocation_details,
out Security::evidenceDescriptorList supported_evidence_types,
out Security::mechanismDescriptorList supported_mechanisms,
out Security::authorityDescriptorList accepted_authorities
);

```

```

void query_NR_policy_info (
    inout sequence<Security::ExtensibleFamily>    NR_policy_id,
    inout sequence<unsigned long>                policy_version,
    inout sequence<Security::InterfaceDefInfo>    object_type,
    inout sequence<Security::TimeT>              policy_effective_time,
    inout sequence<Security::TimeT>              policy_expiry_time,
    inout sequence<boolean>                      revoked,
    inout sequence<Security::Opaque>             revocation_details,
    inout sequence<Security::evidenceDescriptorList> supported_evidence_types,
    inout sequence<Security::mechanismDescriptorList> supported_mechanisms,
    inout sequence<Security::authorityDescriptorList> accepted_authorities
);

```

```

void update_NR_policy_info (
    in Security::ExtensibleFamily    NR_policy_id,
    inout unsigned long              policy_version,
    inout Security::InterfaceDefInfo object_type,
    inout Security::TimeT            policy_effective_time,
    inout Security::TimeT            policy_expiry_time,
    inout boolean                    revoked,
    inout Security::Opaque            revocation_details,
    inout Security::evidenceDescriptorList supported_evidence_types,
    inout Security::mechanismDescriptorList supported_mechanisms,
    inout Security::authorityDescriptorList accepted_authorities
);

```

```

void delete_NR_policy_info (
    in Security::ExtensibleFamily    NR_policy_id,
    in unsigned long                 policy_version
);

```

```

};

```



```

//Interface NRMechanism
interface NRMechanism {

    readonly attribute Security::PolicyType    policy_type;

//NR_mechanism operations
    void set_NR_mech (
        in Security::NRmech
        in string
        in Security::Opaque
        in Security::Opaque
        in boolean
        in Security::Opaque
        in Security::UtcT
    );

    void get_NR_mech (
        in Security::NRmech
        inout string
        inout Security::Opaque
        inout Security::Opaque
        inout boolean
        inout Security::Opaque
        inout Security::UtcT
    );

    void query_NR_mech(
        inout sequence<Security::NRmech>
        inout sequence<string>
        inout sequence<Security::Opaque>
        inout sequence<Security::Opaque>
        inout sequence<boolean>
        inout sequence<Security::Opaque>
        inout sequence<Security::UtcT>
    );

    void delete_NR_mech(
        inout Security::NRmech
    );

// Authority operations
// authorityDescriptor holds Name, Role and
// Last__revocation_check_offset
    void set_NR_authority (
        in Security::authorityDescriptor
        in Security::Opaque
        in boolean
        authority,
        parameters,
        Standard_mechanism,

```

```

        in Security::Opaque                interface_details
    );

    void get_NR_authority (
        in Security::authorityDescriptor  authority,
        inout Security::Opaque            parameters,
        inout boolean                      Standard_mechanism,
        inout Security::Opaque            interface_details
    );

    void query_NR_authority(
        inout sequence<Security::authorityDescriptor>  authority,
        inout sequence<Security::Opaque>                parameters,
        inout sequence<boolean>                          Standard_mechanism,
        inout sequence<Security::Opaque>                interface_details
    );

    void delete_NR_authority(
        in Security::authorityDescriptor  authority
    );

// Evidence operations
    void set_NR_evidence (
        in string                      evidence_name,
        in Security::EvidenceType      evidence_type,
        in Security::duration_in_minutes evidence_validity_duration,
        in boolean                     must_use_trusted_time,
        in Security::UtcT              date_on_system,
        in Security::Opaque            parameters,
        in boolean                     Standard_mechanism,
        in Security::Opaque            interface_details
    );

    void get_NR_evidence (
        in string                      evidence_name,
        inout Security::EvidenceType   evidence_type,
        inout Security::duration_in_minutes evidence_validity_duration,
        inout boolean                  must_use_trusted_time,
        inout Security::UtcT           date_on_system,
        inout Security::Opaque         parameters,
        inout boolean                  Standard_mechanism,
        inout Security::Opaque         interface_details
    );

    void query_NR_evidence(
        inout sequence<string>          evidence_name,
        inout sequence<Security::EvidenceType> evidence_type,
        inout sequence<Security::duration_in_minutes>

```

```

        inout sequence<boolean>          evidence_validity_duration,
        inout sequence<Security::UtcT>    must_use_trusted_time,
        inout sequence<Security::Opaque>  date_on_system,
        inout sequence<boolean>          parameters,
        inout sequence<Security::Opaque>  Standard_mechanism,
                                          interface_details
    );

    void delete_NR_evidence(
        in string                      evidence_name,
        in Security::EvidenceType      evidence_type
    );
};

// Interface SecureInvocationPolicy
interface SecureInvocationPolicy {

    readonly attribute Security::PolicyType    policy_type;

    void set_interop_policy (
        in long                      interop_policy_id,
        in Security::InteropPolicyType interop_policy_type,
        in Security::InterfaceDefInfo  object_type,
        in Security::Opaque            domain_id,
        // Authentication segment
        in Security::MechRequiresSupports auth_mech,
        in Security::AuthPolicyRequiresSupports auth_policy_config,
        //AuthenticationType
        in long                      auth_mapping,
        // Access segment
        in Security::MechRequiresSupports access_mech,
        in Security::SecureInvocationFamily access_Type_policy_config,
        //Type=Rights(get,set,manage;etc), Capability,...
        in Security::SecureInvocationFamily access_Attribute_policy_config,
        //Role, Public,...
        in long                      access_Type_mapping,
        in long                      access_Attribute_mapping,
        //Delegation segment
        in Security::DelegationPolicyRequiresSupports
        delegation_policy_config,
        in long                      delegation_mode_mapping,
        // QoP segment
        in Security::MechRequiresSupports qop_mech,
        in Security::QOPPPolicyRequiresSupports qop_policy_config,
        in long                      qop_type_mapping,
        in long                      msg_part_mapping,

```

```

// NR segment
    in Security::MechRequiresSupports      nr_mech,
    in Security::NRPolicyRequiresSupports  nr_policy_config,
    in long                                nr_evidence_mapping,
// Audit segment
    in Security::MechRequiresSupports      audit_mech,
    in Security::AuditPolicyRequiresSupports audit_policy_config,
    in long                                audit_event_mapping,
    in long                                audit_selector_mapping,
// The date the mapping is set is automatically set by the ORB.
// It takes the current date.
    in Security::authorityDescriptor      authority,
    in Security::UtcT                     expiry_time
);

void get_interop_policy (
    in long                                interop_policy_id,
    out Security::InteropPolicyType        interop_policy_type,
    out Security::InterfaceDefInfo         object_type,
    out Security::Opaque                   domain_id,
    out Security::MechRequiresSupports     auth_mech,
    out Security::AuthPolicyRequiresSupports
auth_policy_config,
    out long                                auth_mapping,
    out Security::MechRequiresSupports     access_mech,
    out Security::SecureInvocationFamily
access_Type_policy_config,
    out Security::SecureInvocationFamily
access_Attribute_policy_config,
    out long                                access_Type_mapping,
    out long                                access_Attribute_mapping,
    out Security::DelegationPolicyRequiresSupports
delegation_policy_config,
    out long                                delegation_mode_mapping,
    out Security::MechRequiresSupports     qop_mech,
    out Security::QOPPolicyRequiresSupports qop_policy_config,
    out long                                qop_type_mapping,
    out long                                msg_part_mapping,
    out Security::MechRequiresSupports     nr_mech,
    out Security::NRPolicyRequiresSupports nr_policy_config,
    out long                                nr_evidence_mapping,
    out Security::MechRequiresSupports     audit_mech,
    out Security::AuditPolicyRequiresSupports audit_policy_config,
    out long                                audit_event_mapping,
    out long                                audit_selector_mapping,
    out Security::authorityDescriptor      authority,
    out Security::UtcT                     expiry_time
);

```

```

void query_interop_policy (
    inout sequence<long>                Interop_policy_id,
    inout sequence<Security::InteropPolicyType> interop_policy_type,
    inout sequence<Security::InterfaceDefInfo> object_type,
    inout sequence<Security::Opaque>      domain_id,
    inout sequence<Security::MechRequiresSupports> auth_mech,
    inout sequence<Security::AuthPolicyRequiresSupports>
        auth_policy_config,
    inout sequence<long>                auth_mapping,
    inout sequence<Security::MechRequiresSupports>
        access_mech,
    inout sequence<Security::SecureInvocationFamily>
        access_Type_policy_config,
    inout sequence<Security::SecureInvocationFamily>
        access_Attribute_policy_config,
    inout sequence<long>                access_Type_mapping,
    inout sequence<long>                access_Attribute_mapping,
    inout sequence<Security::DelegationPolicyRequiresSupports>
        delegation_policy_config,
    inout sequence<long>                delegation_mode_mapping,
    inout sequence<Security::MechRequiresSupports> qop_mech,
    inout sequence<Security::QOPPolicyRequiresSupports>
        qop_policy_config,
    inout sequence<long>                qop_type_mapping,
    inout sequence<long>                msg_part_mapping,
    inout sequence<Security::MechRequiresSupports> nr_mech,
    inout sequence<Security::NRPolicyRequiresSupports>
        nr_policy_config,
    inout sequence<long>                nr_evidence_mapping,
    inout sequence<Security::MechRequiresSupports>
        audit_mech,
    inout sequence<Security::AuditPolicyRequiresSupports>
        audit_policy_config,
    inout sequence<long>                audit_event_mapping,
    inout sequence<long>                audit_selector_mapping,
    inout sequence<Security::authorityDescriptor> authority,
    inout sequence<Security::UtcT>      expiry_time
);

```

```

void update_interop_policy (
    in long                interop_policy_id,
    inout Security::InteropPolicyType interop_policy_type,
    inout Security::InterfaceDefInfo object_type,
    inout Security::Opaque domain_id,
    inout Security::MechRequiresSupports auth_mech,
    inout Security::AuthPolicyRequiresSupports auth_policy_config,
    inout long                auth_mapping,

```

```

    inout Security::MechRequiresSupports    access_mech,
    inout Security::SecureInvocationFamily
                                         access_Type_policy_config,
    inout Security::SecureInvocationFamily
                                         access_Attribute_policy_config,
    inout long                             access_Type_mapping
    inout long                             access_Attribute_mapping,
    inout Security::DelegationPolicyRequiresSupports
                                         delegation_policy_config,
    inout long                             delegation_mode_mapping,
    inout Security::MechRequiresSupports    qop_mech,
    inout Security::QOPPolicyRequiresSupports qop_policy_config,
    inout long                             qop_type_mapping,
    inout long                             msg_part_mapping,
    inout Security::MechRequiresSupports    nr_mech,
    inout Security::NRPolicyRequiresSupports nr_policy_config,
    inout long                             nr_evidence_mapping,
    inout Security::AuditPolicyRequiresSupports audit_policy_config,
    inout long                             audit_event_mapping,
    inout long                             audit_selector_mapping,
    inout Security::authorityDescriptor    authority,
    inout Security::UtcT                    expiry_time
);

void delete_interop_policy (
    in long
    inout Security::InteropPolicyType
    inout Security::InterfaceDefInfo
    inout Security::Opaque
    interop_policy_id,
    interop_policy_type,
    object_type,
    domain_id,
);

};

interface MappingLookup {

    void query (
        in Security::ServiceType
        in Security::Constraint
        in Security::Preference
        in unsigned long
        out Security::MappingSeq
        out MappingIterator
        type,
        constr,
        pref,
        how_many,
        maps,
        map_itr
    );

};

interface DomainMapping {
    Security::MappingId add (

```

```

        in Security::ServiceType
        in string
        in unsigned short
        in unsigned short
        in sequence<octet>
        in sequence<octet>
        in unsigned short
        in unsigned short
        in sequence<unsigned >
        in sequence<octet>
        in sequence<Security::domain_values> mapped_values,
        in Security::Opaque
        in Security::Opaque
        out Security::UtcT
        serviceType,
        policyClassification,
        family_definer1,
        family_id1,
        attribute_type1,
        defining_authority1,
        family_definer2,
        family_id2,
        attribute_type2,
        defining_authority2,
        remoteDomainId,
        remoteDomainAuthority,
        timeStamp
    );

    void withdraw (
        in Security::MappingId
    );

    Security::MappingSeq describe (
        in Security::MappingId
    );

    void modify (
        in Security::MappingId
        in Security::MappingIdSeq
        in Security::MappingSeq
        id,
        del_list,
        modify_list
    );

    void list(
        in unsigned long
        out Security::MappingSeq
        out MappingIterator
        how_many,
        ids,
        id_itr
    );

};

interface MappingIterator {

    unsigned long max_left ();

    boolean next_n (
        in unsigned long
        out Security::MappingSeq
        n,
        maps
    );

```

```

    void destroy ();

};

// END OF SECURITYADMIN MODULE
};

// *****
// *      Module:      SECIOP      *
// *      Function:    Secure Inter-ORB protocol      *
// *****

module SECIOP {

    typedef sequence <octet> Opaque;

    const IOP::ComponentID TAG_GENERIC_SEC_MECH = 12;

    const IOP::ComponentID TAG_ASSOCIATION_OPTIONS = 13;

    const IOP::ComponentID TAG_SEC_NAME = 14;

    const IOP::ComponentID TAG_ACCESS_CONTROL = 15;

    const IOP::ComponentID TAG_AUDIT = 16;

    const IOP::ComponentID TAG_NON_REPUDIATION=17;

    const IOP::ComponentID TAG_SSL_SEC_TRANS=18;

    struct AssociationOptions{
        Security::AssociationOptions    target_supports;
        Security::AssociationOptions    target_requires;
        Security::MsgPartList           msg_part_supported;
        Security::MsgPartList           msg_part_required;
        sequence <TaggedComponent>     integ_mechs_supported;
        sequence <TaggedComponent>     integ_mechs_required;
        sequence <TaggedComponent>     conf_mechs_supported;
        sequence <TaggedComponent>     conf_mechs_required;
    }

    struct GenericMechanismInfo {
        sequence <octet>                security_mechanism_type;
        sequence <octet>                mech_specific_data;
    }
}

```



```

sequence < IOP::TaggedComponent>      components;
};

struct AccessControl {
    sequence< Security::SecureInvocationFamily>  Operation_supports;
    Security::SecureInvocationFamily             Operation_requires;
    sequence< Security::SecureInvocationFamily>  Attribute_supports;
    Security::SecureInvocationFamily             Attribute_requires;
    sequence < Security::DelegationMode>         Delegation_supports;
    Security::DelegationMode                     Delegation_requires;
    sequence <TaggedComponent>                   access_mechs_supported;
    sequence <TaggedComponent>                   access_mechs_required;
};

struct Audit {
    Security::AuditPolicyRequiresSupports        Audit_policy;
    sequence <TaggedComponent>                   Audit_mechs_supported;
    sequence <TaggedComponent>                   Audit_mechs_required;
};

struct NonRepudiation {
    Security::NRPolicyRequiresSupports           NR_policy;
    sequence <TaggedComponent>                   NR_mechs_supported;
    sequence <TaggedComponent>                   NR_mechs_required;
};

struct SSL{
    Security::AssociationOptions                 target_supports;
    Security::AssociationOptions                 target_requires;
    unsigned short                              port;
}

// prefix with MT (as in Serv_idl.idl) so that it does not conflict with the struct names
enum MsgType {
    MTEstablishContext,
    MTCompleteEstablishContext,
    MTContinueEstablishContext,
    MTDiscardContext,
    MTMessageError,
    MTMessageInContext
};

struct ulonglong {
    unsigned long low;
    unsigned long high;
};

```

Appendix A: IDL for Comprehensive CORBA Security

```
};

typedef unsigned long ContextId;

enum ContextIdDefn {
    Client,
    Peer,
    Sender
};

struct EstablishContext {
    ContextId
    sequence <octet>
};
client_context_id;
initial_context_token;

struct CompleteEstablishContext {
    ContextId
    boolean
    ContextId
    sequence <octet>
};
client_context_id;
target_context_id_valid;
target_context_id;
final_context_token;

struct ContinueEstablishContext {
    ContextId
    sequence <octet>
};
client_context_id;
continuation_context_token;

struct DiscardContext {
    ContextIdDefn
    ContextId
};
message_context_id_defn;
message_context_id;

struct MessageError {
    ContextIdDefn
    ContextId
    long
    long
};
message_context_id_defn;
message_context_id;
major_status;
minor_status;

struct MessageInContext {
    ContextIdDefn
    ContextId
    Sequence<octet>
};
message_context_id_defn;
message_context_id;
message_protection_token;

// END OF SECIOP MODULE
};
```

Appendix B - IDL for Security-Aware Trader Service

This appendix will present the IDL for the Security-Aware Trader Service.

Module Structure

The modules used in the IDL are now:

- **CosTrading** : Security-Aware Trading Module
- **CosTradingDynamic**: Trading Dynamic Property Module
- **CosTradingRepos**: Trading Service Type Repository Module

The module structure has been preserved. The CORBA 2.0 object interfaces are still used, but some of the parameter lists are extended and a new security policy interface has been added.

Object Interfaces

Comments through out the IDL code will explain the modifications to and addition of new object interfaces. The new IDL code will be highlighted in **bold**.

IDL

```
#include <orb.idl>

// IDL for Security-Aware Trading Function Module

module CosTrading {

// forward references to our interfaces

interface Lookup;
interface Register;
interface Link;
interface Proxy;
interface Admin;
interface OfferIterator;
interface OfferIdIterator;

// type definitions used in more than one interface

typedef string Istring;
typedef Object TypeRepository;

typedef Istring PropertyName;
typedef sequence<PropertyName> PropertyNameSeq;
typedef any PropertyValue;

struct Property {
    PropertyName name;
    PropertyValue value;
};
typedef sequence<Property> PropertySeq;

struct Offer {
    Object reference;
    PropertySeq properties;
};
typedef sequence<Offer> OfferSeq;

typedef string OfferId;
typedef sequence<OfferId> OfferIdSeq;

typedef Istring ServiceTypeName;

typedef Istring Constraint;

enum FollowOption {
    local_only,
```

```

        if_no_local,
        always
    };

typedef Istring LinkName;
typedef sequence<LinkName> LinkNameSeq;
typedef LinkNameSeq TraderName;

typedef string PolicyName;
typedef sequence<PolicyName> PolicyNameSeq;
typedef any PolicyValue;

struct Policy {
    PolicyName name;
    PolicyValue value;
};
typedef sequence<Policy> PolicySeq;

// exceptions used in more than one interface

exception IllegalTraderAccess {}; // Security-Aware Trader exception

exception IllegalServiceOfferAccess{};//Security-AwareTraderexception

exception UnknownMaxLeft {};

exception NotImplemented {};

exception IllegalServiceType {
    ServiceTypeName type;
};

exception UnknownServiceType {
    ServiceTypeName type;
};

exception IllegalPropertyName {
    PropertyName name;
};

exception DuplicatePropertyName {
    PropertyName name;
};

exception PropertyTypeMismatch {
    ServiceTypeName type;
    Property prop;
};

```

```
exception MissingMandatoryProperty {
    ServiceTypeName type;
    PropertyName name;
};

exception ReadonlyDynamicProperty {
    ServiceTypeName type;
    PropertyName name;
};

exception IllegalConstraint {
    Constraint constr;
};

exception InvalidLookupRef {
    Lookup target;
};

exception IllegalOfferId {
    OfferId id;
};

exception UnknownOfferId {
    OfferId id;
};

exception DuplicatePolicyName {
    PolicyName name;
};

// the interfaces

interface TraderComponents {

    readonly attribute Lookup lookup_if;
    readonly attribute Register register_if;
    readonly attribute Link link_if;
    readonly attribute Proxy proxy_if;
    readonly attribute Admin admin_if;
};

// Security-Aware Trader Attributes
interface SecurityAttributes {
    readonly attribute boolean Security_Aware;
    readonly attribute boolean access_control_trader;
    readonly attribute boolean access_control_service_offers;
    readonly attribute boolean encrypt_stores;
```

```
    readonly attribute boolean encrypt_comms;
    readonly attribute boolean integrity_check_stores;
    readonly attribute boolean integrity_check_comms;
    readonly attribute boolean nr_trade;
    readonly attribute boolean audit_trade;
};

interface SupportAttributes {
    readonly attribute boolean supports_modifiable_properties;
    readonly attribute boolean supports_dynamic_properties;
    readonly attribute boolean supports_proxy_offers;
    readonly attribute TypeRepository type_repos;
};

interface ImportAttributes {
    readonly attribute unsigned long def_search_card;
    readonly attribute unsigned long max_search_card;
    readonly attribute unsigned long def_match_card;
    readonly attribute unsigned long max_match_card;
    readonly attribute unsigned long def_return_card;
    readonly attribute unsigned long max_return_card;
    readonly attribute unsigned long max_list;
    readonly attribute unsigned long def_hop_count;
    readonly attribute unsigned long max_hop_count;
    readonly attribute FollowOption def_follow_policy;
    readonly attribute FollowOption max_follow_policy;
};

interface LinkAttributes {
    readonly attribute FollowOption max_link_follow_policy;
};

interface Lookup: TraderComponents, SecurityAttributes, SupportAttributes {
    typedef Istring Preference;

    enum HowManyProps { none, some, all };

    union SpecifiedProps switch ( HowManyProps ) {
        case some: PropertyNameSeq prop_names;
    };

    exception IllegalPreference {
        Preference pref;
    };

    exception IllegalPolicyName {
```

```

PolicyName name;
};

exception PolicyTypeMismatch {
    Policy the_policy;
};

exception InvalidPolicyValue {
    Policy the_policy;
};

void query (
    in ServiceTypeName type,
    in Constraint constr,
    in Preference pref,
    in PolicySeq policies,
    in SpecifiedProps desired_props,
    in unsigned long how_many,
    out OfferSeq offers,
    out OfferIterator offer_itr,
    out PolicyNameSeq limits_applied
) raises (
    IllegalTraderAccess,//Security-Aware Trader exception
    IllegalServiceOfferAccess,//Security-Aware Trader except.
    IllegalServiceType,
    UnknownServiceType,
    IllegalConstraint,
    IllegalPreference,
    IllegalPolicyName,
    PolicyTypeMismatch,
    InvalidPolicyValue,
    IllegalPropertyName,
    DuplicatePropertyName,
    DuplicatePolicyName
);
};

interface Register : TraderComponents,SecurityAttributes,
    SupportAttributes {

    struct OfferInfo {
        Object reference;
        ServiceTypeName type;
        PropertySeq properties;
    };

    exception InvalidObjectRef {
        Object ref;
    };

```



```

};

exception UnknownPropertyName {
    PropertyName name;
};

exception InterfaceTypeMismatch {
    ServiceTypeName type;
    Object reference;
};

exception ProxyOfferId {
    OfferId id;
};

exception MandatoryProperty {
    ServiceTypeName type;
    PropertyName name;
};

exception ReadonlyProperty {
    ServiceTypeName type;
    PropertyName name;
};

exception NoMatchingOffers {
    Constraint constr;
};

exception IllegalTraderName {
    TraderName name;
};

exception UnknownTraderName {
    TraderName name;
};

exception RegisterNotSupported {
    TraderName name;
};

OfferId export (
    in Object reference,
    in ServiceTypeName type,
    in PropertySeq properties
) raises (
    IllegalTraderAccess, // Security-Aware Trader exception
    IllegalServiceOfferAccess, // Security-Aware Trader except.

```

```
InvalidObjectRef,  
IllegalServiceType,  
UnknownServiceType,  
InterfaceTypeMismatch,  
IllegalPropertyName,  
PropertyTypeMismatch,  
ReadonlyDynamicProperty,  
MissingMandatoryProperty,  
DuplicatePropertyName  
);  
  
void withdraw (  
    in OfferId id  
) raises (  
    IllegalTraderAccess, // Security-Aware Trader exception  
    IllegalServiceOfferAccess, // Security-Aware Trader except  
    IllegalOfferId,  
    UnknownOfferId,  
    ProxyOfferId  
);  
  
OfferInfo describe (  
    in OfferId id  
) raises (  
    IllegalTraderAccess, // Security-Aware Trader exception  
    IllegalServiceOfferAccess, // Security-Aware Trader except  
    IllegalOfferId,  
    UnknownOfferId,  
    ProxyOfferId  
);  
  
void modify (  
    in OfferId id,  
    in PropertyNameSeq del_list,  
    in PropertySeq modify_list  
) raises (  
    NotImplemented,  
    IllegalTraderAccess, // Security-Aware Trader exception  
    IllegalServiceOfferAccess, // Security-Aware Trader except  
    IllegalOfferId,  
    UnknownOfferId,  
    ProxyOfferId,  
    IllegalPropertyName,  
    UnknownPropertyName,  
    PropertyTypeMismatch,  
    ReadonlyDynamicProperty,  
    MandatoryProperty,  
    ReadonlyProperty,
```

```

        DuplicatePropertyName
    );

    void withdraw_using_constraint (
        in ServiceTypeName type,
        in Constraint constr
    ) raises (
        IllegalTraderAccess, // Security-Aware Trader exception
        IllegalServiceOfferAccess, // Security-Aware Trader except
        IllegalServiceType,
        UnknownServiceType,
        IllegalConstraint,
        NoMatchingOffers
    );

    Register resolve (
        in TraderName name
    ) raises (
        IllegalTraderName,
        UnknownTraderName,
        RegisterNotSupported,
        IllegalTraderAccess, // Security-Aware Trader exception
        IllegalServiceOfferAccess // Security-Aware Trader except
        RegisterNotSupported
    );
};

interface Link : TraderComponents, SupportAttributes,
    SecurityAttributes, LinkAttributes {

    struct LinkInfo {
        Lookup target;
        Register target_reg;
        FollowOption def_pass_on_follow_rule;
        FollowOption limiting_follow_rule;
        OctetSeq Link_security;
    };

    exception IllegalLinkName {
        LinkName name;
    };
    exception UnknownLinkName {
        LinkName name;
    };

    exception DuplicateLinkName {
        LinkName name;
    };
};

```

```

};

exception DefaultFollowTooPermissive {
    FollowOption def_pass_on_follow_rule;
    FollowOption limiting_follow_rule;
};

exception LimitingFollowTooPermissive {
    FollowOption limiting_follow_rule;
    FollowOption max_link_follow_policy;
};

void add_link (
    in LinkName name,
    in Lookup target,
    in FollowOption def_pass_on_follow_rule,
    in FollowOption limiting_follow_rule
) raises (
    IllegalLinkName,
    DuplicateLinkName,
    InvalidLookupRef, // e.g. nil
    DefaultFollowTooPermissive,
    LimitingFollowTooPermissive
);

void remove_link (
    in LinkName name
) raises (
    IllegalLinkName,
    UnknownLinkName
);

LinkInfo describe_link (
    in LinkName name
) raises (
    IllegalLinkName,
    UnknownLinkName
);

LinkNameSeq list_links ();

void modify_link (
    in LinkName name,
    in FollowOption def_pass_on_follow_rule,
    in FollowOption limiting_follow_rule
) raises (
    IllegalLinkName,
    UnknownLinkName,

```

```

        DefaultFollowTooPermissive,
        LimitingFollowTooPermissive
    );

};

interface Proxy : TraderComponents, SecurityAttributes,
                SupportAttributes {

    typedef Istring ConstraintRecipe;

    struct ProxyInfo {
        ServiceTypeName type;
        Lookup target;
        PropertySeq properties;
        boolean if_match_all;
        ConstraintRecipe recipe;
        PolicySeq policies_to_pass_on;
    };

    exception IllegalRecipe _
        ConstraintRecipe recipe;
    };

    exception NotProxyOfferId {
        OfferId id;
    };

    OfferId export_proxy (
        in Lookup target,
        in ServiceTypeName type,
        in PropertySeq properties,
        in boolean if_match_all,
        in ConstraintRecipe recipe,
        in PolicySeq policies_to_pass_on
    ) raises (
        IllegalServiceType,
        UnknownServiceType,
        InvalidLookupRef, // e.g. nil
        IllegalPropertyName,
        PropertyTypeMismatch,
        ReadonlyDynamicProperty,
        MissingMandatoryProperty,
        IllegalRecipe,
        DuplicatePropertyName,
        DuplicatePolicyName
    );
};

```

```

);

void withdraw_proxy (
    in OfferId id
) raises (
    IllegalOfferId,
    UnknownOfferId,
    NotProxyOfferId
);

ProxyInfo describe_proxy (
    in OfferId id
) raises (
    IllegalOfferId,
    UnknownOfferId,
    NotProxyOfferId
);

};

interface Admin : TraderComponents, SupportAttributes,
    SecurityAttributes, ImportAttributes, LinkAttributes {

    typedef sequence<octet> OctetSeq;

    // exceptions used for the Security Attributes
    exception SecurityAttributesRequired {};

    readonly attribute OctetSeq request_id_stem;

    unsigned long set_def_search_card (in unsigned long value);
    unsigned long set_max_search_card (in unsigned long value);
    unsigned long set_def_match_card (in unsigned long value);
    unsigned long set_max_match_card (in unsigned long value);
    unsigned long set_def_return_card (in unsigned long value);
    unsigned long set_max_return_card (in unsigned long value);
    unsigned long set_max_list (in unsigned long value);
    boolean set_supports_modifiable_properties (in boolean value);
    boolean set_supports_dynamic_properties (in boolean value);
    boolean set_supports_proxy_offers (in boolean value);
    unsigned long set_def_hop_count (in unsigned long value);
    unsigned long set_max_hop_count (in unsigned long value);
    FollowOption set_def_follow_policy (in FollowOption policy);
    FollowOption set_max_follow_policy (in FollowOption policy);
    FollowOption set_max_link_follow_policy (in FollowOption policy);

```

```

// Set operations for Security Attributes
    boolean set_Security_Aware (in boolean value);
    boolean set_access_control_trader (in boolean value);
    boolean set_access_control_service_offers (in boolean value);
    boolean set_encrypt_stores (in boolean value);
    boolean set_encrypt_comms (in boolean value);
    boolean set_integrity_check_stores (in boolean value);
    boolean set_integrity_check_comms (in boolean value);
    boolean set_nr_trade (in boolean value);
    boolean set_audit_trade (in boolean value);

    TypeRepository set_type_repos (in TypeRepository repository);

    OctetSeq set_request_id_stem (in OctetSeq stem);

    void list_offers (
        in unsigned long how_many,
        out OfferIdSeq ids,
        out OfferIdIterator id_itr
    ) raises (
        NotImplemented
    );

    void list_proxies (
        in unsigned long how_many,
        out OfferIdSeq ids,
        out OfferIdIterator id_itr
    ) raises (
        NotImplemented
    );
};

interface OfferIterator {
    unsigned long max_left (
    ) raises (
        UnknownMaxLeft
    );

    boolean next_n (
        in unsigned long n,
        out OfferSeq offers
    );

    void destroy ();
};

interface OfferIdIterator {

```

```

        unsigned long max_left (
        ) raises (
            UnknownMaxLeft
        );

        boolean next_n (
        //   in unsigned long n,
        //   out OfferIdSeq ids
        );

        void destroy ();
    };

}; /* end module CosTrading */

```

// IDL for Dynamic Property Module

```

module CosTradingDynamic {

    exception DPEvalFailure {
        CosTrading::PropertyName name;
        CORBA::TypeCode returned_type;
        any extra_info;
    };

    interface DynamicPropEval {

        any evalDP (
            in CosTrading::PropertyName name,
            in CORBA::TypeCode returned_type,
            in any extra_info
        ) raises (
            DPEvalFailure
        );
    };

    struct DynamicProp {
        DynamicPropEval eval_if;
        CORBA::TypeCode returned_type;
        any extra_info;
    };
}; /* end module CosTradingDynamic */

```

// IDL for Service Type Repository Module

```

module CosTradingRepos {

```



```

interface ServiceTypeRepository {

// local types
    typedef sequence<CosTrading::ServiceTypeName>
        ServiceTypeNameSeq;
    enum PropertyMode {
        PROP_NORMAL, PROP_READONLY,
        PROP_MANDATORY, PROP_MANDATORY_READONLY
    };
    struct PropStruct {
        CosTrading::PropertyName name;
        CORBA::TypeCode value_type;
        PropertyMode mode;
    };
    typedef sequence<PropStruct> PropStructSeq;

    typedef CosTrading::Istring Identifier;
    struct IncarnationNumber {
        unsigned long high;
        unsigned long low;
    };
    struct TypeStruct {
        Identifier if_name;
        PropStructSeq props;
        ServiceTypeNameSeq super_types;
        boolean masked;
        IncarnationNumber incarnation;
    };

    enum ListOption { all, since };
    union SpecifiedServiceTypes switch ( ListOption )
    {
        case since: IncarnationNumber incarnation;
    };

// local exceptions
    exception ServiceTypeExists {
        CosTrading::ServiceTypeName name;
    };
    exception InterfaceTypeMismatch {
        CosTrading::ServiceTypeName base_service;
        Identifier base_if;
        CosTrading::ServiceTypeName derived_service;
        Identifier derived_if;
    };
    exception HasSubTypes {
        CosTrading::ServiceTypeName the_type;
    };

```

```

        CosTrading::ServiceTypeName sub_type;
    };
    exception AlreadyMasked {
        CosTrading::ServiceTypeName name;
    };
    exception NotMasked {
        CosTrading::ServiceTypeName name;
    };
    exception ValueTypeRedefinition {
        CosTrading::ServiceTypeName type_1;
        PropStruct definition_1;
        CosTrading::ServiceTypeName type_2;
        PropStruct definition_2;
    };
    exception DuplicateServiceTypeName {
        CosTrading::ServiceTypeName name;
    };

// attributes
    readonly attribute IncarnationNumber incarnation;

// operation signatures
    IncarnationNumber add_type (
        in CosTrading::ServiceTypeName name,
        in Identifier if_name,
        in PropStructSeq props,
        in ServiceTypeNameSeq super_types
    ) raises (
        CosTrading::IllegalServiceType,
        CosTrading::ServiceTypeExists,
        CosTrading::InterfaceTypeMismatch,
        CosTrading::IllegalPropertyName,
        CosTrading::DuplicatePropertyName,
        CosTrading::ValueTypeRedefinition,
        CosTrading::UnknownServiceType,
        CosTrading::DuplicateServiceTypeName
    );

    void remove_type (
        in CosTrading::ServiceTypeName name
    ) raises (
        CosTrading::IllegalServiceType,
        CosTrading::UnknownServiceType,
        CosTrading::HasSubTypes
    );

    ServiceTypeNameSeq list_types (
        in SpecifiedServiceTypes which_types

```

```
);

TypeStruct describe_type (
    in CosTrading::ServiceTypeName name
) raises (
    CosTrading::IllegalServiceType,
    CosTrading::UnknownServiceType
);

TypeStruct fully_describe_type (
    in CosTrading::ServiceTypeName name
) raises (
    CosTrading::IllegalServiceType,
    CosTrading::UnknownServiceType
);

void mask_type (
    in CosTrading::ServiceTypeName name
) raises (
    CosTrading::IllegalServiceType,
    CosTrading::UnknownServiceType,
    AlreadyMasked
);

void unmask_type (
    in CosTrading::ServiceTypeName name
) raises (
    CosTrading::IllegalServiceType,
    CosTrading::UnknownServiceType,
    NotMasked
);

};
}; /* end module CosTradingRepos */
```

Appendix C - IDL for Generic Security Service API

This appendix presents the IDL for the GSS-API server. Since GSS-API plays an integral part in the implementation of the Enhanced Security Service, a GSS-API server was built using this IDL code. It complies with the GSS-API standard.

IDL

```

//*****
//
// NAME : GSSAPI.idl
//
// DESCRIPTION: IDL for PhD demo - Operates as a GSS-API server
//              providing the required GSS-API operations, implemented
//              using cryptlib functions
//
//*****

```

```
// GSSAPI.idl
```

```
module GSSAPI {
```

```
// IDL definition of GSS-API operations.
```

```
interface GSSAPI {
```

```
// Data types used in this idl file.
```

```
typedef unsigned long OM_uint32;
```

```
typedef long gss_ctx_id_t;
```

```
typedef string gss_cred_id_t;
```

```
typedef any gss_name_t;
```

```
typedef string internalname;
```

```
typedef short obj_id;
```

```
typedef short obj_id_seq[10];
```

```
typedef long contexthandle;
```

```

typedef string credentialhandle;
typedef string octetstring;
//typedef sequence<octet> octetstring;
//typedef unsigned char ostring;
//typedef string Soctetstring;
typedef char byteBuffer[1024];

// IDL operations

// CREDENTIAL MANAGEMENT CALLS

OM_uint32 GSS_Acquire_cred(
    in internalname desired_name,
    in short lifetime_req,
    in obj_id_seq desired_mechs,
    in short cred_usage,
    out short major_status,
    out short minor_status,
    out credentialhandle output_cred_handle,
    out obj_id_seq actual_mechs,
    out short lifetime_rec
);

OM_uint32 GSS_Release_cred(
    in credentialhandle cred_handle,
    out short major_status,
    out short minor_status
);

OM_uint32 GSS_Inquire_cred(
    in credentialhandle cred_handle,
    out short major_status,
    out short minor_status,
    out internalname cred_name,
    out short lifetime_rec,
    out short cred_usage,
    out obj_id_seq mech_set
);

OM_uint32 GSS_Add_cred(
    in credentialhandle input_cred_handle,
    in internalname desired_name,
    in short initiator_time_req,
    in short acceptor_time_req,
    in obj_id desired_mech,
    inout short cred_usage,
    out short major_status,

```

```

        out short minor_status,
        out credentialhandle output_cred_handle,
        out obj_id actual_mechs,
        out short initiator_time_rec,
        out short acceptor_time_rec,
        out obj_id_seq mech_set
    );

OM_uint32 GSS_Inquire_cred_by_mech(
    in credentialhandle cred_handle,
    in obj_id mech_type,
    out short major_status,
    out short minor_status,
    out internalname cred_name,
    out short lifetime_rec_initiate,
    out short lifetime_rec_accept,
    out short cred_usage
);

```

// CONTEXT LEVEL CALLS

```

OM_uint32 GSS_Init_sec_context(
    in gss_cred_id_t claimant_cred_handle,
    in short input_context_handle,
    in internalname target_name,
    inout obj_id_seq mech_type,
    in boolean deleg_req_flag,
    in boolean mutual_req_flag,
    in boolean replay_det_req_flag,
    in boolean sequence_req_flag,
    in boolean anon_req_flag,
    in short lifetime_req,
    in octetstring chan_bindings,
    in byteBuffer input_token,
    in short tincount,
    out short major_status,
    out short minor_status,
    out contexthandle output_context_handle,
    out byteBuffer output_token,
    out short tcount,
    out boolean deleg_state,
    out boolean mutual_state,
    out boolean replay_det_state,
    out boolean sequence_state,
    out boolean anon_state,
    out boolean trans_state,
    out boolean prot_ready_state,
);

```

```
    out boolean conf_avail,  
    out boolean integ_avail,  
    out short lifetime_rec  
);
```

```
OM_uint32 GSS_Accept_sec_context(  
    in credentialhandle acceptor_cred_handle,  
    in short input_context_handle,  
    in octetstring chan_bindings,  
    in byteBuffer input_token,  
    in short tincount,  
    out short major_status,  
    out short minor_status,  
    out internalname src_name,  
    inout obj_id mech_type,  
    out contexthandle output_context_handle,  
    out boolean deleg_state,  
    out boolean mutual_state,  
    out boolean replay_det_state,  
    out boolean sequence_state,  
    out boolean anon_state,  
    out boolean trans_state,  
    out boolean prot_ready_state,  
    out boolean conf_avail,  
    out boolean integ_avail,  
    out short lifetime_rec,  
    out credentialhandle delegated_cred_handle,  
    out byteBuffer output_token,  
    out short toutcount  
);
```

```
OM_uint32 GSS_Delete_sec_context(  
    in contexthandle context_handle,  
    out short major_status,  
    out short minor_status,  
    out contexthandle output_context_token  
);
```

```
OM_uint32 GSS_Process_context_token(  
    in contexthandle context_handle,  
    out octetstring input_context_token,  
    out short major_status,  
    out short minor_status  
);
```

```
OM_uint32 GSS_Context_time(  
    in contexthandle context_handle,  
    out short major_status,  
    out short minor_status,  
    out short lifetime_rec  
);
```

```
OM_uint32 GSS_Inquire_context(  
    in short input_context_handle,  
    out short major_status,  
    out short minor_status,  
    out internalname src_name,  
    out internalname targ_name,  
    out short lifetime_rec,  
    out obj_id mech_type,  
    out boolean deleg_state,  
    out boolean mutual_state,  
    out boolean replay_det_state,  
    out boolean sequence_state,  
    out boolean anon_state,  
    out boolean trans_state,  
    out boolean prot_ready_state,  
    out boolean conf_avail,  
    out boolean integ_avail,  
    out boolean locally_initiated  
);
```

// PER-MESSAGE CALLS

```
OM_uint32 GSS_GetMIC(  
    in contexthandle context_handle,  
    in short qop_req,  
    in octetstring message,  
    out short major_status,  
    out short minor_status,  
    out byteBuffer per_msg_token,  
    out short tcount  
);
```

```
OM_uint32 GSS_VerifyMIC(  
    in contexthandle context_handle,  
    in octetstring message,  
    in byteBuffer per_msg_token,  
    in short tcount,  
    out short qop_state,  
    out short major_status,
```



```

        out short minor_status
    );

OM_uint32 GSS_Wrap(
    in contexthandle context_handle,
    in boolean conf_req_flag,
    in short qop_req,
    in octetstring input_message,
    out short major_status,
    out short minor_status,
    out boolean conf_state,
    out byteBuffer output_message,
    out short tcount
);

OM_uint32 GSS_UnWrap(
    in contexthandle context_handle,
    in byteBuffer input_message,
    in short tcount,
    out boolean conf_state,
    out short qop_state,
    out short major_status,
    out short minor_status,
    out octetstring output_message
);

// LIBRARY OPTIONS
OM_uint32 GSS_SetOptions(
    in short optiontype,
    in short optionvalue
);

OM_uint32 GSS_GetOptions(
    in short optiontype,
    out short optionvalue
);

};

}; /* end module GSSAPI */

```

Status Codes for GSS-API

This appendix also includes the status codes required for GSS-API. There are two types of status code:

- Major Status Codes: provide a mechanism-independent indication of call status;
- Minor Status Codes: provide a mechanism-specific indication of status.

Only Major Status Codes are defined in the specification, because as Minor codes are dependent on the mechanisms used.

GSS-API Major Status Codes

FATAL ERROR CODES	Code	Definition
GSS_S_BAD_BINDINGS	901	Channel bindings mismatch
GSS_S_BAD_MECH	902	Unsupported mechanism requested
GSS_S_BAD_NAME	903	Invalid name provided
GSS_S_BAD_NAME_TYPE	904	Name of unsupported type provided
GSS_S_BAD_STATUS	905	Invalid input status selector
GSS_S_BAD_SIG	906	Token had invalid integrity check
GSS_S_CONTEXT_EXPIRED	907	Specified security context expired
GSS_S_CREDENTIALS_EXPIRED	908	Expired credentials detected
GSS_S_DEFECTIVE_CREDENTIALS	909	Defective credentials detected
GSS_S_DEFECTIVE_TOKEN	910	Defective token detected
GSS_S_FAILURE	911	Failure, unspecified at GSS-API level
GSS_S_NO_CONTEXT	912	No valid security context specified
GSS_S_NO_CRED	913	No valid credentials provided
GSS_S_BAD_QOP	914	Unsupported QOP value
GSS_S_UNAUTHORIZED	915	Operation unauthorized
GSS_S_UNAVAILABLE	916	Operation unavailable
GSS_S_DUPLICATE_ELEMENT	917	Duplicate credential element requested
GSS_S_NAME_NOT_MN	918	Name contains multi-mechanism elements

INFORMATORY STATUS CODES	Codes	Definition
GSS_S_COMPLETE	801	Normal completion
GSS_S_CONTINUE_NEEDED	802	Continuation call to routine required
GSS_S_DUPLICATED_TOKEN	803	Duplicate per-message token detected
GSS_S_OLD_TOKEN	804	Timed-out per-message token detected
GSS_S_UNSEQ_TOKEN	805	Reordered (early) per-message token detected
GSS_S_GAP_TOKEN	806	Skipped predecessor token(s) detected

Appendix D - Cryptlib & Prototype information

This appendix presents an overview of *cryptlib*, a cryptography library, which was used in the implementation of the Enhanced Security System. It was used as to provide the security mechanisms, such as encryption and certificates.

Cryptlib is written by **Peter Guttman** (pgut001@cs.auckland.ac.nz), and in part by Eric Young, Colin Plumb, and others. The cryptlib manual is available at the cryptlib web site if further details are required on the product (<http://www.cs.auckland.ac.nz/~pgut001/cryptlib/>).

The *cryptlib* encryption library provides an easy-to-use interface that allows programmers add strong encryption and authentication services to their software. *cryptlib* uses several encryption, hash, MAC, public-key and digital signature mechanisms (see Table A-1 below). *cryptlib* is supplied as source code for Unix (shared or static libraries), DOS, Windows (16- and 32-bit DLL's), and the Amiga.

Algorithms

cryptlib provides a standardised interface to a number of popular encryption algorithms, as well as providing a high-level interface which hides the implementation details and provides an operating-system-independant encoding method which makes it easy to transfer encrypted data from one system to another. Although use of the

high-level interface is recommended, programmers can directly access the lower-level encryption routines for implementing custom encryption protocols or methods not provided by *cryptlib*.

Algorithm	Key size	Block size	Type
Blowfish	448	64	Cipher-block
CAST-128	128	64	Cipher-block
DES	56	64	Cipher-block
Triple DES	112 / 168	64	Cipher-block
IDEA	128	64	Cipher-block
RC2	1024	64	Cipher-block
RC4	2048	8	Cipher-stream
RC5	832	64	Cipher-block
Safer	128	64	Cipher-block
Safer-SK	128	64	Cipher-block
MD2	—	128	MD-Hash
MD4	—	128	MD-Hash
MD5	—	128	MD-Hash
MDC-2	—	128	MD-Hash
RIPEMD-160	—	160	MD-Hash
SHA	—	160	MD-Hash
HMAC-MD5	128	128	MAC
HMAC-SHA	160	160	MAC
HMAC-RIPEMD-160	160	160	MAC
Diffie-Hellman	4096	—	Key Exchange
DSA	4096 ¹	—	Digital Signature
ElGamal	4096	—	Public-key
RSA	4096	—	Public-key Digital Signature

Table A - 1: cryptlib mechanisms

Certificate Management

In relation to certificate management, cryptlib implements full X.509 certificate support, including all X.509 version 3 extensions. Since cryptlib is itself capable of processing certification requests into certificates, it is also possible to use cryptlib to

¹ The DSA standard only defines key sizes from 512 to 1024 bits, cryptlib supports longer keys but there is no extra security to be gained from using these keys.

provide full CA services. cryptlib can import and export certification requests, certificates, and CRL's in straight binary format,. This covers the majority of certificate and certificate transport formats used by a wide variety of software such as web browsers and servers.

Key Database Interface

cryptlib provides an interface to both native-format and external key collections. The cryptlib native format uses commercial-strength RDBMS's to store keys in the internationally standardised X.509 format. The cryptlib key database integrates seamlessly into existing databases, for example an existing database containing user names and email addresses may be extended to become a public key database with a single cryptlib function call. Existing applications need not even be aware that their address list database has become a public-key database.

cryptlib also supports external flat-file key collections such as PGP key rings and X.509 keys stored in disk files. The key collections may be freely mixed (so for example a private key could be stored in a disk file, a PGP keyring or on a smart card with the corresponding X.509 public key certificate being stored in an Oracle or SQL Server database).

Cryptographic Random Number Management

cryptlib contains an internal secure random data management system which provides the cryptographically strong random data used to generate session keys and public/private keys, in public-key encryption operations, and in various other areas

which require secure random data. The random data pool is updated with unpredictable process-specific information as well as system-wide data such as current disk I/O and paging statistics, network, SMB, LAN manager, and NFS traffic, packet filter statistics, multiprocessor statistics, process information, users, VM statistics, process statistics, open files, inodes, terminals, vector processors, streams, and loaded code, objects in the global heap, loaded modules, running threads, process, and tasks, and an equally large number of system performance-related statistics covering virtually every aspect of the operation of the system. The exact data collected depends on the hardware and operating system, but generally includes quite detailed operating statistics and information. In addition if a `/dev/random`-style randomness driver (which continually accumulates random data from the system) is available, cryptlib will use this as a source of randomness.

Prototype - Hardware & Software

The hardware platform used for the implementation consists of a PC with utilising Microsoft NT Server. The details of the hardware specification are as follows:

Personal Computer:	Omega – Cyrix P166
Processor:	Cyrix P166 133MHz
RAM:	97280KB
Hard Drive:	2 GB
Network Card:	SMC Ethernet Card
Operating system:	Microsoft NT Server 4.1; Service Pack 3

The platform was selected to accommodate the software packages (see below) that were required to build the demonstration software. The software packages used to implement the demonstration application were as follows:

Middleware:	IONA's Orbix 2.2c, 2.3c, 3.02
Programming language:	Microsoft Visual C++ version 4.2, 5, 6
Cryptography software:	cryptlib version 2.1b
Database Package	Microsoft Access

IONA's Orbix was selected because, when the research began, it provided the most comprehensive set of tools and functions of any of the available middleware products available for the Microsoft NT platform [i]. Initially, Orbix version 2.2c, with Microsoft Visual C++ Version 4.2 [ii], was used but this was later upgraded to Orbix version 2.3c using Microsoft Visual C++ version 5 and, finally, Orbix version 3.02 and Microsoft Visual C++ 6. Microsoft Foundation Classes (MFCs) [iii] were available in Visual C++ and were used to in building the user-interface of the demonstration software. The Microsoft Foundation Class Library (MFC) is an application framework for programming in Microsoft Windows and provides much of the code necessary for managing windows, menus, and dialog boxes; performing basic input/output; storing collections of data objects; and so on.

cryptlib [iv] is a security toolkit which allows programmers to easily add encryption and authentication security services to their software. cryptlib provides a transparent and consistent interface to a number of widely-used security services and algorithms (see Appendix D), which are accessed through a straightforward, standardized interface with parameters such as the algorithm and key size being selectable by the user.

In relation to certificates, cryptlib implements full X.509 support, including all version 3 extensions. Since cryptlib is itself capable of processing certification requests into certificates, it is also possible to use cryptlib to provide full CA services. cryptlib can import and export certification requests, certificates in straight binary format, and therefore covers the majority of certificate and certificate transport formats used by a wide variety of software, such as web browsers and servers. cryptlib was chosen because it is freely available and provides an extensive set of encryption and certificate management functions for the Win 32 platform.

Microsoft Access was the selected database package, used to store administrative data, because it utilises the widely supported Open Database Connectivity (ODBC) API, which provides the ability to write applications that are independent of any particular database management system (DBMS). Therefore, it is representative of a large section of the database world.

References

-
- [i] Standish Group
"CORBA ORBs"
Standish Group, February 1997.
 - [ii] I. Horton
"Beginning Visual C++ 4"
Wrox Press Ltd., 1996
 - [iii] M. Blaszcak
"Professional MFC with Visual C++ 5"
Wrox Press Ltd., 1997.
 - [iv] P. Gutmann
"Encryption Toolkit Version 2.1b"
P. Guttman, August 1998
<http://www.cs.auckland.ac.nz/~pgut001/cryptlib/>

E – Papers

This appendix present the papers based on the research work.

1. *“Addressing security in an Integrated Service Engineering environment.”*
Proceedings of EUROMEDIA 96, London, UK, December 1996.
2. *“CORBA Middleware Services: are they secure?”*
Proceedings of EUROMEDIA 2001, London, UK, April 2001.

Addressing security in an Integrated Service Engineering environment

E.M.Joyce, S.M.Furnell, P.L.Reynolds and P.W.Sanders

**Network Research Group, School of Electronic, Communication and Electrical Engineering,
University of Plymouth, Plymouth, United Kingdom.**

Abstract

This paper examines the requirements for security in the emerging area of Integrated Service Engineering (ISE). The ISE field is currently characterised by two alternative architectures, TINA and OSA, and the structure for a generic service machine encompassing both approaches is discussed. A number of ISE-specific security requirements are then identified and a conceptual solution is proposed based upon the Comprehensive Integrated Security System (CISS) architecture. This is shown to successfully map onto the structure of the ISE service machine. The paper is based upon ongoing research in this area which will lead to a practical implementation.

Introduction

Integrated Service Engineering (ISE) is an environment which handles the development, deployment and provision of services on a telecommunications infrastructure. This paper examines the issue from a security perspective, identifying the requirements involved in realising a secure system.

The current state-of-the-art in the ISE field is characterised by two architectures, namely TINA (Telecommunications Information Networking Architecture) [1] and OSA (Open Service Architecture) [2,3]. In order to reap the benefits of both approaches, this discussion will introduce a generic service machine structure that has been produced by the merging of the two architectures. This provides a platform upon which security can be implemented.

As can be seen from figure 1, the service machine has a layered structure. The top layer is the telecommunication applications level, which is divided into different segments (or separations) - Management, Service and Resource. Management and Service are taken from the TINA structure. Management deals with all entities relating to the control of the managing systems and can be divided by the OSI functional separations for systems management (i.e. FCAPS - fault, configuration, accounting, performance and security).

Service deals with all aspects of the service environment. It can be divided into Support and Session. Support is similar to the support services offered in OSA's service machine (i.e. trading service). Session deals with an actual service instance

and how it is completed. It takes on the TINA structure by sub-dividing into Access, Service and Communication. The service can then be viewed from the User or Provider perspective.

The Resource segment is seen as an amalgamation of TINA's Element and Network Element segments. It handles control of all resources at any level. Its Adaptors handle the mapping of physical network Resources and logical network Elements, so that they can be used by a service.

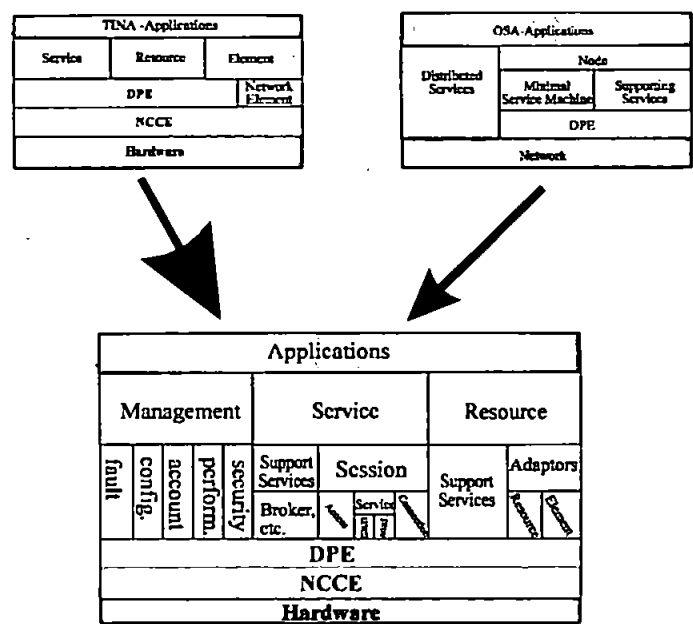


Fig 1: Generic Service Machine Structure

Security requirements for ISE

This section identifies a number of security considerations with specific relevance to the ISE environment. These are based upon issues identified by the OMG (Object Management Group) in respect of security for distributed objects [4].

Authentication: It is essential that system entities (e.g. users, services and components) can be identified and authenticated. Within ISE there are two possible scenarios. Firstly, the entity is identified and authenticated locally and, therefore, the validation can be trusted within the local domain. Alternatively, the entity may have been identified and authenticated by another node within the distributed system. In this case, the local security system still has to be able to validate the entity. A trusted third party (TTP) may be used here to issue a proof of authenticity that the local system can trust.

Authorisation and Access Control: Each identifiable entity should have an associated set of privileges which will be used when it is looking to access some other entity or resource. A large scale system will require the use of groups to cut down

administration overheads. However, it is sometimes desirable to have finer granularity, where privileges can be assigned to an individual entity to reduce the amount of damage any one entity can do. Therefore, the security system will have to be able to cope with different levels of authorisation. Again two levels of operation may be useful, one within a local system, and one using TTP certification to allow access across different domains.

Audit: System users must be able to be held accountable for their actions. As such, an audit trail should be maintained to record (selected) security-relevant events (e.g. data access, object activation etc.) associated with specific user identities. The log itself must be protected to prevent unauthorised modification.

Propagation of Attributes: An entity may invoke some other entity enabling the latter to carry out operations on its behalf. In order to facilitate authorisation and access control, the initial entity should be able to delegate its privileges. However, it may wish to restrict these (e.g. to a specific time or a certain access level, such as read instead of read/update) and different domain security policies may pose some difficulty. Firstly trust will have to be established between two domains. Secondly, the attributes from one domain may need to be mapped to authorised attributes in the other domain to provide validation for access control and auditing.

Secure Communications: Distributed communications require protection to preserve confidentiality, as well as to guard against corruption, redirection or other forms of attack. It is, therefore, necessary to guarantee secure end-to-end communications encompassing integrity, confidentiality and non-repudiation. A facility should also be available to specify the quality of protection. This would allow an entity to specify whether a whole session, or a particular message, should be protected and to what level.

Administration: Within the security system, identifiable entities need to be registered. The identities, their related privileges and other information (such as security groups/roles, access control lists) need to be maintained. Administrative operations should be restricted to valid entities, e.g. security administrators or parent entities who may register their child entity with the security system. It should be possible to split operations so that responsibility can be divided between different entities. This division could be either by function (where, say, a security auditor would be different from a security administrator) or by role (where service providers may have different functions available than do network providers).

Inter-domain Operations: ISE is an open and distributed environment. It must provide for international country boundaries or, more importantly, interactions between different administrative domains, as has been highlighted in the previous sections. This means that security policies and administration within a local domain need to be preserved, but this has to co-exist with the preservation of inter-domain security. RM-ODP proposes the use of traders and TTPs for this purpose [5]. This allows federation to take place between domains, via the trader, while trust is guaranteed by the TTP.

The Comprehensive Integrated Security System

Members of the research team have previously been involved in work relating to the design and development of the Comprehensive Integrated Security System (CISS) architecture - which facilitates a layered approach to security in an open distributed environments [6]. Given this legacy knowledge, it was deemed appropriate to consider CISS as an example platform upon which to demonstrate security in ISE. However, before examining this applicability in any detail, it is first necessary to provide some background information about CISS itself. The architecture supports local domain security, inter-domain security and incorporates modularity so that it can operate as an *add-on* service. It has five distinct layers, as depicted in figure 2 and described below.

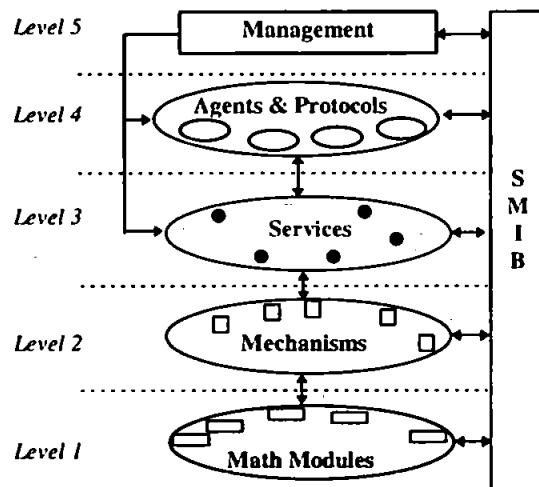


Fig 2 : CISS Layered Architecture

Mathematical Modules are used to implement Security Mechanisms. They are the lowest level, as they cannot be functionally broken down into lower sub-components. Several modules can be combined to implement a security mechanism.

Security Mechanisms are used to implement Security Services. Examples of mechanisms are simple password or digital signatures for authentication; encryption for data confidentiality.

Security Services are used by Security Agents. By combining different mechanisms, security services of varying efficiency and strength can be created to comply with a security policy.

Security Agents and Protocols provide the necessary interaction between CISS administration and security services. There are ten in total, as described in table 1 below.

Security Management deals with the support and control of secure operations. This includes:

- management and control of data (e.g. mechanism parameters);
- distribution of data (e.g. keys and security policy information);
- monitoring, logging and recovery (e.g. to ensure a stable security state);
- inter-domain management (e.g. exchange of security information to allow inter-domain communications).

CISS Agent	Function
User Agent (UA)	Interactions between operational/management users and CISS.
Security Administrator Agent (SAA)	Interaction with network management personnel and the security administrator, and agent for security policy controls by management.
Security Services Agent (SSA)	Provision, co-ordination and management of security services - the core of CISS.
Security Mechanisms Agent (SMA)	Provision, co-ordination and management of security mechanisms.
SMIB Agent (SMIBA)	Allows access to the SMIB, and performs all related operations on behalf of other CISS components.
Agent for Operational Environment Interactions (OPENA)	Interactions with the operational environment, primarily in the local environment.
Association Agent (AA)	Establishes and maintains security in the overall peer-entity associations.
Inter-Domain Communications Agent (IDCA)	Responsible for secure communications between heterogeneous security domains.
Monitoring Agent (MA)	Monitoring of all security relevant events, access to the security log and management of operations upon it.
Recovery Agent (RA)	Responsible for security violation detection and error recovery.

Table 1 : CISS Agents

Interactions with users and applications occurs via Application Program Interfaces (APIs).

Mapping CISS to the ISE architecture

It is now necessary to place a security architecture onto the generic service machine. It is not only the machine, but the security for the ISE environment that must also be considered. Therefore, the chosen solution for this problem is the application of the CISS architecture. The reasons for this are outlined in table 2 below.

Reason	Detail												
Open and distributed	CISS provides for an open and distributed environment which is a necessary requirement for ISE. It allows for local and inter-domain communications.												
Not service or mechanism specific	The architecture allows selection from multiple mechanisms and services in order to enforce security, enabling it to adapt to local security policies. These mechanisms allow the important ISE services of authentication and access control to be enforced.												
Modular	The use of APIs and modular structure allows CISS to be easily "added-on" to any system.												
Structure	CISS provides separate security service and management structures. This division is seen as important in ODP environments.												
Meets general requirements	CISS can provide for general security requirements in distributed heterogeneous systems, e.g. scalability, consistency, interoperability, availability, regulatory requirements, usability, performance.												
Meets ISE security requirements	<p>CISS can provide the ISE-specific security requirements via the following agents :</p> <table> <tr> <td>Identification & Authentication:</td><td><i>SMIBA, SSA, SMA, UA</i></td></tr> <tr> <td>Authorisation & Access Control:</td><td><i>SMIBA, SSA, SMA, UA</i></td></tr> <tr> <td>Propagation of Attributes:</td><td><i>SMIBA, UA</i></td></tr> <tr> <td>Secure Communication:</td><td><i>OPENA, AA, IDCA</i></td></tr> <tr> <td>Administration:</td><td><i>SAA, SMIBA</i></td></tr> <tr> <td>Inter-domain Operations:</td><td><i>IDCA</i></td></tr> </table>	Identification & Authentication:	<i>SMIBA, SSA, SMA, UA</i>	Authorisation & Access Control:	<i>SMIBA, SSA, SMA, UA</i>	Propagation of Attributes:	<i>SMIBA, UA</i>	Secure Communication:	<i>OPENA, AA, IDCA</i>	Administration:	<i>SAA, SMIBA</i>	Inter-domain Operations:	<i>IDCA</i>
Identification & Authentication:	<i>SMIBA, SSA, SMA, UA</i>												
Authorisation & Access Control:	<i>SMIBA, SSA, SMA, UA</i>												
Propagation of Attributes:	<i>SMIBA, UA</i>												
Secure Communication:	<i>OPENA, AA, IDCA</i>												
Administration:	<i>SAA, SMIBA</i>												
Inter-domain Operations:	<i>IDCA</i>												

Table 2 : CISS in ISE

The CISS functional structure is considered to be compatible with that of the service machine. This mapping is shown in figure 3 and briefly explained in table 3.

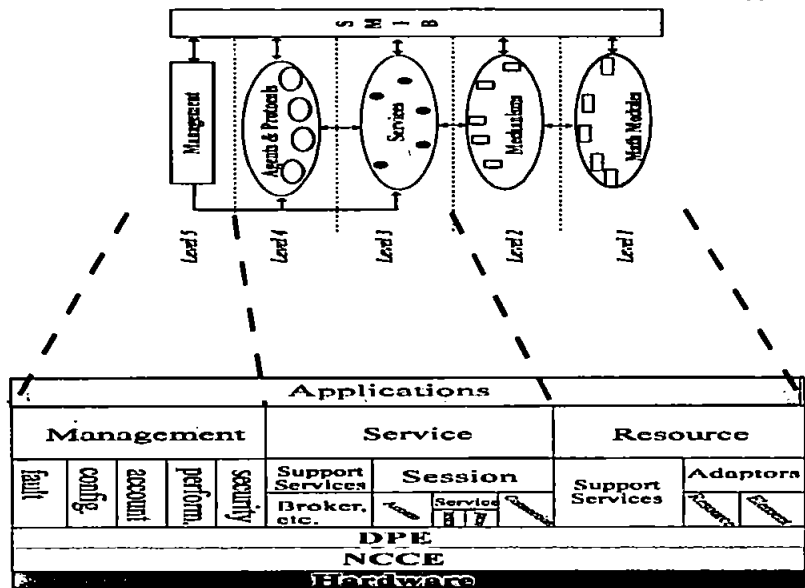


Fig. 3 : Functional Mapping between CISS and service machine

Service Machine	CISS	Detail
Management	Layer 1: Management	There is a direct mapping between the Management layers in both CISS and the Service Machine. Both provide the support and control functionality necessary.
Service	Layer 2: Agents and Protocols; Layer 3: Services	The service machine Service segment will map onto layers 2 and 3 in CISS as both are dealing with services, i.e. entities which have the ability to complete operations and are not just components). These provide the functionality of the service machine or security system.
Resource	Layer 4: Mechanisms; Layer 5: Math. Modules	The Resource segment in the service machine maps onto layers 4 and 5. In both cases, these are the lower levels of the architectures. They are the components which are combined to produce the services required.

Table 3 : Functional mapping between the service machine and CISS

CISS Agents on a Service Machine

Security agents provide the necessary interaction between CISS administration and security services. They are the core of the security architecture. It is therefore

necessary to see how they map onto the service machine structure. Figure 4 shows the suggested placement of each agent, with justification provided in table 4.

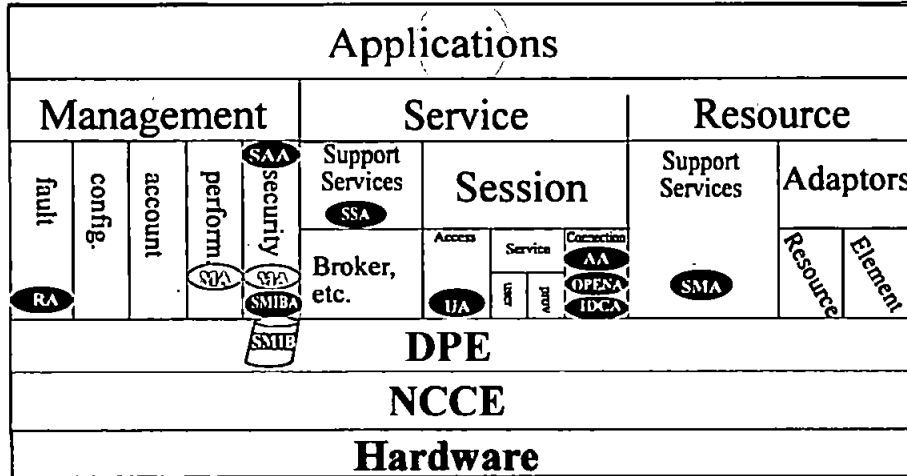


Fig. 4 : CISS agents in a service machine

CISS Agent	Service Machine Area	Detail
User Agent (UA)	Access	The TINA model defines the Access area as a users ability to have flexible access to services [7]. It also defines a user agent which represents and acts on behalf of the user. It receives requests from users to establish or join service sessions. The CISS UA allows interactions between users and CISS. Therefore, it should be placed in the Access area.
Security Administrator Agent (SAA)	Security	The Security area is responsible for the support and control of security services. However, this has not been fully defined in either OSA or TINA. The SAA provides interaction between network management personnel or the security administrator, and agents for security policy controls by management. Therefore, the SAA should be placed in the Security areas to allow the security administrator access.
Security Services Agent (SSA)	Support Services (Service Segment)	The Support Services area of the service segment, will provide any non-core services, i.e. those not required to actually provide the service session, but which can support it. The SSA deals with the provision, co-ordination and management of security services. Therefore, it will be involved in providing security when a service session is established or joined.

<i>CISS Agent</i>	<i>Service Machine Area</i>	<i>Detail</i>
Security Mechanisms Agent (SMA)	Support Services (Resource Segment)	The Support Services area of the resource segment will provide any supporting services required for resources, either software or hardware. The SMA deals with the provision, co-ordination and management of security mechanisms. As security mechanisms are viewed as resources, the SMA should be located in the Support Services area of the resource segment.
SMIB Agent (SMIBA)	Security	As previously stated, the Security area provides for the support and control of security functions and includes all security relevant data. In CISS, the SMIB is a central repository where all such security relevant data is maintained. The SMIBA allows access to the SMIB and performs all operations on behalf of other CISS components. Therefore, the SMIBA should be located in the Security area.
Operational Environment Interactions Agent (OPENA)	Connection	The Connections area handles the communications connections associated with a service session, as described in TINA. The OPENA interacts with the operational environment to allow access to resources in a secure way. Therefore, the OPENA should be placed in the Connection area, to allow secure communications within the local environment.
Association Agent (AA)	Connection	The AA establishes and maintains security in the overall peer-entity associations, i.e. it provides communication with other applications in the same domain. Therefore, it too should be placed in the Connection area to provide secure communications within the current security domain.
Inter-Domain Communications Agent (IDCA)	Connection	The IDCA is responsible for secure communications between heterogeneous security domains, i.e. inter-domain communications. As the Connection agent deals with all communications, the IDCA should be placed there.
Monitoring Agent (MA)	Performance or Security	The Performance area is responsible for monitoring and managing system performance. The MA monitors all security relevant events, provides access to the security log and manages

<i>CISS Agent</i>	<i>Service Machine Area</i>	<i>Detail</i>
		all operations on it. Therefore, aspects of the MA should be placed in the Performance area. However, the Security area is another possible location for this agent, as monitoring can also be considered a function of security (e.g. a service such as user or session supervision).
Recovery Agent (RA)	Fault	The Fault area is responsible for detecting errors and then managing the corresponding recovery mechanisms. The RA is responsible for all security violation detection and CISS error recovery. Therefore, the RA should be placed in the Fault area.

Table 4 : CISS Agents in a service machine

A Working Model Example

The following example demonstrates how a security model should operate with a working service. The service is broken up into basic steps that a user would take. Each step is then subdivided into the activities executed by the service. The security services required are then listed under each of the appropriate service activities and related back to the requirements listed in section 2. The service example described is a user engaging in a document editing session with another party and is based upon a modified version of a TINA service example that is described in [7].

1. *User A selects a terminal to give him access to the network.*
2. *User A logs on to the network. This can be done using one of several mechanisms, but for this example an identifier and password are used (a smartcard would be another possible mechanism).*
 - (a) The ID and password are taken by the security service to authenticate the user.

An information base will be referenced to check that the ID exists and that the password is valid. Trading may be required if the ID cannot be found locally, as will a TTP if the ID is in a different security policy domain. User A's privileges will be returned to a local information base if they are not held locally. This will help performance.

Relates to: Identification and Authentication, Administration (to maintain the information base), Inter-domain (access user information in another domain, if necessary)
 - (b) User A's user agent is now associated with a terminal agent.

User A's privileges will be checked to ensure he is permitted to use the terminal he is currently logging onto. Once validated the user agent and terminal agent are associated.

User A's privilege's are propagated to his user agent.

The log-on is logged by the security service.

Relates to: Administration, Authorisation and Access Control, Propagation of Attributes, Audit

3. *User A is presented with a menu of capabilities at his terminal.*

- (a) The security service checks User A's privileges to see what capabilities he can access.

The security service checks User A's privileges. A list of valid options are created.

Security service may need to validate that the terminal can access these capabilities also, by validating the terminal agents privileges.

Relates to: Administration, Authorisation and Access Control

- (b) The security service sends a list of the valid options to the terminal agent.
(c) Terminal agent presents a menu on the terminal.

4. *User A selects an option for document editing.*

- (a) A request is passed to the user agent to establish a document editing service session.

- (b) The user agent creates, via the factory in the DPE, a service session manager (SSM).

The factory will be checked by the security service to ensure it has the capabilities to create such a service, and that it can do so for the specified user, User A. Again an information base, holding the factory capabilities will need to be tested and checked against User A's privileges.

Relates to: Administration, Authorisation and Access Control

- (c) User A is joined to the session by creating a user agent
The event is logged.
Relates to: Audit

5. *User A selects a document to be opened.*

- (a) The user agent sends a request to open a document to the SSM.

The security service checks that User A has access to the specified document and with the correct access type (e.g. read/write).

The security service locates the document, a trader may be necessary, and checks that document can be accessed.

The security service notifies the SSM that the request has been validated.

Relates to: Administration, Authorisation and Access Control

- (b) The SSM opens the document in the session, by notifying the user agent and connecting the document resource agent.
The event is logged.
Relates to: Audit

6. *User A requests that User B is added to the session.*

- (a) User agent sends the request to the SSM.

- (b) SSM locates User B using the specified ID. A trader and TTP may be required to locate User B.

The security service locates User B and identifies and authenticates him.

The security service accesses User B's privileges. This may be in a remote information base via a remote security service.

The security service then checks that User A and User B can join in a session together.

Once validated, security service notifies the SSM.

Relates to: Identification and Authentication, Administration, Inter-domain, Authorisation and Access Control.

- (c) User B's user agent alerts the appropriate terminal agent of the incoming request.
- (d) User B's terminal agent then alters the terminal by presenting a window on the terminal.
- (e) User B accepts the request.
- (f) The response is sent to the SSM.
 - User B is the validated to ensure he has access to the opened document.
 - Relates to: Authorisation and Access Control.
- (g) SSM creates a user session for User B.
 - User B's privileges are propagated to his user agent.
 - The event is logged.
 - Relates to: Propagation of Attributes, Audit

7. User A requests SSM to set-up a video conference connection with User B.

- (a) User A's user agent requests SSM to establish video conference connection with User B.
- (b) SSM requests the connection service manager (CSM) to establish a stream between the end-user applications on the two terminals.
 - The security services will validate that both User A and User B, and their terminals, have the appropriate capabilities.
 - The security service will validate that the CSM has access to the appropriate resources to establish the stream.
 - Relates to: Authorisation and Access Control
- (c) CSM establishes a stream between the users and sends a response to the SSM.
 - The stream needs to be secured.
 - Relates to: Secure Communications
- (d) The SSM sends a response to User A.
 - The event is logged.
 - Relates to: Audit

This example shows how a security model would operate if all validations were successful. However, if one failed, then the request would be denied, the appropriate response sent to the requesting agent and the event then logged. Depending on the severity of the violation, other measures may have to be taken, such as the security administrator being alerted. However, the precise actions will depend on the domain security policy.

Conclusion

Integrated Service Engineering is a relatively new term which has only come to prominence in the last few years. It considers the problem of service development,

deployment and provision in the heterogeneous telecommunications environment today. Security, on the other hand, is a much older school, with well developed mechanisms and theories. However these are continuously scrutinised and modified to deal with the new problems posed in a computerised/technological world. The paper has considered how they may be applied to the specific issue of ISE.

The CISS architecture is shown to be a complete security solution for distributed networks. However, it also adheres to the requirements specified for ISE in supporting local security, inter-domain security and providing a modular service that can be integrated into any system compatible with the ISE architecture. Further practical work is ongoing in this area and will lead to the development of a demonstrator system in due course.

References

- [1] Barr, W.J.; Boyd, T. and Inoue, Y. 1993. "The TINA Initiative", *IEEE Communications Magazine*, March 1993.
- [2] Prevedourou, D.; Stamoulis, G.D.; Tonnby, I. and An, T. 1994. "Providing Services in a World of the IBC Resources: An Architectural Approach", in *Proceedings of the IS&N '94 conference* (Aachen, Germany, September 1994).
- [3] Bruno, G.; Lucidi, F.; Insulander, J. and Larsson, U. 1994. "A Service-Driven Vision of Integrated Broadband Communications: the OSA Approach", in *Proceedings of the IS&N '94 conference* (Aachen, Germany, September 1994).
- [4] OMG. 1994. *OMG White Paper on Security*. Issue 1.0. Object Management Group Security Working Group. B. Fairthorne (Ed.). April 1994.
- [5] ISO. 1993. *Working document on topic 9.1 - ODP Trader*. ISO/IEC JTC1/SC 21/WG7 N 807.
- [6] S. Muftic, S.; Patel, A.; Sanders, P.; Colon, R.; Heijnsdijk, J. and Pulkkinen, U. 1994. *Security Architecture for Open Distributed Systems*. J. Wiley & Sons.
- [7] TINA-C. 1995. *Overall Concepts and Principles of TINA*. Version 1.0 Publicly Released, <http://www.tinac.com>, February 1995.

CORBA Middleware services – are they secure?

E.M.Joyce, S.M.Furnell, P.L.Reynolds and P.W.Sanders

Network Research Group, School of Electronic, Communication and Electrical Engineering,
University of Plymouth, Plymouth, United Kingdom.

1. Introduction

Distributed object systems are used everywhere – the Internet, telecommunications, banking... the list goes on. But securing such systems is not a simple task. For instance consider one of today's middleware choices, the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) is such a technology. Although there is a security solution, this paper will show that it has not addressed all the possible security threats.

In CORBA, a client is an entity that wishes to invoke an operation on a target object via the Object Request Broker (ORB). The object implementation comprises the code and data that realise the target object's behaviour. The ORB receives a request and then locates an appropriate object implementation, and transmits the request data and results between the client and the target object. There is also a set of supporting services that are used to extend the ORB functionality, and without which a standardised distributed solution would not be possible. It is the security of these services that this paper will focus on.

According to the International Standards Organization (ISO), security should be provided in a modular format [1]. This architecture divides system management into functional units, FCAPS – the 'S' being the security module. A system should be able to function independent of the security service, and when the security module is introduced the same system should now operate in a functionally similar but secured fashion. This type of thinking is practical in a centralized system, such as a mainframe, where the Trusted Computing Base (TCB) [2] is contained within a single system. The security service can monitor all requests and provide the required security functionality. However, distributed systems are more complex. Distributed objects introduce complications and the TCB is no longer contained in a single system and may need to operate across multiple systems and security domains. This results in an extended set of security requirement for a distributed processing environment (DPE) such as CORBA, and therefore the modular solution may be inadequate.

1.1 Security Issues for Supporting Services in a DPE

CORBA currently consists of an ORB and 15 CORBA services [3]. Each service is implemented by a number of objects, the interfaces of which are defined in Interface Definition Language (IDL). Currently security is implemented by applying the security rules to these service objects. This means that access can be granted to a client, when requesting use of a CORBA service object, if the client possesses the appropriate privilege attributes. However, even looking at an overview of the services some security issues become apparent. They are outlined below:

- **Persistence State Service (PSS):** The PSS stores components persistently on a variety of storage servers. Although access to the persistent storage objects are controlled, the stored data is not secured – the security service has no

control over this; it would be an implementation level detail, i.e. if the data was stored in a database, the implementer would enable database security.

- **Naming Service:** The Naming Service (NS) locates components by name. Once an object can access the NS, it can access all names in the service, as there are no security restrictions. Also NSs can be federated, i.e. two naming services are linked together to operate like a single service. If the federation exists across different security domains the client is unaware that he is crossing a domain boundary and security controls could be by-passed
- **Event Service:** This service allows “consumers” to register/unregister interest in specific events. The “suppliers” then generate information about this event and send it to the consumers via an event channel. It is a basic publish/subscribe or notification service. Security has not been defined for the event channels, i.e. access control is not available for specific events on a single channel, and there is no indication whether the channel requires encryption. Also the event service demands a certain amount of Quality of Service (QoS), i.e. guaranteed delivery, persistence of event data in the event of an event channel failure and use of logging facility. If the event channel was subject to encryption then the supporting QoS mechanisms, would also need to ensure security, e.g. the persisted data would have to be protected.
- **Query Service:** This allows a client to use query operations for attributes associated with objects, in much the same way SQL can be used to query a database of records by querying the fields in the records. It provides for asynchronous query, so that the query can be issued and the client does not have to block while waiting for a response. No security precautions have been added and so there is no way to identify what attributes a client can perform queries on, e.g. does the client have the security clearance to query a payroll attribute on an employee database. Another problem is Denial of Service, e.g. a rogue client can flood the query service with too many asynchronous or long running synchronous queries thereby causing the services to halt or crash.
- **Trader Service:** Similar in function to the NS, the Trader allows an importer to locate an object, published by an exporter, but this time it does so by identifying a set of required properties, e.g. like the Yellow Pages. A security problem could arise if some of the services offered by the trader require higher security clearance than others; there is no way of controlling access to particular offers in a single Trader.

Obviously there are security issues that exist in CORBA services that are not handled currently by CORBA Security Service (CORBASec). The above descriptions are just high-level overviews of such problems, but the problem demands further detailed investigation. Therefore a single service was selected and examined in detail.

1.2 Selecting a CORBA service

A Trader facilitates the dynamic offering and discovery of service instances of particular types within a distributed environment. As such, it allows clients to advertise their available services and to also match their needs against other advertised services.

Traders have an important role to play in future Internet and telecommunications networks. It can perform its basic 'yellow pages' function in the world of e-commerce by providing access to internet services, e.g. a financial Trader may provide lists of financial services that a user may wish to buy over the Internet, everything from car loans to share brokerage services. The user can decide which Trader to advertise its services in, and which Trader to import services from. The Traders can be structured to provide a greater degree of choice, e.g. a financial services Trader, may be linked to a car loans Trader and a stock brokerage Trader (and many other such traders) as opposed to having the services registered directly in its own registry.

Resnick [4] suggested that the Trader could be used to standardise World Wide Web (WWW) facilities. There are a dizzying array of choice of search engines, web crawlers and white pages such as Yahoo, HotBot, and Alta Vista. However, these facilities, especially the search engines, lack a programmatic interface and differ not just in implementation but also in how they are accessed, how predicates are formed and how Uniform Resource Locators (URLs) are registered. Therefore a synergy between the CORBA Trader and the Internet facilities would offer a solution. Search engines would benefit from a standardised programmatic API, represented in CORBA IDL.

It is also important to remember that CORBA is not just for Internet use. It is designed to work on any heterogeneous distributed object environment. Therefore some other possible uses of the Trader have been suggested by the Distributed Systems Technology Centre (DSTC) research group in University of Canberra, Australia [5]:

- real-time trading, e.g. dynamic configuration of services within telecommunications switches (combining bandwidth from local and trunk carriers to provide an end-to-end service);
- large scale trading, e.g. using trading to access network elements from network management applications for a national telephone system.

2. The need for Security

After the publicity and damage caused by viruses and such as the "Love Bug" [6] and numerous hacker attacks, business are taking security seriously. Businesses have suffered huge losses as a result of cybercrime. On 8 December 2000, a hacker stole 55,000 credit card numbers from CreditCard.com, and when the company refused to pay any money for extortion, the hacker posted the numbers on a web-site [7]. According to the 5th annual "Computer Crime and Security Survey", conducted by the Computer Security Institute (CSI) and the US Federal Bureau of Investigation, such cyber-crimes are widespread, diverse in nature and on the increase [8]. 90% of survey respondents reported computer security breaches within the last year; 74% suffered financial loss as a result of security breaches and of the 42% (i.e. 273 respondents) who were willing to quantify those losses, the financial lose was estimated to be \$265,589,940.

Security for any distributed system uses five basic and partially overlapping services as specified by the International Standards Organisation (ISO):

- **Authentication:** The security service should be able to guarantee that the user/resource is actually who/what it claims to be. One type of threat is known

as a **masquerade**; that is when an entity successfully pretends to be some other legal entity and thereby gains illegal access to a resource.

- **Access control:** Protects resources from unauthorised use. It can be used on various assets, e.g., communications, data. It provides for the various types of access to a resource, e.g. read, write, update, or execution;
- **Confidentiality:** Confidentiality means being able to guarantee the privacy and secrecy of a resource such as a data file containing personnel details. Apart from unauthorised access to a resource, the loss of anonymity or the misappropriation of messages or data records can be considered breaches of security;
- **Integrity:** Integrity of resources ensures that they are always available and correct, no matter what corruption attempts have been made. Therefore any integrity services must guard against any threats involving illegal asset/resource modification;
- **Non-repudiation:** Repudiation is the denial of an action by an entity, e.g. a user may deny sending or receiving a message. Non-repudiation forces an entity to *own up* to its participation in some action. Denial of origin, transmission, receipt or participation are all repudiation threats.

By applying these concepts, a system can be made secure. However to implement security, these concepts must be realised. Security **mechanisms**, or methodologies, must be used to actually implement these security services, e.g. cryptography, digital signatures, access control lists. The ISO also defines a security policy as a set of criteria for provision of security services. It defines what is and what is not permitted in the area of security during general operation of a secured system. It must be implemented by taking the appropriate security measures. However, no security measures, no matter how ingenious they may be, will be effective unless the user understands what needs to be protected and can determine what mechanisms are used, i.e. what the policy is. Security needs a complete and usable **administration** system that will allow users to maintain and operate security on a day-to-day basis.

It is clear that the intense interest in security in web-based [9] and other distributed systems security [10,11] means that Traders will have to incorporate security if they are to be included in this future. Even though Traders can make use of CORBASec to counteract threats, there are still some security holes. These Trader-Security issues are addressed below, after describing how CORBASec and the Trader operate.

2.1 CORBA Security Service

CORBASec provides a framework for distributed object security. There are two levels of security. Level 1 provides protection for applications that are “unaware” of security, by transparently calling security functions on object invocation. Level 2 security provides more facilities and allows applications themselves to control the security provided, i.e. security-aware applications.

CORBASec currently supports certain levels of authentication, access control, confidentiality, integrity and non-repudiation. Another feature of CORBA security is

the use of credential delegation between objects. It allows credentials to be propagated along an object request chain.

Security is implemented by a number of objects, as shown in figure 1 below. Apart from the specific security interfaces, CORBA makes use of two objects, **Current** and **Credentials**. Current, a pseudo-object initially used by the transaction service to propagate transaction context, it is now adopted by security to propagate the security context. It does so by holding a reference to Credentials. Once a user is authenticated, a Credentials object is created. It holds information such as roles, privileges and an authenticated ID.

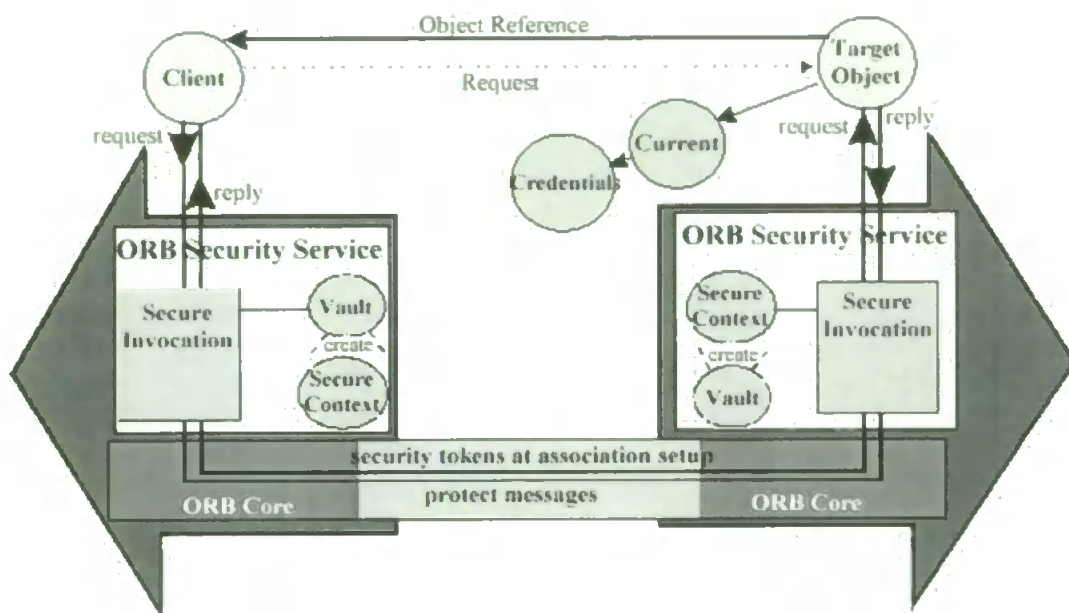


Figure 1 : CORBA Security Service

3. Traders

The OMG / CORBA Trader [12] provides the ability to match a service request, against a list of supported services provided by potential servers, as illustrated in figure 2. The exporter will advertise its available services, by notifying the Trader. The Trader keeps a Registry of such advertisements. An importer makes a request on the Trader for a particular service, specifying any conditions that need to be met. The Trader checks its Registry to find a matching service type, with corresponding conditions. The Trader then notifies the importer of the exporter and the service.

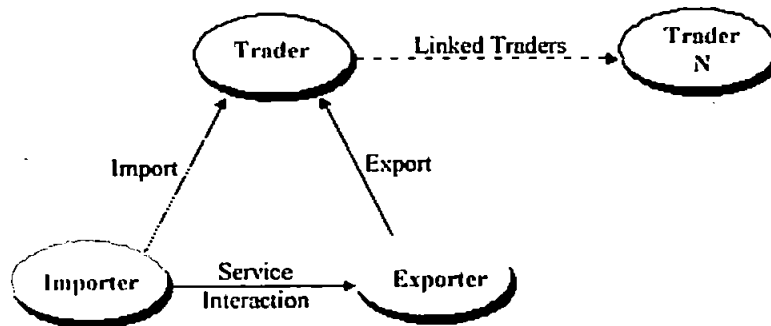


Figure 2 : Trader Interactions

If a Trader cannot find a matching service, it will then pass the request onto another linked (or federated) Trader. The linked Trader can then check its Registry to see if it can match the original request. Therefore trading allows an importer access to multiple Trading domains. The second Trading data store is the Service Type Repository. It stores, retrieves, manages and names service types² that are used in the Registry. Importers, Exporters and the Traders are all part of the Trading Community, i.e. all objects that interact to import/export services.

Each Trader also has Attributes. These define a Trader's characteristics, e.g. policies for scoping the extent of a search.

4. Security issues related to Trading & Traders

Traders, in a distributed environment like the Internet, are open to attack, just like any part of a distributed system. The following outlines the areas most vulnerable to security breaches and the security services that must be used to counteract them.

4.1 Authentication

Traders receive requests for imports/exports from members of the trading community. Like any system resource, they are susceptible to masquerade. Authentication is the service required to deal with this threat. It is a two-way process; traders, as well as importers and exporters should be identifiable and authenticatable. One possible way of achieving this is the use of certification by Trusted Third Parties (TTP). The ISO's X.509 [13], an authentication framework using public-key certificates, could be used. It is a hierarchy of Certification Authorities (CA) which issue signed certificates³. Authentication is accomplished through the presentation of a certificate signed by a trusted CA.

4.2 Access Control

² Service Types are associated with a traded service and are used to describe the service. They comprise an interface type and zero or more named property types [7].

³ A Signed Public-Key Certificate is someone's public key, signed by a trustworthy party. X.509 specifies a structure for public-key certificates that includes the user's unique name, a version number, algorithm identifier, issuer's name, validity period, etc.

Access Control needs to be handled at two levels. Firstly, access control of the Trader itself should be considered, i.e. who has access to the Trader. Secondly, access control of service offers must be dealt with, i.e. which service offers an importer can see.

Unauthorised Trader Access

Traders should have security attributes. Two trading community objects, e.g. Trader and exporter, have access to the security domain Access Control Manager – in CORBA this would be the AccessDecision object. Therefore, AccessDecision can make decisions relating to who can have access to which Trader, using the domain's access control mechanisms and working in accordance with the access control policies.

Unauthorised Service Offer Access

Even if an importer has access to a Trader it may not have access to all the service offers the Trader holds. Some of the service offers may be of a higher security classification. Therefore, a Trader will have to hold an associated security attribute with each service offer held in the Registry.

Current Access Control Limitations

Although access control of the Trader can currently be handled by CORBA's AccessDecision object, the access control of the service offers within the Registry cannot. It would require the storage of a security attribute in the Registry itself. The reason for this is that such an attribute would be used to sort and make selections when providing service offer lists to importers. This problem is also linked to Delegation, as the security attribute would have to be set and would probably be delegated from the exporter, e.g. use the exporter's security level.

4.2 Integrity and Confidentiality

Integrity and confidentiality of data, stored or in transit, must be guaranteed in a distributed system; this has to include trading-related data.

Stored Data

Details of service offers, including an object reference, are stored in the Registry. Therefore it must be protected, as an intruder may try to gain unauthorised access to a service, by gaining illegal access to the object. Similarly details of the Service Type held in the Repository, should be protected to ensure that intruders do not have knowledge of "how" to use the service type, i.e. interface details, parameters, etc.

It is not wise to assume that the Trader's backend data, i.e. the data stored in the *Registry* and *Repository*, is hidden behind object interfaces and, therefore, is not as vulnerable to attack as object references that are exported through the interface. Intruders do not always use legitimate access mechanisms and, therefore, the 'backdoor' entry must be considered. Such data will usually be held in persistent storage, such as a database, or flat file. Therefore the Trader, if operating as a security-aware service, should be able to guarantee that the data is secure, even when it is in storage. Cryptographic mechanisms are used to ensure that the confidentiality and integrity of the data is preserved.

However, these types of solutions are product dependent and so the only way to ensure a truly generic solution would be to use the Persistent State Service⁴ (PSS) in a secure fashion.

Inter-Community Communications

Since a Trader is operating in a distributed environment, this provides an intruder with ample access to intercept any communications between members of a trading community. From such interceptions, one may be able to re-construct Registry/Repository information. In addition, replay attacks have to be considered.

All communications between trading community members should be encrypted to ensure the confidentiality of any intercepted messages. Another form of communications security is a digital signature. The Digital Signature Standard (DSS) [14] uses a public key to verify to a recipient the integrity of data and the identity of the sender of the data. The DSS can also be used by a third party to ascertain the authenticity of a signature and its associated data. Finally replay attacks can be dealt with by using sequencing data.

Use could again be made here of security-aware CORBA services. In this case it would also be necessary for the Query service⁵ to be security-aware. This would allow the Trader or other trading community members to interrogate the Registry/Repository, in a secure manner.

Current Integrity and Confidentiality Limitations

Securing trader data, such as that held in the Registry and Repository, needs to be addressed. Currently these databases are not encrypted. Also trading community communications should be secured. The level of security would depend on the objects involved and their security level, as well as the level of the service offers being exported/imported.

4.3 Non-Repudiation

The trading community is made up of distributed objects, which are less predictable, due to their flexible and granular nature. There are two problems. Firstly, if the intruder is an authorised user, or is successfully masquerading as an authorised user, how can their actions be discovered? For example, an intruder can masquerade as an importer, and query Traders to find useful service offers. The processing of a monitoring database may help, by providing clues to an intruder's activities. Secondly, if adhoc interactions are taking place, how can it be proven that a specific interaction took place, if one party wishes to deny the event, i.e. accountability? Irrefutable evidence is required, i.e. a non-repudiation service.

Monitoring

All security related events should be monitored. These events are defined by the security policy. Apart from notifying an administrator, via an alarm, that an illegal action has been taken, monitoring could also provide clues to a previously unknown

⁴ The *Persistent State Service* provides a single interface for storing components persistently on a variety of storage servers – including object databases, relational databases and flat files.

⁵ The *Query service* provides query operations for objects. It is a superset of SQL.

intruder, e.g. an importer making multiple unauthorised import requests on several Traders. However this requires data filtering to find trends that can be used to raise a system administrator's suspicions, i.e. intrusion detection.

Irrefutable Evidence

Non-repudiation is used to provide irrefutable evidence that certain events took place. For example, digital signatures can be used with audit logs to record events. Just as other system resources are subject to a non-repudiation policy, so too are all the trading community members.

Current Non-Repudiation Limitations

There are two issues relating to non-repudiation. Firstly, the current CORBAMSec non-repudiation service is not complete. It deals with evidence generation and verification, but does not address delivery and evidence storage. Secondly, non-repudiation is considered to be an optional service. It is available, but only to security-aware applications. It should be made available to security-unaware applications.

5. Modifications required for Security-Aware Traders

Both the Trader and the Security Service require modification if they are to provide a Security-Aware Trader.

5.1 Security-Aware Trader Attributes

Attributes are already used in the Trader specification to provide a framework for describing the behaviour of any OMG Trader. It is proposed that *Security Attributes* be added for use by the Trader. They will control the security behaviour of a Trader, by specifying which security services the Trader uses, i.e. just how security-aware the Trader is. The suggested security attributes are defined in Table 1 below.

Security Policy-Attributes	Flags use of following function
Security-aware	All other policies are checked as the Trader is using security (at some level)
Access_control_trader	Includes Trader in ACL and uses authentication with trading community members, etc.
Access_control_service_offers	Provides access control on the service offers listed in a query
Encrypt_stores	Encrypts Registry and Repository
Encrypt_comms	Encrypts communications
Integrity_check_stores	Integrity checks Registry and Repository
Integrity_check_comms	Integrity checks communications
NR_trade	Non-repudiation of Trading related events
Audit_trade	Audit Trading related events

Table 1 : Trader Security Attributes

For example, a Trader could be a *Public Trader*. This means that everyone would have access to it and it would have no security applied, i.e. the *Security-aware* attribute would be set to off, indicating that all other attributes were also turned off.

Alternatively a Trader may be a *Secured Trader*. It would be *Security-aware* and have *all* other attributes turned on, i.e. it would use all the available security services. Another option is to make a Trader a *Security-Aware Trader*. In this case the security-aware attribute would be on, and *some* of the other attributes would be on, e.g., *Encrypt_stores* and *Integrity_check_stores*, but not *NR_trader* or *Audit_trader*, thereby providing a specified level of security.

5.2 Security-Aware Trader Data Structures

The two Trader data structures are the Repository and the Registry. The Repository should not have to be modified, as it will hold the security attributes in the same manner as it currently holds any other properties.

The Registry will not have to be modified either. It holds details of the instances of service offers: This includes the service type, an object reference and a set of properties held as name-value pairs. A new security property that defines the security level of a service offer will now be held in the Registry so that access controls can be applied to the offer. The exporter will specify the security level.

5.3 Security-Aware Trader Interfaces

There are eight interfaces defined for a CORBA Trader. However only one of these interfaces should have to be modified, namely the **Admin** interface. The Admin interface allows the administrator to configure the Trader, by using *Set* methods on the Trader's *Attributes*. These methods will now have to deal with the additional security attributes specified in table 1 above, to control the Trader's security behaviour. If *Security-aware* is set to on, then at least one other security attribute must be set to on also; otherwise an error will be returned on the *Set* method. If *Security-aware* is set to off, then all other security attributes must also be set to off; otherwise an error will be returned on the method.

5.4 An Enhanced CORBA Security Service

The CORBA security service is itself incomplete. There are certain facilities missing or incomplete. Firstly non-repudiation is only supports evidence generation and verification. It does not deal with delivery, storage or adjudication issues. Secondly, the audit facility is a simple one and does not address the needs of today's Intrusion Detection Systems. Thirdly, Secure Interoperability is also limited between security domains. Both domains must possess the same mechanisms and policies. Such limitations would mean that if two federated traders existed in different security domains, they may not be able to communicated if they have to do so securely. Finally, security administration is another problem area. Most ORB security product vendors promote the fact that they have gone beyond the CORBA Level 2 specification and provide administration services, but sure security administration should be part of the overall standards to allow integration between products. By enhancing CORBASec to make these facilities available, it would provide better security for ORB operations. However, this is a complete topic in itself and outside the scope of this paper.

5.5 Security-Aware CORBAService

As was mentioned earlier, if other CORBAServices were secured then a more generic security solution could be applied. If services such as the PSS, Query and Collection services were security-aware they would able to guarantee security of the data they were accessing. Then other CORBAServices, such as the Trader, could make use of them. For example, if the PSS was secure, the Trader could use it to access its Registry and Repository.

5.6 Modification Summary

Figure 3 (based on the OMG Trader), summarises the modifications that have to be made to the CORBA Trader to create a Security-aware Trader. The modifications are as follows:

1. New Trader Security Attributes;
2. New Registry Security Property;
3. Modified Admin interface;
4. Use of the Enhanced Security Service (including Enhanced Secure Interoperability Service);
5. Use of security-aware CORBAServices

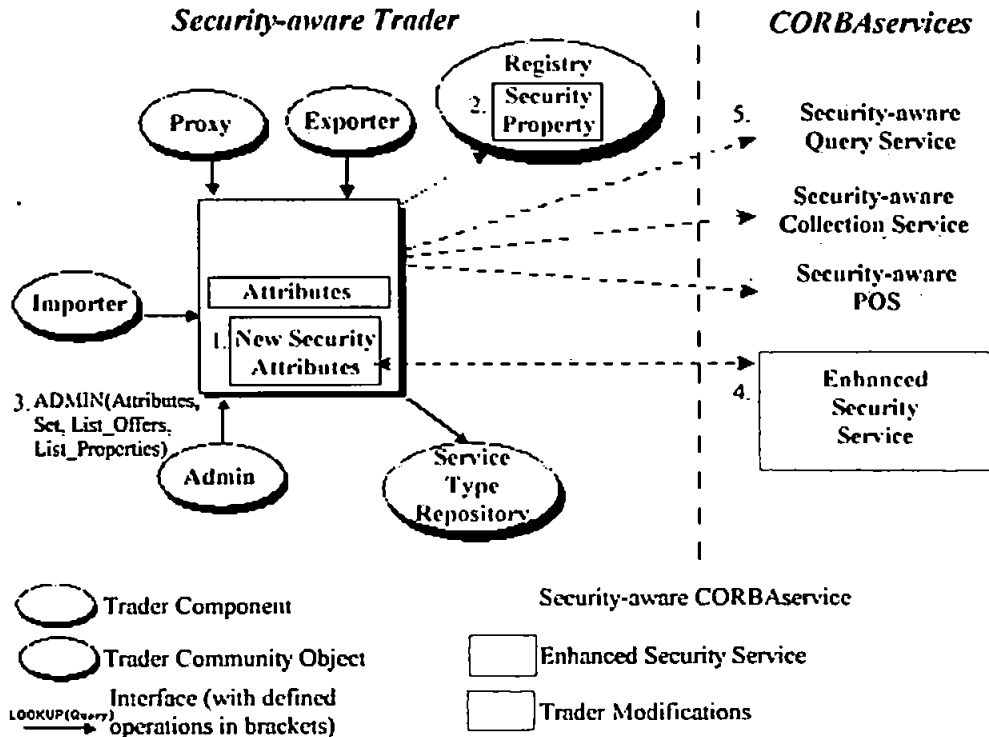


Figure3: Modifications to create a Security-aware Trader

6. Conclusion

In a distributed object system such as the Internet, services could be built using objects. Therefore finding the objects required, local or remote, is pivotal to the success of such an environment. A Trader can do this. However, the Trader provides a very vulnerable point for attack, providing an intruder with access to a multitude of services. Therefore it should be made security-aware. It should be able to ensure that only authorised clients can access it, and that clients can only view the service offers which they are authorised to see. To provide a Security-Aware Trader, modifications have to be made to the CORBA Trader and Security services.

However the Trader was only a detailed example given in this paper, to act as a proof of concept. But other CORBAServices need to be secured, and be part of the TCB, if the OMG is to provide a secure environment, where security administration does not become fragmented and therefore impossible to manage. The bottom line is that security cannot be completely treated as an "add-on" facility. Within CORBA, each CORBAService has to be "aware" of security and able to interact with comprehensive security service.

References

[1] ITU, "ITU X.700 Series – System Management", ITU, <http://www.itu.int>

- [2] OMG Security Working Group, "*OMG White Paper on Security*", Issue 1.0, OMG FTP site, April 1994.
- [3] R. Orfali, D. Harkey, J. Edwards, "*Instant CORBA*", J. Wiley & Sons, 1997.
- [4] R. Resnick, "*Intergalactic Distributed Objects*", Dr.Dobb's SourceBook, January/February 1997.
- [5] M. Bearman, "*Tutorial on ODP Trading Function*", DSTC, University of Canberra, Australia, <http://www.dstc.edu.au>
- [6] D.I. Hopper, "*Destructive ILOVEYOU virus strikes worldwide*", CNN, 4 May, 2000, <http://www.cnn.com>
- [7] P. Chavez, "*55,000 credit card numbers stolen, posted by hacker*", Nandotimes, December 14, 2000, <http://www.nandotimes.com>
- [8] CSI/FBI, "*2000 Computer Crime and Security Survey*", CSI/FBI, December 2000, <http://www.csi.com>
- [9] D. Rodgers, "*Developing Secure, Web-Based Applications*", Software Development Journal, May 1998, <http://www.sdmagazine.com/supplement/ss/feature/s985f2c.shtml>
- [10] The Australian, "*Mobile fraud runs riot*", The Australian, 22 September, 1998.
- [11] E. Leahy, "*Ericsson Fraud Management Solution – FraudOffice*", Ericsson, Business Evolution and Components Seminar, 12 March, 1999.
- [12] OMG, "*OMG RFP5 Submission: Trading Object Service*", OMG Document orbos/96-05-06, Version 1.0.0, May 19 1996,
- [13] CCITT, *Recommendation X.509 "The Directory-Authentication Framework"*, Consultation Committee, International Telephone and Telegraph, International Telecommunications Union, Geneva, 1989.
- [14] National Institute of Standards and Technology (NIST), "*Proposed Federal Information Processing for Digital Signature Standard (DSS)*", Federal Register, v. 56, n. 169, 30 August 1991.

Appendix F – Letters

This appendix present letters of support for the research. The letters are from:

1. Declan O'Sullivan, who acted as an industrial supervisor, from IONA Technologies.
2. Orange

UNIVERSITY OF DUBLIN

Fax: 353 1 6772204
Tel: 353 1 6081765
Telex: 93782 TCD EI



Department of Computer Science
School of Engineering
Trinity College
Dublin 2

Declan O'Sullivan
Lecturer
Computer Science Department
Trinity College Dublin
Dublin 2
Ireland

Prof. Paul Reynolds
Orange PCS
Bradley Stoke
Bristol
BS32 4QJ
UK

With Reference to thesis of Elizabeth Joyce

Dear Paul,

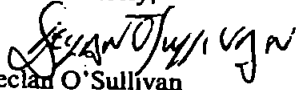
Thank you for forwarding on Elizabeth's thesis for review. It was a pleasure to act as Elizabeth's industrial supervisor in IONA Technologies.

This research is a significant and pragmatic contribution to the area of security and distributed systems. As Elizabeth has quite rightly highlighted, security has been too often been an after thought in system design, and this deficiency is all too painfully being increasingly exposed in value added telecom and internet services.

The framework proposed is impressive, especially since it proposes solutions to a wide set of separate but interlinked problems, namely security components for DPE; security interoperability components; and security aware DPE services. On this later point, the choice of Trader for analysis is I believe particularly welcomed, especially given the emergence of Trader-like services in the wider web services community (e.g. UDDI) which is gathering momentum.

Overall the research has demonstrated in my opinion: a thorough analysis of the problems faced by the DPE community; design and proposal of an innovative framework solution; and a pragmatic approach to proof of concept; leading to a step forward in the state of the art with respect to CORBA, TINA and DPE security.

Yours sincerely,


Declan O'Sullivan



Dr Stephen Furnell
University of Plymouth
Drake Circus
Plymouth
UK

St James Court
Great Park Road
Almondsbury Park
Bradley Stoke
Bristol BS32 4QJ
Phone 01454 624800
Fax 01454 618501
Web Site: www.orange.co.uk

Monday 10 December 2001

Dear Stephen,

Reference Security Service for CORBA

Orange has been experimenting with the use of distributed processing environments for some five years; starting in applications we are currently investigating its use in the transport layer.

It is clear that before CORBA can be used in earnest two things must happen; one it needs to be more scalable, and two, it needs to be more secure. It is the latter that caused us to be involved with Elizabeth Joyce's research.

Whilst Elizabeth has focused upon the development of a generic security service she spends a significant amount of time to understand our, i.e. the mobile operator communities, requirements. She has used these requirements to validate the applicability of her research. Indeed, we are impressed enough with the results she has achieved that we intend to continue the experimentation work within our laboratories.

Orange has been pleased to be associated with her research which we believe has contributed to the State of the Art in security for distributed systems.

Yours sincerely,

A handwritten signature in black ink, appearing to read "Paul Reynolds", with a long horizontal line extending to the right.

Paul Reynolds