E C M A

EUROPEAN COMPUTER MANUFACTURERS ASSOCIATION

STANDARD ECMA - 206

Association Context Management

including Security Context Management

December 1993

ECMA

EUROPEAN COMPUTER MANUFACTURERS ASSOCIATION

STANDARD ECMA - 206

Association Context Management

including Security Context Management

December 1993

Brief history

ECMA, ISO and ITU-T are working on standards for distributed applications in an open system environment. Security and management of the relationship between those applications (association management) is a major concern in information processing.

In July 1988, ECMA TR/46, Security in Open Systems - A Security Framework, was published. Based on the concepts of this framework, ECMA-138, Security in Open Systems - Data Elements and Service Definitions, was produced (December 1989). It defines a set of Security Services for use in the Application Layer of the ISO OSI Reference Model. One of these services, the Secure Association Service, has been expanded and is incorporated in the present work.

Most of the basic aspects of association management have been addressed by ISO in the work on ACSE (ISO 8649).But complex association establishments (e.g. three-ways authentication) or association modifications are not covered by ACSE.

Security aspects of interconnection in the Application Layer are addressed by the Security Exchange Service Element defined in the Generic Upper Layers Security document (ISO/IEC DIS 11586). But this service does neither cover non-security aspects of associations nor propose a model for negotiation of attributes.

This ECMA Standard defines a flexible service and protocol which can manage all aspects of associations, from the simplest to the most complex, between distributed applications. It also defines a generic negotiation model for use by distributed applications.

This ECMA Standard is based on the practical experience of ECMA member Companies. It is oriented towards urgent and well understood needs.

Adopted as an ECMA Standard by the General Assembly of December 1993.

Table of Contents

1 Scope

This Standard:

- Defines a model for management of the characteristics of associations between applications in a distributed system. The associations can be for interactive (e.g. VT) and non-interactive (e.g. FTAM) applications.
- Is a framework to provide achievement of availability, integrity and confidentiality of an association. It does not define how to use specific security mechanisms to protect an association.
- Defines an Association Context Information Model which is the "language" to manage the characteristics of associations.
- Defines Service and Protocol for association context management that meets the requirements for a Secure Association Service as defined in ECMA-138.
- Maps association management to a (non-exclusive) set of application layer protocols: ACSE, ROSE, OSI-RPC.
- Is security policy independent. (e.g. an association might be either application- or system-initiated).
- Supports associations across multiple domains.

This Standard does not:

- Define mechanisms for: authentication, access control, confidentiality, integrity or cryptographic key management.
- Specify how security services like authentication, access control, confidentiality, integrity or cryptographic key management are applied to or initialised in an abstract-association, but it provides the framework to enable these services.
- Specify an Inter-Domain Facility
- Define an API which can service distributed applications to establish associations.
- Enforce Association Context Management, but defines a framework of how to achieve it in a standardised way.

The field of application of this ECMA Standard is the design and implementation of distributed open systems that support access of users to applications and access between distributed applications.

2 Conformance

In order to conform to this Standard, an implementation shall:

a declare which ACM protocol version(s) it supports;

b provide the means to send and receive the following ACM PDUs with the syntax defined in clause 13 of this Standard: ACMinitiateReq, ACMinitiateCompleteReq, ACMreleaseReq, ACMreleaseResp and ACMabortReq;

c declare which of the following optional ACM PDUs it can send and/or receive: ACMinitiateContinueReq, ACMmodifyReq, ACMmodifyContinueReq and ACMmodifyCompleteReq;

d for all PDUs under (b) and (c) above, provide the means to engage in all and only the sequences defined in clause 13 of this Standard;

e for all PDUs under (b) and (c) above, provide the means to encode when sending and to accept when receiving, all those fields which are declared without the ASN.1 keyword OPTIONAL in clause 12 of this Standard;

f for all PDUs under (b) and (c) above, declare which OPTIONAL fields it can encode and/or accept;

g encode and accept all PDUs under (b) and (c) above, and all fields under (e) and (f) above, according to the ASN.1 Basic Encoding Rules (ISO 8825-1);

NOTE
This condition does not preclude the use of other encoding rules as an optional alternative.

h provide a specification in the public domain, or a reference to such a specification, of a mapping of all supported ACM PDUs on to underlying communications services.

NOTE
It is not required that the supported mappings include any of those specified in clause 13 of this Standard.

3 References

ECMA TR/46	Security in Open Systems: A Security Framework (1989)
ECMA-138	Security in Open Systems: Data Elements and Service Definitions (1988)
ECMA-apa	Authentication and Privilege Attribute Security Application with Related Key Distribution Functions (in preparation)
ISO 7498: 1989	Information Processing Systems - Open Systems Interconnection - Reference Model
ISO 8326: 1987	Information Processing Systems - Open Systems Interconnection - Connection Oriented Session Service
ISO 8649: 1988	Information Processing Systems - Open Systems Interconnection - Service Definition for the Association Control Service Element
ISO 8650: 1990	Information Processing Systems - Open Systems Interconnection - Protocol Specification for the Association Control Service Element
ISO 8822: 1988	Information Processing Systems - Open Systems Interconnection - Connection Oriented Presentation Service
ISO 8824: 1993	Information Processing Systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1)
ISO 8824-2: 1993	Information Technology Systems - Open Systems Interconnection - Abstract Syntax Notation One (ASN.1) - Part 2: Information Object Specification
ISO 8825-1: 1993	Information Processing Systems - Open Systems Interconnection - Specification of Encoding Rules for Abstract Syntax Notation One (ASN.1) - Part 1 - Basic Encoding Rules
ISO/IEC 9072: 1989	Information Processing Systems - Open Systems Interconnection - Remote Operation Syntax, Service Definition and Protocol
ISO/IEC 9545: 1989	Information Processing Systems - Open Systems Interconnection - Application Layer Structure
ISO/IEC 9545/DAM 1: 1992	Information Technology - Open Systems Interconnection - Application Layer Structure - Amendment 1: Extended Application Layer Structure
ISO/IEC 9594-8: 1990	The Directory: Authentication Framework (ITU-T Rec. X.509)
ISO/IEC 9595: 1991	Information Processing Systems - Open Systems Interconnection - Common Management Information Service
ISO/IEC 10165-4: 1992	Information Processing Systems - Open Systems Interconnection - Structure of Management Information - Part 4 Guidelines for the Definition of Managed Objects
ISO/IEC 10745: 1993	Information Processing Systems - Open Systems Interconnection - Upper Layer Security Model
ISO/IEC CD 11578	Information Technology - Open Systems Interconnection - Remote Procedure Call Specification. (Four parts).
ISO/IEC DIS 11586-2	Generic Upper Layers Security - Part 2: Security Exchange Service Element (SESE) Service definition

ISO/IEC CD 10746-1	Basic Reference Model of Open Distributed Processing - Part 1: Overview and Guide to Use
ISO/IEC JTC1/SC27 N782	Working Draft - Security Information Objects
Internet RFC 1510	The Kerberos Network Authentication Service (V5) (J. Kohl & C. Neuman - September 1993)
Internet RFC 1508	Generic Security Service Application Program Interface

4 Definitions

For the purpose of this Standard the following definitions apply.

4.1 Imported definitions

The following definitions are imported from ISO 9545:

- application-association
- application-context
- application-entity-invocation
- single-association-object

4.2 New definitions

4.2.1 Abstract-association

A relationship between any two applications in a distributed processing system, in which they co-operate with one another. The term application in this Standard denotes either the initiator side or the responder side.

NOTE 1
This concept must be distinguished from that of an application-association as defined in ISO 9545. An abstract-association is more general than an application-association. An abstract-association may use one or more connections. Also, at times it may have no connection, e.g. because of suspension or error. An abstract-association may use different types of connection, for example: application-associations, OSI-TP dialogues, RPC bindings.

NOTE 2
ISO/IEC 10745, the Upper Layer Security Model, refers to ISO 9545 for a definition of an application-relationship, but ISO 9545 does not define this term. Clause 5.4.6 of ISO 9545 should, however, be noted.

NOTE 3
The term abstract-associations used in this Standard must be distinguished from that of an abstract-association as defined in X.400.

4.2.2 Association-context

The properties of an abstract-association, as currently agreed by the applications participating in the abstract-association. It is realised as a set of **context attributes** (q.v.).

4.2.3 Association Context Management

The functions of initiating, controlling and releasing an abstract-association, by means of operations on its association-context.

4.2.4 Context Attribute

An element of an association-context.

5 Notational Conventions

The ACM protocol is specified in this Standard using Abstract Syntax Notation One (ASN.1) as defined in ISO 8824.

6 Acronyms and Abbreviations

ACM	Association Context Management
ACMSE	Association Context Management Service Element

ACSE	Association Control Service Element
AEI	Application-Entity-Invocation
API	Application Programming Interface
FTAM	File Transfer Access and Management
GSS-API	Generic Security Service API
GULS	Generic Upper Layers Security
ODP	Open Distributed Processing
OSI	Open Systems Interconnection
OSI-TP	OSI Transaction Processing
PAC	Privilege Attribute Certificate
PAS	Privilege Attribute Service
PDU	Protocol Data Unit
ROSE	Remote Operations Service Element
RPC	Remote Procedure Call
SAO	Single Association Object
SESE	Security Exchange Service Element
TMN	Telecommunication Management Network
VT	Virtual Terminal

7 General Description

7.1 Background

Association Context Management (ACM) is one of a number of standards which enable distributed computing, including distributed systems security. Its purpose is to provide one common standard method of communicating context information between applications needing to make an abstract-association.

ACM is applicable for a wide variety of OSI application layer services, including ACSE, ROSE, RPC and OSI-TP. The common ACM PDUs, which contain association attributes, can be transported in different specific application PDUs. The mapping is different for each case.

One particular subset of association attributes relevant to ACM is security attributes. ACM does not specify the security information attributes itself, but can carry a variety of attributes including those defined in ECMA-138, ECMA-apa, ITU-T X.509, ISO Security Information Objects. In this way, ACM is security mechanism independent and security protocol independent. But ACM can *support* a range of security protocols, such as authentication and authorisation during association establishment and negotiation of integrity and confidentiality parameters valid for an established association, by handling of association security attributes.

ACM is aligned to support standard security APIs which are emerging in IETF, POSIX, X/Open (e.g. GSS-API). Thus ACM will complement the APIs and can handle the underlying mechanisms for them. Annex C of this Standard describes the relationship of ACM to GSS-API.

Management of ACM is based on the managed object model as defined in ISO 10165-4 - Guidelines for the Definition of Managed Objects (see annex E).

7.2 Introduction to ACM

This Standard contains the definition of an Association Context Management (ACM) Service. This service is used to establish, control and terminate **abstract-associations** between applications in a distributed system, by negotiating, maintaining and deleting the **association-context** for each such abstract-association.

This ACM Standard provides a Secure Association Service as defined in ECMA-138 by supporting authorisation of abstract-associations and enabling negotiation of additional security facilities for them.

Establishment of abstract-associations allows applications to connect to each other and to negotiate the properties of their abstract-association.

Control of abstract-associations allows applications to modify the properties of their abstract-association.

Termination of abstract-associations provides ways to release established abstract-associations in a controlled manner as well as to interrupt abstract-associations using an abort functionality. Even if abort is used (i.e. there is no proper release of an abstract-association) the abort function is defined so that no security gaps or inconsistencies are generated.

The concept of Association Context Management is independent of supporting facilities and different services as well as of underlying communications concepts that provide a physical or logical link between end systems or applications. For ACM, a mapping of the functionality described in this Standard to the desired communications standards has to be performed. Mappings to some of the most frequently used communications standards are described in clause 13 and annex D of this Standard.

8 Requirements

This clause identifies the requirements which this Standard is intended to satisfy. 8.1 describes general requirements. 8.2 - 8.8 describe requirements identified by considering the ODP aspects (as defined in CD 10746-1).

The list of requirements given below does not claim to be an exhaustive one. This Standard defines an association-context model that allows to meet each of the requirements listed below by definition of the respective context attributes and use of the ACM service primitives.

8.1 General Requirements

8.1.1 Flexible Application

This Standard shall be designed so that it can be used to control characteristics of abstract-associations which are not specified by this Standard. It shall not be necessary to create new versions of or addenda to this Standard in order to meet such requirements.

8.1.2 Non-duplication

Features of other standards, which help to control abstract-associations, shall not be duplicated by this Standard.

8.1.3 Mechanism Independence

This Standard shall be independent of the specific mechanisms that may be used with it (e.g. different authentication procedures, encryption algorithms, etc.).

8.2 Identification Requirements

8.2.1 Peer Entity Identification

This Standard shall enable the applications in an abstract-association to exchange information about their identities.

8.2.2 End User Identification

This Standard shall enable the applications in an abstract-association to exchange information about the identity of the end user(s) for whom they act.

8.3 Security Requirements

This clause contains a non exhaustive list of identified security requirements

8.3.1 Peer Entity Authentication

This Standard shall support transfer of information which will enable the applications in an abstract-association to authenticate one another. (However, the definition of specific kinds of authentication information is outside the scope of this Standard).

8.3.2 End User Authentication

This Standard shall support transfer of information concerning the authentication of the end user. (However, the definition of specific kinds of authentication information is outside the scope of this Standard).

8.3.3 Data Confidentiality

This Standard shall enable the applications in an abstract-association to negotiate the confidentiality parameters to be applied to this abstract-association. (However, the provision of confidentiality services is outside the scope of this Standard).

8.3.4 Data Integrity

This Standard shall enable the applications in an abstract-association to negotiate the integrity parameters to be applied to this abstract-association. (However, the provision of integrity services is outside the scope of this Standard).

8.3.5 Access Control/Authorisation

This Standard shall enable information about access control information to be communicated securely between the applications in an abstract-association. For example, it shall support the transfer of PACs as defined in ECMA-apa - see annex A.

8.3.6 Non-Repudiation of Data

This Standard shall enable information about non-repudiation of data origin and delivery to be communicated between the participants in an abstract-association. (However, the provision of a non-repudiation service is outside the scope of this Standard).

NOTE
Non-repudiation of abstract-associations was also considered but not found to be required.

8.3.7 Association Audit

This Standard shall provide sufficient information about an abstract-association so that the applications can provide audit information on its creation, use and termination. (However, the provision of an audit facility is outside the scope of this Standard).

8.3.8 Availability

The association-context shall remain available to the applications at all times between the establishment and the termination of the abstract-association, even when communications resources are unavailable to support the abstract-association.

8.3.9 Security Recovery

This Standard shall provide the means to perform appropriate recovery actions, for example aborting the abstract-association or modifying the association-context.

8.4 Management Requirements

8.4.1 Policies

This Standard shall recognise that the characteristics of abstract-associations may be subject to system-defined management policies.

8.4.2 Inter-Domain

This Standard shall support abstract-associations which cross the boundaries of different domains, such as security, management or naming domains.

8.4.3 Account Identification

This Standard shall enable the applications in an abstract-association to exchange information about the accounts which are relevant to their abstract-association. Making use of this information is up to the application and therefore not covered by this Standard.

8.5 User Access Requirements

8.5.1 Cultural Adaptation

This Standard shall enable the negotiation of natural language and cultural conventions applicable to an abstract-association.

8.6 Communications Requirements

8.6.1 Combination

ACM shall initialise the facilities of the underlying communications so as to satisfy both its own requirements and those of the applications.

8.6.2 Simplicity

Where ACM selects communications facilities not required by the applications, it shall not require the applications to be aware of them.

8.6.3 Communications Independence

This Standard shall not be aimed at a specific underlying communications method.

8.7 Storage Requirements

No requirements identified.

8.8 Process Requirements

8.8.1 Client-Server Style

This Standard shall support abstract-associations made for client-server interworking (e.g. RPC).

8.8.2 Interactive Applications

This Standard shall support abstract-associations with remote interactive applications.

8.8.3 Proxy/Delegation

This Standard shall enable abstract-associations to be linked for the purpose of forming a delegation chain.

8.8.4 Execution Environment

This Standard shall enable a responder application to be invoked to run in a stated execution environment.

9 Relationship to Other Standards and Recommendations

The ACM Standard relationships with other standards and recommendations are identified under the following main categories:

- Communication
- Security
- Management (including Security Management)
- Operating Systems and APIs
- Architecture

9.1 Communications Standards

ACM will be mapped on to, and thus use services of ACSE, ROSE, OSI-TP, and OSI-RPC. Some of the required mappings are described in clause 13 of this Standard.

9.2 Security Standards

ACM will support the exchange of security information objects and services defined in ECMA-138, ECMA-apa, ITU-T X.509, ISO Security Information Objects, GULS.

Standards defining specific security functions will make use of ACM to initialise and control their parameters.

9.3 Management Standards

Systems implementing ACM services and information objects can be managed by implementing OSI Management or ITU-T TMN.

9.4 Operating Systems and API Standards

ACM is aligned with standards for distributed security APIs currently emerging from POSIX, X/Open and IETF.

9.5 Architecture Standards

ACM will use concepts from ECMA TR/46, ECMA-138, ISO 9595 and ISO 10745.

10 Models

10.1 Associations and Contexts

This Standard defines an abstract-association to be a relationship between two communicating applications. This abstract-association is supported by the association-context, which is used to manage the abstract-association. The association-context itself consists of context attributes that are used to establish, maintain and release abstract-associations. Association Context Management as defined in this Standard addresses management of different association-contexts.

The nature and use of the association-context is illustrated in figure 1, which shows that the context has several parts, including that of security. Association Context Management provides the facilities to set up, modify and delete the contents of the context, and to negotiate these between the two sides. Security services access the security part of the association-context in order to control their operation, and other services use other parts of the association-context in similar ways. This diagram shows how Association Context Management is distinguished from other services although possibly made available through the same interface standards.

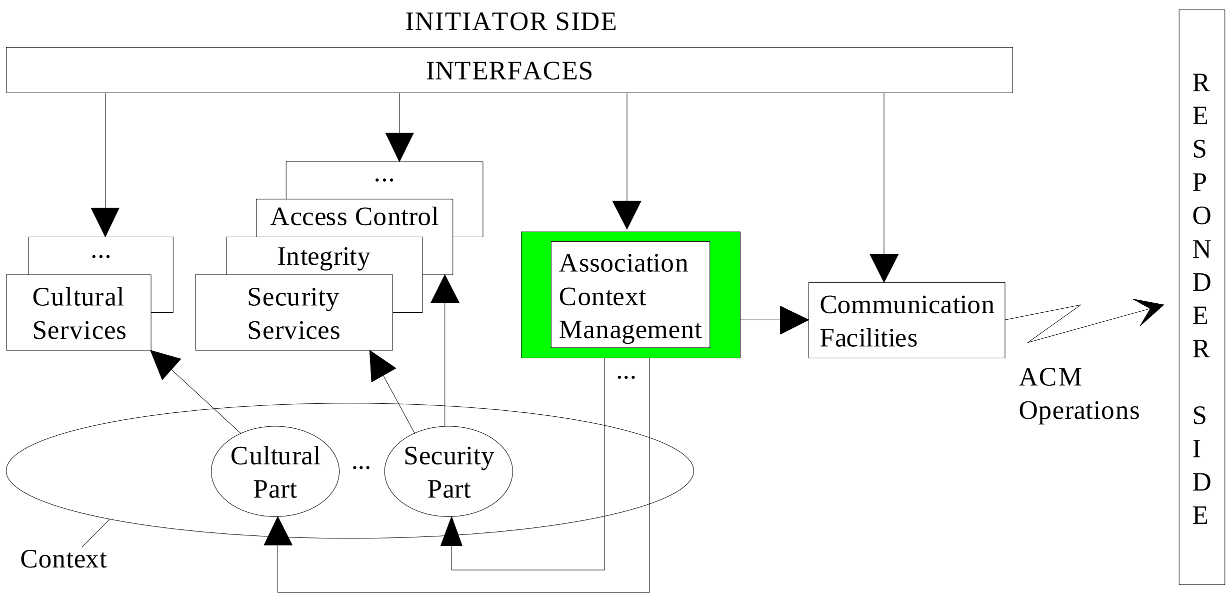

Figure 1 - Management and Use of Association-Context

NOTE
There are many different possibilities to integrate Association Context Management in the application layer, depending on the implementation option chosen. ACM could for instance be implemented as a separate "layer" above the communication facilities, or it could be implemented underneath an interface which simply returns a complete ACM PDU. In the latter case the ACM PDU is transferred to the responder side through a specific call of the communication facilities interface. The existence of the arrow between ACM and the communication facilities therefore depends on the actual ACM implementation.

A typical association-context consists of a set of attributes, providing the means to ensure that the desired properties (e.g. integrity or confidentiality) can be achieved on an abstract-association. Different qualities can be supported by the same or by different attributes. The complete context representing an abstract-association does not only consist of attributes that provide security features, but can also include different attributes (e.g. to support cultural conventions). The set of attributes applicable to a given abstract-association will generally be a subset of all those attribute types that have been specified. The subset will generally be selected from an extensible set of registered attribute types which will be defined in a wide variety of standards. Applications that might want to use private attribute types should register these before use.

An application that is going to establish an abstract-association must specify the desired properties of an abstract-association. Therefore it chooses attribute values to make up the corresponding association-context. The attribute values may be determined or constrained by other factors such as security policy, technical limitations, etc. In case of association across multiple domains, the attribute values of an association-context may be constrained by an inter-domain service.

10.2 Levels of Context

There are three levels of context known to an application-entity-invocation, as illustrated in figure 2. The outer level defines the context of the application-entity-invocation as a whole (made of several single-association-objects). The middle level contains a context for each abstract-association as defined in this Standard. The inner level contains a context for each communications instance in which the application entity invocation is involved. The association-context corresponds to the middle level in this structure.

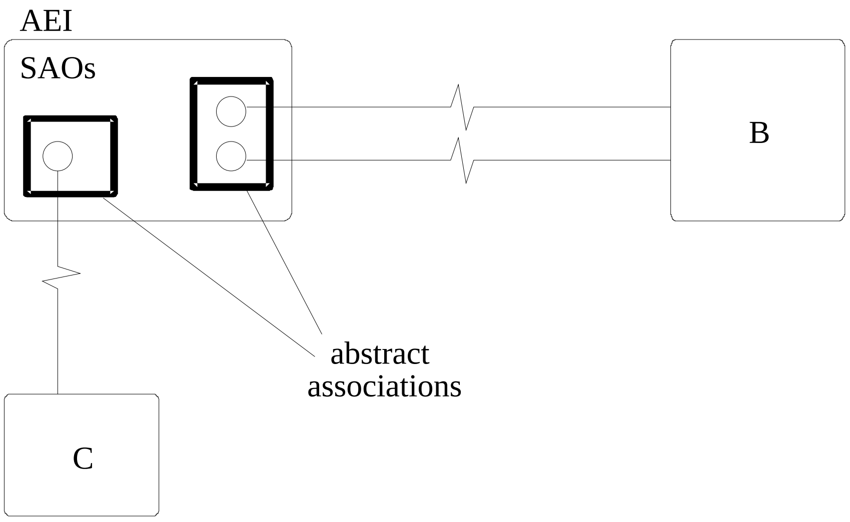

Figure 2 - Three Levels of Context

10.3 Operations on the Context

The ACM Service supports the following operations:

Initiate: Initialises the abstract-association and creates an association-context at the initiator and at the responder side.

Release: Orderly release of an established abstract-association, deletes the association-context at the initiator and at the responder side.

Abort: Aborts an established abstract-association, deletes the association-context at the initiator and at the responder side.

Modify: Modifies the association-context by adding, deleting and modifying context attributes and their values at the initiator and at the responder side.

10.4 Abstract-Association Negotiation Model

The initiator and responder sides of an abstract-association can negotiate about the association-context whenever abstract-associations are established or modified. Nevertheless, not all context attributes are suitable for negotiation purposes. Some context attributes can be sent without the need of an acknowledgement (e.g. the last parameter of a three-way authentication exchange).

The negotiation of context attributes takes place as follow: one side proposes the context attributes, the other side either starts further exchanges or directly agrees or directly disagrees.

When the second side starts a further exchange, it can add its own proposals, to which the first side in turn responds. This continues until the negotiations are completed.

In case of agreement, the responder side accepts the association-context. If the initiator offers several values for a context attribute, the response message contains the appropriate choice of the responder side.

In case of disagreement, the responder side rejects the association-context. In doing so, it can suggest different context attribute values that would have been more appropriate. To initialise a context with those attributes, the initiator side has to start again with a new offer.

Any offer can be based on the result of previous negotiation steps, on a request from the other side or on information about the other side that is available by any other means (e.g. from a directory). Those three methods are (or any combination of them is) supported by this Standard. Nevertheless, the specification of methods to collect attributes information for the construction of abstract-association is out of the scope of ACM.

10.5 Nested Association-contexts

When establishing a new association-context, an application may wish to include attributes of an already existing association-context. Therefore, it can refer, within a context attribute, to the context identifier of the already established association-context. This context identifier is created during the first exchange of the abstract-association initialisation.

ACM does not provide any mechanisms to maintain a relationship between association-contexts, as ACM is not aware of it.

10.6 Mapping of ACM to communication standards

The model of ACM is independent of the communication infrastructure used. ACM can be mapped to several ISO communication standards like: ACSE, ROSE, OSI-TP or OSI-RPC.

As different communication standards do not completely support the model of ACM exchanges, a full mapping is not always possible (the mapping to ACSE for instance, requires also use of the P-DATA service primitive).

The use of communication standards that restrict their scope to certain area (or specific parts) of an association-context is only possible when no other context attributes shall be supported by ACM. A mapping of ACM to SESE (as defined in CD 11586) therefore requires all context attributes to be security related. More details and some of the protocol mappings are given in chapter 13.

10.7 Management of Association Context Management

Management of ACM is modelled according to the OSI management framework (ISO 7498-4). It consists of creation, control and deletion of ACM managed objects and of distribution and collection of information about ACM by a third party not involved in the abstract-association (e.g. an administrator or a system operator).

This control includes monitoring of certain context attributes, receiving notification about ACM related operations and performing all actions necessary to maintain proper ACM operation.

10.7.1 Global Description

Management of ACM must not be confused with the normal ACM functionality. Two different kinds of ACM actions are distinguished:

- Functional actions *of* ACM:
  These are the actions that directly correspond to the establishment, maintenance or release of abstract-associations on behalf of applications. They only affect ACM and the two applications involved in an abstract-association. These operations are explained in detail in chapter 11.

- Management actions *on* ACM:
  These are actions that are not triggered by one of the applications involved in an abstract-association but by a manager of ACM.

All management actions on ACM are reflected in the Management Information Base (MIB) which includes the complete set of managed objects (MOs) that are relevant to ACM.

10.7.2 Management Functions on ACM Managed Objects

Managed objects in terms of ISO standards are management views of network resources. The objects that ACM deals with are abstract-associations. The obvious ACM managed objects therefore are the abstract-associations from a management point of view. ACM management includes features like monitoring abstract-associations or collecting information on specific abstract-associations.

Potentially, Association Managers are managed objects - this is implementation dependent. Therefore this Standard does not specify an "Association Manager" managed object class. If this is done, then the abstract-association managed object should be modelled as being contained in the Association Manager managed object.

ACM management enables an administrator to monitor current abstract-associations and to delete any which are unacceptable. For consistency reasons, modification of abstract-associations should not be performed by an ACM administrator. Management of ACM is supported by notifications sent to the manager whenever an abstract-association managed object is created, modified or deleted. These notifications may be triggered by functional use of ACM service primitives.

An abstract-association is managed by performing management operations on its attributes. The attributes of the abstract-association managed object are defined by the ACM implementation. They may include security attributes as well as other attributes such as cultural conventions or an abstract-association identifier.

The naming attributes of abstract-association managed objects are inherited from the managed object class **"top"** defined in ISO 10165-4. The abstract-association managed object includes notification for:

- managed object creation
- managed object modification
- managed object deletion

Further description of management information for ACM can be found in annex E of this Standard.

10.7.3 Relationship between ACM and other Management Functions

There are several objects in the area of ACM that are related to other management functions:

- parameters that are associated with the communications infrastructure (e.g. cryptographic algorithm or key length) can be managed directly at the infrastructure management interface (if provided).
- the definition of managed objects for ACM must not restrict any potential ACM implementations by specifying detailed ways how to manage specific counters and values. Therefore this Standard does not identify any parameters (like time-out periods or number of retries) that are recognised to be implementation dependent.
- abstract-association establishment, modification, release and abort is topic of ACM functionality and therefore not covered by management of ACM.
- defining and adding new attribute types is a specification activity rather than a management activity and is therefore not part of the management of ACM.

11 Service Description

The Association Context Management Service Element (ACMSE) offers the following facilities realised by the associated service primitives:

Initiate-Context facility

– ACM-Initiate	(unconfirmed)
– ACM-Initiate-Continue	(unconfirmed)
– ACM-Initiate-Complete	(unconfirmed)

Release-Context facility

– ACM-Release	(confirmed)

Modify-Context facility

– ACM-Modify	(unconfirmed)
– ACM-Modify-Continue	(unconfirmed)
– ACM-Modify-Complete	(unconfirmed)

Abort-Context facility

– ACM-Abort	(unconfirmed)

NOTE
These service primitives are based on the concept of the Secure Association Service as developed in ECMA-138.1

11.1 Model of Multiple Exchanges

In order to support application protocols that require more than a two-way exchange (e.g. 3-way authentication), ACM specifies unconfirmed service primitives to perform the establishment and the modification of abstract-associations (see above).

11.1.1 Model of Initiate Exchanges

The process of initiating an abstract-association starts with an ACM-Initiate followed by zero or more ACM-Initiate-Continues in alternate directions and ends with an ACM-Initiate-Complete, which can come from either side.

A detailed description of the initiation process for abstract-associations is as follows:

Step 1:

The initiator (client) side starts by making an ACM-Initiate request. The associated PDU proposes in context parameters context attributes for the abstract-association that is to be established. Additional context parameters can contain data which are used in the first step of any specialised protocol (for example, in case of an 3-way authentication protocol, the first challenge).

Step 2:

Depending of the number of exchanges required in the application protocol (e.g. 3 steps to achieve mutual authentication) the other side may now perform one of the following steps:

Step 2a (if 2 or less exchanges are required):

make an ACM-Initiate-Complete request and therefore end the abstract-association establishment.

or

Step 2b (if more than 2 exchanges are required):

make an ACM-Initiate-Continue request in order to indicate that there is at least one more step to be performed.

Step 2b can be repeated in alternating directions until all exchanges necessary to establish the abstract-association have taken place.

Step 3:

The side that received the final ACM-Initiate-Continue indication answers with an ACM-Initiate-Complete request, thereby completing the establishment of the abstract-association.

11.1.2 Model of Modify Exchanges

The process of modifying an abstract-association starts with an ACM-Modify followed by zero or more ACM-Modify-Continues in alternate directions and ends with an ACM-Modify-Complete, which can come from either side. The detailed sequence of exchanges is performed analogously to the description in 11.1.1.

11.2 Parameters for negotiating the association-context

The ACM-Initiate, ACM-Initiate-Continue, ACM-Initiate-Complete, ACM-Modify, ACM-Modify-Continue and ACM-Modify-Complete services provide facilities for the service users to initialise and modify the association-context. This is done in each service primitive by specifying parameters that are used to propose an association-context initialisation or modification, and to reply to such a proposal. The ACM-Initiate service primitive has parameters to build the initial association-context, while ACM-Initiate-Continue, ACM-Initiate-Complete, ACM-Modify, ACM-Modify-Continue and ACM-Modify-Complete can be used to operate on an association-context already existing.

The following parameters are used for this purpose:

- context-parameter: for suggesting new or modified attributes in the association-context or for asking the other side a context parameter
- negotiation-parameter: to reply to a context-parameter

To remove one or more values from a multi-valued context-attribute, a context-parameter is used to set the value set of the attribute to the respective subset. To disable a context attribute, a context-parameter is used to set its value to "NULL". Context-attributes that only have the "NULL" value do not influence the abstract-association any more.

To ask the other side to send a context parameter, the value of the context parameter may be used to indicate what properties are requested.

The context-parameter consists of

- cpRef (context-parameter Reference): this is a number given to the parameter by the issuer to enable recognition of a responding negotiation-parameter
- cpType (context-parameter Type): an OBJECT IDENTIFIER for the parameter type
- cpValue (context-parameter Value): this is used either:
  - to suggest a value for the context-attribute, or
  - to offer a set of values for the partner to select from, or
  - to propose to disable a context-attribute with the NULL value as described above, or
  - to describe properties required for the context parameter requested to the other side.

The negotiation-parameter consists of:

- cpRef (context-parameter Reference): the same number as in the context-parameter that the negotiation-parameter responds to
- npId (negotiation-parameter Identifier): a number used either to indicate that a value for a context-parameter is selected from a set of offered values or to say what aspect of the proposed context-parameter is not accepted. Two special values are assigned:
  - npId = -1 indicates that the context-parameter is not acceptable at all.
  - npId = 0 indicates that the negotiation-parameter contains a selected value for a cpValue.
- npValue (negotiation-parameter Value): this is used either:
  - to choose from different values offered in a selection list of the context-parameter received, or
  - to indicate what is unacceptable about the received context-parameter.

  When npId = -1 (the context-parameter is rejected), npValue is "NULL".

  When npId = 0 (a cpValue is selected), the syntax of npValue is the syntax of the related context-parameter value.

In order to reject a context-parameter for multiple reasons, these reasons are combined in a single negotiation-parameter.

An illustration of this system is given in annex A.

11.3 How to design and implement context parameters

Designers of distributed applications that will use ACM, and designers of facilities to support aspects of distributed applications such as their security or their internationalisation, can plan to make effective use of ACM by the following method.

Firstly, designers should identify those aspects of their design on which they require the distributed applications to agree on. These are the context attributes that will be set up (and modified, if required) using ACM. At this stage, the context attributes can be identified without worrying about packaging them together into context-parameters and negotiation-parameters which map to the ACM service and protocol - that is the next stage.

When designers have defined the required context attributes at a logical level, they are in a position to decide how to package them for negotiation purposes. It is not necessary to package them - in some cases one context attribute may give rise to one context-parameter. But sometimes it is desired to package them together, for example if they are to be transmitted together under a common cryptographic seal. In such cases several context-attributes may be brought together in one context-parameter.

NOTE
The construction of packages is task of the ACM user. Applications that access ACM through a high-level interface (e.g. GSS-API) do not have to be aware of these packaging activities.

At this stage, designers should specify the following things about each context-parameter:

* An **OBJECT IDENTIFIER** (cpType) by which context-parameters of that type will be globally recognisable

* The **value syntax** of the values for context-parameters of this type. (This Standard places no restriction on the syntax)

* A numbered **list of negotiable properties** or aspects of context-parameters of this type. Each negotiable property is expressed by a negotiation parameter. Each entry in the list should state:
  - A **property id** for this negotiable property (npId). This should be an integer, starting at 1, such that each property is given a unique property id within the scope of this type of context-parameter
  - The **property syntax**, which is the syntax for the part of the negotiation-parameter which actually states the selected value or expresses the problem with the context-parameter.

An implementor should begin from the syntax for context-parameters and negotiation-parameters defined in clause 12 of this standard. He should arrange to generate the 'cpRef' reference number of each context-parameter at run time. All the other fields in these two types are handled by substituting the values from the specification described above into the ACM data types defined in clause 12.

An example of this design procedure, referring to the ECMA Privilege Attribute Certificate, is given in annex A.

11.4 Key to parameter tables

The key to the parameter tables in the following clauses is:

req	request primitive
ind	indication primitive
rsp	response primitive
cnf	confirmation primitive
M	Mandatory parameter, must be supplied by the service user
O	Presence of the parameter is an option for the service user
C	Presence of the parameter is subject to conditions stated in the text
=	The parameter is the same as in the previous column of the table

11.5 ACM-Initiate

Begins the initialisation of the abstract-association and proposes an initial state of the association-context.

	req	ind
underlying-service-params	C	C
context-parameters	M	=
initiator-context-id	O	=

11.5.1 Parameters

underlying-service-params: depending on the implementation option chosen for ACM (see note 7), there might be the need for the ACM user to pass parameters to the communication services that ACMSE is mapped on. Different services (e.g. ACSE, OSI-RPC etc.) will require different parameters, which are of no concern to ACMSE. Please refer to the relevant standards.

context-parameters: used as described in 11.2 to propose the initial state of the association-context.

initiator-context-id: identifier chosen by the initiator of the ACM-Initiate primitive to refer to the context being created.

11.5.2 Sequences

See 11.7.2.

11.6 ACM-Initiate-Continue

Continues the initialisation of the abstract-association and carries extra information on context attributes useful to create an association-context at the initiator and at the responder side.

	req	ind
underlying-service-params	C	C
negotiation parameters	C	=
context-parameters	O	=
initiator-context-id	C	=
responder-context-id	C	=

11.6.1 Parameters

underlying-service-params: depending on the implementation option chosen for ACM (see note 7), there might be the need for the ACM user to pass parameters to the communication services that ACMSE is mapped on. Different services (e.g. ACSE, OSI-RPC etc.) will require different parameters, which are of no concern to ACMSE. Please refer to the relevant standards.

negotiation parameters: used as described in 11.2 to select from alternatives offered in the context-parameters of the immediately preceding ACM-Initiate or ACM-Initiate-Continue indication primitive.

context-parameters: used as described in 11.2 to propose extra information on the initial state of the association-context. This parameter is for use in proposing further attributes beyond those already negotiated.

initiator-context-id: identifier chosen by the initiator of the ACM-Initiate primitive to refer to the context being created.

responder-context-id: identifier chosen by the responder to the ACM-Initiate primitive to refer to the context being created.

The parameters *initiator-context-id* and *responder-context-id* are only present in the first ACM-Initiate-Continue sent after a ACM-Initiate and when *initiator-context-id* was present in the ACM-Initiate primitive.

11.6.2 Sequences

See 11.7.2.

11.7 ACM-Initiate-Complete

Completes the initialisation of the abstract-association and creates, in case of success, an association-context at the initiator and at the responder side.

	req	ind
underlying-service-params	C	C
negotiation parameters	C	=
context-parameter	O	=
initiator-context-id	C	=
responder-context-id	C	=
result	M	=
reject-reason	C	=

11.7.1 Parameters

underlying-service-params: depending on the implementation option chosen for ACM (see note 7), there might be the need for the ACM user to pass parameters to the communication services that ACMSE is mapped on. Different services (e.g. ACSE, OSI-RPC etc.) will require different parameters, which are of no concern to ACMSE. Please refer to the relevant standards.

negotiation-parameters: used as described in 11.2 to accept or reject the initial state of the association-context. This parameter is included in this service primitive if, and only if:

– the responder accepts the association and choices of attribute values which were offered in the immediately preceding ACM-Initiate or ACM-Initiate-Continue indication primitive, **or**

– the responder rejects the association and the reject-reason is "see returned parameters".

context-parameters: used as described in 11.2 to propose extra information on the initial state of the association-context. This parameter is included in this service primitive if, and only if:

– The responder accepts the association. Then, this context parameter cannot be negotiable; it can only be used to carry context information which does not need to be acknowledged.

– The responder rejects the association and the reject-reason is "see returned parameters". Then this parameter carries context-parameters that were not proposed and that would have been appropriate to propose.

initiator-context-id: identifier chosen by the initiator of the ACM-Initiate primitive to refer to the context being created.

responder-context-id: identifier chosen by the responder to the ACM-Initiate primitive to refer to the context being created.

The parameters *initiator-context-id* and *responder-context-id* are only present in a positive ACM-Initiate-Complete, when no ACM-Initiate-Continue was previously sent and when *initiator-context-id* was present in ACM-Initiate (see 10.5).

result: indicates whether the responding service user accepts or rejects the proposed abstract-association. Values are:

- accept
- reject

reject-reason: shall be included if result = reject, otherwise omitted. Values are:

- no reason given
- context not authorised
- see returned parameters
- no common ACMSE version

11.7.2 Sequences

11.7.2.1 Two-way initialisation of abstract-association

The following shows the sequence of actions performed by both sides when an abstract-association is successfully established with two messages.

Initiator		**Responder**
Authorise abstract-association		
Preliminary definition of the context		
Make ACM-Initiate.req	>>>>>>	Receive ACM-Initiate.ind
		Decide to accept the context
		Initialise association-context
		Invoke the application
Receive ACM-Initiate-Complete.ind	<<<<<<	Make ACM-Initiate-Complete.req
Initialise the association-context		

11.7.2.2 Three-way Initialisation of abstract-association

The following shows the sequence of actions performed by both sides when an abstract-association is successfully established with three messages.

Initiator		Responder
Authorise abstract-association		
Preliminary definition of the context		
Make ACM-Initiate.req	>>>>>>	Receive ACM-Initiate.ind
		Decide to continue the initialisation
		Further definition of the context
Receive ACM-Initiate-Continue.ind	<<<<<<	Make ACM-Initiate-Continue.req
Decide to accept the context		
Initialise the association-context		
Make ACM-Initiate-Complete.req	>>>>>>	Receive ACM-Initiate-Complete.ind
		Initialise the association-context
		Invoke the application

11.8 ACM-Release

Releases an established abstract-association, deletes the association-context at the initiator and at the responder side.

	req	ind	rsp	cnf
release-reason	O	=		
result			M	=
reject-reason			C	=

11.8.1 Parameters

release-reason: the reason for requesting the release of the abstract-association. Values are:

- no reason given
- normal release (end of work)
- context modifications not agreed

result: indicates whether the responding service user agrees to terminate the abstract-association or not. Values are:

- accept
- reject

reject-reason: the reason for rejecting the release of the abstract-association. Values are:

- no reason given
- external dependencies: this context cannot be released as its existence depends on existing external dependencies (e.g. other related contexts).

11.8.2 Sequences

11.8.2.1 Successful release of the abstract-association

The following shows the sequence of actions performed by both sides when an abstract-association is successfully released. The "initiator" in this case is the initiator of the release service, and may (subject to rules of the application context) be either end of the association.

Initiator		**Responder**
Make ACM-Release.req	>>>>>>	Receive ACM-Release.ind
		Inform the application
		Delete the association-context
Receive ACM-Release.cnf	<<<<<<	Make ACM-Release.rsp
Delete the association-context		

11.8.2.2 Collision of release requests

In order that there shall be no ambiguity when ACM-Release requests collide, the release request from the initiator of the abstract-association shall be processed and the one from the responder of the abstract-association shall be discarded. The resulting sequence is shown below.

Abstract-association initiator		**Abstract-association responder**
Make ACM-Release.req	-- --	Make ACM-Release.req
	\/	
	/\	
Receive ACM-Release.ind	<- ->	Receive ACM-Release.ind
(discard it)		Inform the application
		Delete the association-context
Receive ACM-Release.cnf	<<<<<<	Make ACM-Release.rsp
Delete the association-context		

11.9 ACM-Abort

This is an unconfirmed primitive. Aborts an established abstract-association.

	req	ind
abort-reason	O	=

11.9.1 Parameters

abort-reason: the reason for requesting the abnormal termination of the abstract-association. Values are:

- no reason given

NOTE
This parameter has been included in order to ensure compatibility with future versions of this Standard, which may specify further values.

11.9.2 Sequences

The following shows the sequence of actions performed by both sides when an abstract-association is aborted. The "initiator" in this case is the initiator of the abort service, and may (subject to rules of the application context) be either end of the association.

Initiator		**Responder**
Delete the association-context		
Make ACM-Abort.req	>>>>>>	Receive ACM-Abort.ind
		Inform the application
		Delete the association-context

11.10 ACM-Modify

The modify service can be used by both initiating and responding application. There are different groups of modifying actions.

1 Overwrite or disable a context attribute
The context of an existing context attribute is changed by replacing a context attribute, e.g. changing cultural conventions from British to Spanish ones.

2 Add a new context attribute to the association-context:
The association-context is extended by adding new context attributes, e.g. new access control information are needed by the application.

3 Ask the other side to send a new context parameter
The initiator asks the responder to send a new context parameter, for instance to update an expired context attribute.

	req	ind
context-parameters	M	=

11.10.1 Parameters

context-parameters: used as described in 11.2 to propose modifications to the association-context.

11.10.2 Sequences

See 11.12.2.

11.11 ACM-Modify-Continue

Continues the modification of the abstract-association and carries extra information on context attributes useful to modify an association-context at the initiator and at the responder side.

	req	ind
negotiation parameters	C	=
context-parameters	C	=

11.11.1 Parameters

negotiation parameters: used as described in 11.2 to select from alternatives offered in the context-parameters of the immediately preceding ACM-Modify or ACM-Modify-Continue indication primitive.

context-parameters: used as described in 11.2 to propose extra information on the state of the association-context.

11.11.2 Sequences

See 11.12.2.

11.7 ACM-Modify-Complete

Completes the modification of the abstract-association and modifies, in case of success, the association-context at the initiator and at the responder side.

	req	ind
negotiation parameters	C	=
context-parameter	O	=
result	M	=
reject-reason	C	=

11.12.1 Parameters

negotiation-parameters: used as described in 11.2 to accept or reject the state of the association-context. This parameter is included in this service primitive if, and only if:

– the responder accepts the modifications of the association and choices of attribute values which were offered in the immediately preceding ACM-Modify or ACM-Modify-Continue indication primitive, **or**

– the responder rejects the modifications and the reject-reason is "see returned parameters" or "inconsistent modifications"

context-parameters: used as described in 11.2 to propose extra information on the state of the association-context. This parameter is included in this service primitive if the responder accepts the modifications of the association. Then, this context parameter cannot be negotiable; it can only be used to carry context information which does not need to be acknowledged.

result: indicates whether the responding service user accepts or rejects the proposed changes to the association-context. Values are:

- accept
- reject

reject-reason: the reason for rejecting the release of the abstract-association. Values are:

- no reason given
- context not authorised
- see returned parameters
- inconsistent modifications (i.e. the proposed modifications would lead to an inconsistent association-context)
- modification collision (see 11.12.2.2)

11.12.2 Sequences

11.12.2.1 Successful two-way modification of the association-context

The following shows the sequence of actions performed by both sides when an association-context is successfully modified in a two-way exchange. The "initiator" in this case is the initiator of the modify service, and may (subject to rules of the application context) be either end of the association.

Initiator		**Responder**
Preliminary change to the association-context		
Make ACM-Modify.req	>>>>>>	Receive ACM-Modify.ind
		Authorise modifications
		Change the association-context
		Inform the application
Receive ACM-Modify-Complete.ind	<<<<<<	Make ACM-Modify-Complete.req
Change the association-context		

11.12.2.2 Collision of modify requests

In order that there shall be no ambiguity when ACM-Modify requests collide, the modifications proposed by the initiator of the abstract-association shall be processed and those proposed by the responder of the abstract-association shall be discarded.

NOTE
Having responded to the initiator's ACM-Modify request, the responder may then send a new ACM-Modify request, proposing further changes to the association-context.

Abstract-association initiator		**Abstract-association responder**
Make ACM-Modify.req	-- --	Make ACM-Modify.req
	\/	
	/\	
Receive ACM-Modify.ind	<- ->	Receive ACM-Modify.ind
(discard it)		Authorise modifications
		Change the association-context
		Inform the application
Receive ACM-Modify-Complete.ind	<<<<<<	Make ACM-Modify-Complete.req
Change the association-context		

11.12.2.3 Collision of modify and release requests

When a ACM-Modify request collides with a ACM-Release request, the release request shall be processed and the modify request shall be discarded, as shown in the sequence below.

NOTE
It is possible to respond negatively to the ACM-Release request and then to send a new ACM-Modify request. The following diagram does not illustrate this, but neither does it preclude it.

```
                            A               B

        Make ACM-Release.req     --  --    Make ACM-Modify.req
                                   \/
                                   /\
      Receive ACM-Modify.ind     <-  ->    Receive ACM-Release.ind
                  (discard it)              Inform the application
                                            Delete the association-context
     Receive ACM-Release.cnf    <<<<<<     Make ACM-Release.rsp
 Delete the association-context
```

12 Protocol Abstract Syntax

```
ACMSE {1 3 0012 0 999} DEFINITIONS IMPLICIT TAGS ::= BEGIN
   EXPORTS ALL

   -- ************ PDU TYPES ************
   ACMinitiateReq ::= [APPLICATION 0] SEQUENCE {
          protocolVersion       [0] BIT STRING
                                      {version1 (0) }
                                      DEFAULT version1,
            contextParams       [1] SET OF ContextParam,
         initiatorContextId     [2] INTEGER OPTIONAL }

   ACMinitiateContinueReq ::= [APPLICATION 1] SEQUENCE {
         negotiationParams      [0] SET OF NegotiationParam OPTIONAL,
            contextParams       [1] SET OF ContextParam OPTIONAL,
         initiatorContextId     [2] INTEGER OPTIONAL,
         responderContextId     [3] INTEGER OPTIONAL }

   ACMinitiateCompleteReq ::= [APPLICATION 2] SEQUENCE {
                       result   [0] Result,
                       reason   [1] InitRejectReason OPTIONAL,
         negotiationParams      [2] SET OF NegotiationParam OPTIONAL,
            contextParams       [3] SET OF ContextParam OPTIONAL,
         initiatorContextId     [4] INTEGER OPTIONAL,
         responderContextId     [5] INTEGER OPTIONAL }

   ACMreleaseReq ::= [APPLICATION 3] SEQUENCE { [0] ReleaseReason OPTIONAL }
```

```
ACMreleaseResp ::= [APPLICATION 4] SEQUENCE {
                        result      [0] Result,
                        reason      [1] ReleaseRejectReason OPTIONAL }

ACMabortReq ::= [APPLICATION 5] SEQUENCE { [0] AbortReason OPTIONAL }

ACMmodifyReq ::= [APPLICATION 6] SEQUENCE {
            contextParams       [0] SET OF ContextParam }

ACMmodifyContinueReq ::= [APPLICATION 7] SEQUENCE {
        negotiationParams       [0] SET OF NegotiationParam OPTIONAL,
            contextParams       [1] SET OF ContextParam OPTIONAL }

ACMmodifyCompleteReq ::= [APPLICATION 8] SEQUENCE {
                        result      [0] Result,
                        reason      [1] ModifyRejectReason OPTIONAL,
        negotiationParams       [2] SET OF NegotiationParam OPTIONAL,
            contextParams       [3] SET OF ContextParam OPTIONAL }

-- *********** GENERAL PARAMETER DATA TYPES ************
    Result ::= ENUMERATED {
                        accept (0),
                         reject (1) }

     InitRejectReason ::= ENUMERATED {
                        noReasonGiven (1),
                        contextNotAuthorised(2),
                        seeReturnedParameters (3),
                        noCommonACMSEVersion (4) }

    ReleaseReason ::= ENUMERATED {
                        normalRelease (0),
                        noReasonGiven (1),
                        contextModificationsNotAgreed (2) }
    ReleaseRejectReason ::= ENUMERATED {
                    noReasonGiven (1),
                    externalDependencies (2) }

    ModifyRejectReason ::= ENUMERATED {
                    noReasonGiven (1),
                    contextNotAuthorised(2),
                    seeReturnedParameters (3),
                    inconsistentModifications (4),
                    modificationCollision (5) }

    AbortReason ::= ENUMERATED {
                    noReasonGiven (1) }

-- ********** DATA TYPES DESCRIBING THE ASSOCIATION CONTEXT **********
    ContextParam ::= SEQUENCE {
                     cpRef     [0] INTEGER OPTIONAL,
                    cpType     [1] OBJECT IDENTIFIER,
                   cpValue     [2] CHOICE {
                                   proposed    [0] ANY DEFINED BY cpType,
                                    offered    [1] SEQUENCE OF ANY DEFINED BY cpType,
                                   deleteIt    [2] NULL,                 -- in ACM-Modify
                                     sendIt    [3] SET OF SEQUENCE {     -- NULL if no properties specified
                                                     npId    [0] INTEGER,
                                                  npValue    [1] ANY -- defined by npId and cpType}
                  } }

    NegotiationParam ::= SEQUENCE {
                     cpRef     [0] INTEGER,
                   details     [1] SET OF SEQUENCE {
                                       npId    [0] INTEGER,   --  -1 if cpValue is rejected
                                                              --   0 if npValue contains selected value for
                 cpValue
                                    npValue    [1] ANY        -- defined by npId and cpType
                                                              -- NULL if npId = -1
                                                              -- cpValue syntax if npId = 0
                  } }

END
```

13 Protocol Mappings

13.1 Introduction and Principles

Arising from the large number of communications standards, there is the requirement that ACM must not be dependent on a specific infrastructure.

This clause provides guidelines how a mapping of the ACM protocol to different communication infrastructure standards can be performed. Caused by the variety of application protocols above ACM and by different features of the underlying communications protocol a complete set of mappings cannot be specified in the ACM standard.

Therefore mappings to some of the most accepted communications standards are provided in the following subclauses. This provides a set of mappings covering a wide range of basic cases. The definition elsewhere of other mappings to meet special needs is not precluded.

13.2 Application Contexts

The application-context may include the ACMSE by one or other of the following methods:

1 **Explicit**: the ACMSE is included explicitly in the specification of the application-context

2 **Implici**t: where another ASE (the X-ASE) imports all definitions of the ACMSE, the X-ASE is included in the specification of the application-context and the ACMSE need not be mentioned explicitly.

In either case, the specification of the application-context may state selections and omissions of optional primitives and parameters of the ACMSE, but is not obliged to do so. If it does so, the negotiation of these options shall still be performed, and the offers and selections made in these negotiations shall conform to those specified and permitted for the application-context.

13.3 ACSE mapping

This clause maps ACMSE on to an application-association which supports a single abstract-association.

13.3.1 ACMinitiateReq

This shall be sent in the user-information of the A-ASSOCIATE request/indication primitives. The underlying-service-params of the ACM-Initiate service map directly into ACSE parameters.

NOTE
The initialisation PDUs of other ASEs may also be sent in the A-ASSOCIATE user-information.

13.3.2 ACMinitiateContinueReq

The first ACMinitiateContinueReq PDU that is sent during the initialisation phase, from the responder side after receipt of an ACMinitiateReq PDU shall be sent in the user-information of the A-ASSOCIATE response/confirm primitive.

All following ACMinitiateContinueReq PDUs shall be sent in the user-data of the P-DATA request/indication primitive.

NOTE
The initialisation PDUs of other ASEs may also be sent in the A-ASSOCIATE user-information.

13.3.3 ACMinitiateCompleteReq

In a two-way initialisation protocol (where no ACMinitiateContinueReq PDU is sent), ACMinitiateCompleteReq PDU shall be sent in the user-information of the A-ASSOCIATE response/confirm primitive.

In all other initialisation protocols (where one or more ACMinitiateContinueReq PDUs are sent), a positive ACMinitiateCompleteReq PDU shall be sent in the user-data of the P-DATA request/indication primitive. A negative ACMinitiateCompleteReq PDU shall be sent in the user-data of the A-Abort request/indication primitive.

NOTE
The initialisation PDUs of other ASEs may also be sent in the A-ASSOCIATE user-information.

13.3.4 ACMreleaseReq

This shall be sent in the user-information of the A-RELEASE request/indication primitive.

13.3.5 ACMreleaseResp

This shall be sent in the user-information of the A-RELEASE response/confirm primitives. If the Session Negotiated Release Functional Unit is not in use for the application association, the result parameter shall state success.

13.3.6 ACMabortReq

This shall be sent in the user-information of the A-ABORT request/response primitives with the ACSE abort-source parameter set to "acse-service-user".

13.3.7 ACMmodifyReq / ACMmodifyContinueReq / ACMmodifyCompleteReq

These shall be sent in the user-information of the P-DATA request/indication primitive.

13.4 ROSE mapping

This clause defines the relationship to ROSE services only when ROSE is mapped to ACSE and Presentation Services as defined in ISO 9072-2. In such a case ACMSE does not map to ROSE but coexists with it, and is mapped to ACSE exactly as specified in 13.3 of this Standard.

13.5 Mappings to OSI RPC

13.5.1 Introduction

There are many possible ways to map ACM on to OSI RPC. The choice among them depends on wider considerations about the desired structure of the application entity, which are beyond the scope of this Standard. This clause defines two mappings which are particularly appropriate:

* 13.5.2 defines a mapping in which ACMSE is placed outside the RPC ASO and uses its services. This enables use, for example, of the Basic RPC ASO defined in ISO-11578-1.

* 13.5.3 defines a mapping in which ACMSE is included within an extended RPC ASO.

13.5.2 Mapping on to OSI Basic RPC ASO Service

When ACMSE is outside the RPC ASO, it must be mapped on to the services provided by the latter. This clause defines a mapping on to the Basic RPC ASO Service described in ISO 11578-3, in which:

- all ACMSE PDUs from the client (the initiator of the abstract-association) are mapped on to the Argument of BR-INVOKE;
- ACMabortReq from the server (the responder of the abstract-association) is mapped on to BR-ERROR in response to the next BR-INVOKE received;
- all other ACMSE PDUs from the server are mapped on to the Result parameter of BR-RESULT.

This mapping is restricted by permitting only the client to initiate, modify and release the abstract-association, and by requiring the server to wait until it has something to respond to in order to convey ACM-Abort. These restrictions keep the mapping very simple and should be acceptable for many purposes. A mapping which avoided these restrictions would necessarily be more complex.

13.5.2.1 ACMinitiateReq / ACMinitiateContinueReq / ACMinitiateCompleteReq

ACMinitiateReq is sent as the Argument of a BR-INVOKE request.

ACMinitiateContinueReq and ACMinitiateCompleteReq, when sent by the initiator of the abstract-association, are sent as the Argument of a BR-INVOKE. When sent by the responder of the abstract-association, they are sent as the Result parameter of the corresponding BR-RESULT.

If the establishment of the abstract-association requires an odd number of messages, it is the initiator of the abstract-association that sends the ACMinitiateCompleteReq PDU, mapped on to BR-INVOKE. The responder of this message shall reply with a BR-RESULT containing a NULL Result parameter.

13.5.2.2 ACMreleaseReq / ACMreleaseResp

Only the initiator of the abstract-association shall send ACMreleaseReq. It shall be sent as the Argument of a BR-INVOKE. The ACMreleaseResp shall be sent as the Result parameter of the corresponding BR-RESULT.

13.5.2.3 ACMmodifyReq / ACMmodifyContinueReq / ACMmodifyCompleteReq

Only the initiator of the abstract-association shall send ACMmodifyReq. It shall be sent as the Argument of a BR-INVOKE request.

ACMmodifyContinueReq and ACMmodifyCompleteReq, when sent by the initiator of the abstract-association, are sent as the Argument of a BR-INVOKE. When sent by the responder of the abstract-association, they are sent as the Result parameter of the corresponding BR-RESULT.

If the modification of the abstract-association requires an odd number of messages, it is the initiator of the abstract-association that sends the ACMmodifyCompleteReq PDU, mapped on to BR-INVOKE. The responder of the abstract-association shall reply with a BR-RESULT containing a NULL Result parameter.

13.5.2.4 ACMabortReq

An ACMabortReq from the initiator of the abstract-association shall be sent as the Argument of a BR-INVOKE.

When the responder of the abstract-association has to send a ACMabortReq PDU, it shall await the next BR-INVOKE indication and shall reply to it with a BR-ERROR with an Error-Value of "application" and the ACMabortReq PDU as the Argument.

13.5.3 Mapping on to OSI Basic RPC ASO-association Service

When ACMSE is included as part of the RPC ASO, the ACM Service becomes part of the Service of that RPC ASO. The ACM PDUs must then be mapped on to the services provided by elements within or below the RPC ASO. This clause defines a mapping on to the Basic RPC ASO-association Service described in ISO 11578-3, in which:

- ACMinitiateContinueReq and ACMmodifyContinueReq are sent as User Data in BRAA-DATA;
- all other ACM PDUs from the client are sent as User Data in BRAA-ONESHOT req/ind;
- all other ACM PDUs from the server are sent as User Data in BRAA-ONESHOT rsp/cnf.

This mapping is restricted by permitting only the client to initiate, modify and release the abstract-association, and by requiring the server to wait until it has something to respond to in order to convey ACM-Abort. These restrictions keep the mapping very simple and should be acceptable for many purposes. A mapping which avoided these restrictions would necessarily be more complex.

13.5.3.1 ACMinitiateReq / ACMinitiateContinueReq / ACMinitiateCompleteReq

ACMinitiateReq shall be sent in the User Data of a BRAA-ONESHOT request.

ACMinitiateContinueReq shall be sent as User Data in BRAA-DATA request.

ACMinitiateCompleteReq, when sent by the initiator of the abstract-association, shall be sent as User Data in a BRAA-DATA request. The responder shall then reply with a NULL User Data in the outstanding BRAA-ONESHOT response.

ACMinitiateCompleteReq, when sent by the responder of the abstract-association, shall be sent as User Data in the outstanding BRAA-ONESHOT response.

13.5.3.2 ACMreleaseReq / ACMreleaseResp

Only the initiator of the abstract-association shall send ACMreleaseReq. It shall be sent as User Data in a BRAA-ONESHOT request. The ACMreleaseResp shall be sent in the User Data of the corresponding BRAA-ONESHOT response.

13.5.3.3 ACMmodifyReq / ACMmodifyContinueReq / ACMmodifyCompleteReq

Only the initiator of the abstract-association shall send ACMmodifyReq. It shall be sent as User Data in a BRAA-ONESHOT request.

ACMmodifyContinueReq shall be sent as User Data in BRAA-DATA request.

ACMmodifyCompleteReq, when sent by the initiator of the abstract-association, shall be sent as User Data in a BRAA-DATA request. The responder shall then reply with a NULL User Data in the outstanding BRAA-ONESHOT response.

ACMmodifyCompleteReq, when sent by the responder of the abstract-association, shall be sent as User Data in the outstanding BRAA-ONESHOT response.

13.5.3.4 ACMabortReq

An ACMabortReq from the initiator of the abstract-association shall be sent as User Data in a BRAA-ONESHOT request.

When the responder of the abstract-association has to send a ACMabortReq PDU, it shall await the next BRAA-ONESHOT indication and shall reply to it with a BRAA-ONESHOT response containing the ACMabortReq PDU as User Data.

14 Protocol State Tables

The state table defines all the legal behaviours of the ACM protocol machine of each partner of a single abstract-association. This Standard does not define any interaction between two or more abstract-associations, although it permits the ACM service user to do so if required. The table applies to both the initiator and the responder of the abstract-association.

The columns of the table represent the states of the abstract-association. The rows represent the events that can happen on the abstract-association. The boxes define the actions that shall be taken by the ACM protocol machine when that event occurs on the abstract-association in the given state, and the state that is entered after that action is taken. The states, events and actions are represented by codes that are defined in the following clauses. Empty boxes in the table signify that the event is not legal in that state (service or protocol error).

An ACM protocol machine has one variable called **init**, which is **true** if this protocol machine is the initiator of the abstract-association, and **false** if it is the responder. The following notations are used in the table:

* **+init** sets init=true
* **-init** sets init=false
* **init? action** means do action if init=true
* **^init? action** means do action if init=false.

14.1 States

The states of an abstract-association are:

S0	no assoc	the association does not exist and has not been proposed to this ACM protocol machine.
S1	Init sent	the ACM protocol machine has sent ACMinitiateReq PDU or ACMinitiateContinueReq PDU and awaits a response from the other end.
S2	Init rec'd	the ACM protocol machine has issued ACM-Initiate.ind or ACM-Initiate-Continue.ind primitive and awaits the next event from the service user.
S3	normal	the association is established and nothing exciting is happening.
S4	Mod sent	the ACM protocol machine has sent ACMmodifyReq PDU or ACMmodifyContinueReq PDU and awaits a response from the other end.
S5	Mod rec'd	the ACM protocol machine has issued ACMmodify.ind or ACMmodifyContinue.ind primitive and awaits the next event from the service user.
S6	Rel sent	the ACM protocol machine has sent ACMreleaseReq PDU and awaits a response from the other end.
S7	Rel rec'd	the ACM protocol machine has issued ACM-Release.ind primitive and awaits the next event from the service user.

14.2 Events

The events that can occur during the life of an abstract-association are:

Ireq:	ACM-Initiate Request primitive is invoked.

Ipdu:	ACMinitiateReq PDU is received.
ICreq:	ACM-Initiate-Continue Request primitive is invoked.
ICpdu:	ACMinitiateContinueReq PDU is received.
IP+req:	ACM-Initiate-Complete Request primitive is invoked with result=accept.
IP+pdu:	ACMinitiateCompleteReq PDU is received with result=accept.
IP-req:	ACM-Initiate-Complete Request primitive is invoked with result=reject.
IP-pdu:	ACMinitiateCompleteReq PDU is received with result=reject.
Mreq:	ACM-Modify Request primitive is invoked.
Mpdu:	ACMmodifyReq PDU is received.
MCreq:	ACM-Modify-Continue Request primitive is invoked.
MCpdu:	ACMmodifyContinueReq PDU is received.
MPreq:	ACM-Modify-Complete Request primitive is invoked.
MPpdu:	ACMmodifyCompleteReq PDU is received.
Rreq:	A-Release Request primitive is invoked.
RQpdu:	ACMreleaseReq PDU is received.
R+resp:	ACM-Release Response primitive is invoked with result=accept.
R+pdu:	ACMreleaseResp PDU is received with result=accept.
R-resp:	ACM-Release Response primitive is invoked with result=reject.
R-pdu:	ACMreleaseResp PDU is received with result=reject.
Areq:	ACM-Abort Request primitive is invoked.
Apdu:	ACMabortReq PDU is received.

14.3 Actions

The actions that can be performed by the ACM protocol machine are:

Ipdu:	send ACMinitiateReq PDU.
Iind:	issue ACM-Initiate Indication primitive.
ICpdu:	send ACMinitiateContinueReq PDU.
ICind:	issue ACM-Initiate-Continue Indication primitive.
IPpdu:	send ACMinitiateCompleteReq PDU.
IPind:	issue ACM-Initiate-Complete Indication primitive.
Mpdu:	send ACMmodifyReq PDU.
Mind:	issue ACM-Modify Indication primitive.
MCpdu:	send ACMmodifyContinueReq PDU.
MCind:	issue ACM-Modify-Continue Indication primitive.
MPpdu:	send ACMmodifyCompleteReq PDU.
MPind:	issue ACM-Modify-Complete Indication primitive.
RQpdu:	send ACMreleaseReq PDU.
Rind:	issue ACM-Release Indication primitive.
RPpdu:	send ACMreleaseResp PDU.

Rcnf:	issue ACM-Release Confirm primitive.
Apdu:	send ACMabortReq PDU.
Aind:	issue ACM-Abort Indication primitive.

\State Evt\	S0 no assoc	S1 Init sent	S2 Init rec'd	S3 normal	S4 Mod sent	S5 Mod rec'd	S6 Rel sent	S7 Rel rec'd
Ireq	Ipdu +init S1							
Ipdu	Iind -init S2							
ICreq			ICpdu S1					
ICpdu		ICind S2						
IP+req			IPpdu S3					
IP+pdu		IPind S3						
IP-req			IPpdu S0					
IP-pdu		IPind S0						
Mreq				Mpdu S4				
Mpdu				Mind S5	init? S4 ^init? Mind S5		S6	
MCreq						MCpdu S4		
MCpdu					MCind S5			
MPreq						MPpdu S3		
MPpdu					MPind S3			

(continued)

\State Evt\	S0 no assoc	S1 Init sent	S2 Init rec'd	S3 normal	S4 Mod sent	S5 Mod rec'd	S6 Rel sent	S7 Rel rec'd
Rreq				RQpdu S6				
RQpdu				Rind S7	Rind S7		init? S6 ^init? Rind S7	
R+resp								RPpdu S0
R+pdu							Rcnf S0	
R-resp								RPpdu S3
R-pdu							Rcnf S3	
Areq		Apdu S0	Apdu S0	Apdu S0	Apdu S0	Apdu S0	Apdu S0	Apdu S0
Apdu	S0	Aind S0	Aind S0	Aind S0	Aind S0	Aind S0	Aind S0	Aind S0

NOTE
The case in the bottom left corner of the table, where an ACMabortReq PDU is received for an association that does not exist, does not necessarily signify a protocol error, since it can occur as a result of a collision of ACMabortReq PDUs, or of a collision of an ACMabortReq PDU with an ACMinitiateCompleteReq PDU with result=reject.

Annex A
(Informative)
ACM negotiation protocol example

A.1 Requirements

ACM protocol can be used to agree on PAC properties required for an association. PACs with the desired properties must be received from a Privilege Attribute Server (this is out of the scope of this Standard). Those properties could be:

1 Delegatable PAC: Specifies PAC properties in respect to delegation:

- PAC not delegatable
- PAC delegatable ; Traceability required
- PAC delegatable ; No traceability required.

2 Privilege types required for the access control decision: Specifies on which type of privilege attribute the application bases its access control.

3 Privilege value required to have the association authorised: Specifies a type of privilege and its value that the PAC must contain.

4 PAC cryptographic protection needed: Specifies whether a signed PAC or a sealed PAC is needed for that association.

A.2 Data structure specification

Context parameter:

```
cpType : { ... }    -- "PAC"
cpValue ::= SEQUENCE {
        ecv     ECMAapa.ECV OPTIONAL,
        pac     ECMAapa.GeneralisedCertificate OPTIONAL }
```

Negotiation parameters:

```
npId : 1
npValue ::= ENUMERATED {
        notDelegatable (0),
        delegatableWithoutTraceability (1),
        delegatableWithTraceability (2) }
npId : 2
npValue ::=
        SEQUENCE OF ECMAapa.Identifier
npId : 3
npValue ::=
        SEQUENCE OF ECMAapa.PrivilegeAttribute
npId : 4
npValue ::= ENUMERATED {
        seal (0),
        signature (1) }
```

A.3 Example: Association requiring a signed PAC

A client possesses both a signed PAC and a sealed PAC. It wants to access an application which only supports signed PAC. Here are the sequences:

```
                                    Initiator              Responder

The initiator sends a sealed PAC along with corresponding ECV :

ACM-INITIATE (
                                    cpRef = 1
                               cpType = "PAC"
              cpValue = Sealed PAC and ECV)    >>>>>>

The Responder determines that the PAC is sealed and indicates that a signed PAC is needed :

                                                       ACM-INITIATE-COMPLETE (
                                                           result=reject
                                                           reason="see returned parameters"
                                                           cpRef = 1
                                                           npId = 4
                                               <<<<<<      npValue = signature)

The initiator sends a signed PAC along with corresponding ECV:

ACM-INITIATE (
                                    cpRef = 9
                               cpType = "PAC"
              cpValue = Signed PAC and ECV)    >>>>>>
```

The responder accepts the signed PAC, checks it and initialises the association-context from the PAC that has been sent.

```
                                                       ACM-INITIATE-COMPLETE (
                                               <<<<<<      result=accept)
```

Annex B
(Informative)
Relationship to Kerberos version 5

B.1 Introduction

This annex shows how the ACM Service and Protocol defined in this Standard can be used to manage an association that uses Kerberos Version 5 security mechanisms as well as other (security and non-security) mechanisms.

ACM is relevant to the Kerberos Client/Server (CS) protocol which is used between the applications themselves. This annexe does not address the Authentication Server (AS) and Ticket Granting Server (TGS) protocols which are also defined in the Kerberos Version 5 Internet Draft.

The benefits of using ACM to convey Kerberos elements are:

* Mappings of ACMSE onto different underlying protocols can be exploited by Kerberos without having to be re-specified.
* Kerberos can be combined with other (security and non-security) mechanisms when establishing associations, without the need to define an enveloping message format. This is accomplished by use of the context parameter and/or association-attribute concepts within the generalised negotiation function of ACM.

B.2 Kerberos Version 5 Client-Server Protocol

The Kerberos version 5 specification [Kerberos Version 5 - Revision #5.1] includes a Client Server Protocol involving the following exchanges:

* KRB_AP_REQ is sent from the Client to the Server at the beginning of their interaction. It includes the Kerberos Ticket, the Kerberos Authenticator, and control fields.
* KRB_AP_REP is sent by the Server to the Client if requested in response to the KRB_AP_REQ.
* KRB_SAFE is sent in either direction when the application needs to send a tamper-proof message to its peer.
* KRB_PRIV is sent in either direction when the application needs to send a confidential message to its peer.
* KRB_ERROR is sent in order to communicate data about an error that has occurred.

KRB_SAFE and KRB_PRIV are related to the use of an association for application messages, not to its management. Hence they are outside the scope of the ACM service and protocol. KRB_AP_REQ and KRB_AP_REP are part of establishing an association, providing peer authentication, helping to authorise the association, and initialising security mechanisms that will be used within the association. All these functions are within the scope of ACM and as such are considered in this annexe. KRB_ERROR is used both in establishment and in use of the association and also in the AS and TGS protocols. It is relevant to ACM when it is used in rejection of an association or in aborting an association after a fatal error.

B.3 Mapping of KRB_AP_REQ and KRB_AP_REP

Both KRB_AP_REQ and KRB_AP_REP messages contain a number of fields, each of which define characteristics of the proposed association. Hence these can be viewed as specifying association-attributes as defined in this Standard. There are two ways to do this: the tightly bound and the loosely bound methods.

In the tightly bound method, the whole KRB_AP_REQ message is sent as a single context parameter in ACM-Initiate, and the whole KRB_AP_REP message is sent as a single context parameter in the following ACM-Initiate-Continue or ACM-Initiate-Complete. The syntax of these context parameters is exactly that specified for the messages themselves in the Kerberos specification.

In the loosely bound method, the individual elements within the KRB_AP_REQ and KRB_AP_REP messages are each encoded as separate context-parameters and negotiation-parameters. The value syntax will often be the same as that of the individual fields within the messages. For example, the kerberosTicket and kerberosAuth fields may be used in this way. The specification of how to do it may take a form similar to that illustrated in annex A.

Both methods gain the benefits listed in B.1 of use of predefined mappings and combination with other mechanisms, so the choice between them should be made in the light of other design objectives. The loosely bound method gives the greatest readiness for enhancement and inter-version compatibility, and also gives the greatest freedom to select and combine both Kerberos and other mechanisms. The tightly bound method might be preferred when it is required to minimise change to implemented systems whose components expect to receive and parse KRB_AP_REQ and KRB_AP_REP messages exactly as specified in the Kerberos specification.

B.4 Mapping of KRB_ERROR

ACM is only involved when KRB_ERROR involves a fatal error that requires termination of the association. It is sensible to transmit this message at the time of aborting an abstract-association.

Communications connection abort facilities often allow for only a very small amount of user data to be transmitted, so it is safest to send the KRB_ERROR message as user data prior to the association abort rather than with it, at least where the flexibility of communications mapping is to be retained. It should also be noted that some communications facilities allow "abort" messages to overtake and erase undelivered data messages. Hence the safest mapping is one in which KRB_ERROR is followed by ACM-Release rather than ACM-Abort, or one where the receiver of the KRB_ERROR invokes ACM-Abort.

Annex C
(Informative)
Relationship to GSS-API

C.1 Introduction

The Generic Security Service API (GSS-API), defined in an Internet Draft, is an application programming interface which allows the interface-user to invoke security related functions in a way that is independent of the security mechanisms chosen to implement those functions. These mechanisms are known to, and implemented by, the interface-provider. The interface-user may be applications software, but equally, and perhaps more usually, it can be middle ware.

The GSS-API provides interfaces to obtain credentials, to initiate and terminate **security contexts**, and to perform secure data transfer and other operations within a security context. The concept of a security context is a subset of the ACM association concept, being concerned specifically with security. ACM is relevant to those parts of GSS-API that establish and terminate security contexts; it is not relevant to other aspects of GSS-API.

The use of the ACM Service is a function of the GSS-API interface-user, and the interface-provider is not be required to be aware of or influenced by the ACM Service. This is consistent with the GSS-API principle of protocol independence. Therefore this annex describes a way for the GSS-API interface-user to co-ordinate the GSS-API interface functions with the ACM service primitives.

C.2 Security Context Establishment

The following sequence shows how ACM can be used for a two-message exchange generated by means of GSS-API calls:

Initiator		**Responder**
Call GSS_Init_sec_context		
(returns GSS_CONTINUE_NEEDED)		
(provides an output-token)		
Make ACM-Initiate.req	>>>>>>	Receive ACM-Initiate.ind
		Call GSS_Accept_sec_context
		(returns GSS_COMPLETE)
		(provides output token)
Receive ACM-Initiate-Complete.ind	<<<<<<	Make ACM-Initiate-Complete.req
Call GSS_Init_sec_context		
(with input-token)		
(returns GSS_COMPLETE)		
(provides NO output-token)		

Longer sequences than this are possible. For example, if the GSS-API provider at the responding end had returned GSS_CONTINUE_NEEDED instead of GSS_COMPLETE, the responder must invoke ACM-Initiate-Continue in order to send the second token, and so on until one or other side is ready to complete the exchange.

It is also possible for the exchange to continue even after the GSS-API provider has ceased generating and expecting tokens, if the ACM service user has negotiations of its own to perform. This is necessary in the case where GSS-API generates only a single token, since ACM requires a response in order to confirm the abstract-association. The sequence for this case is as follows:

Initiator		Responder
Call GSS_Init_sec_context		
(returns GSS_COMPLETE)		
(provides an output-token)		
Make ACM-Initiate.req	>>>>>>	Receive ACM-Initiate.ind
		Call GSS_Accept_sec_context
		(returns GSS_COMPLETE)
		(provides NO output token)
Receive ACM-Initiate-Complete.ind	<<<<<<	Make ACM-Initiate-Complete.req
(does NOT call GSS_Init_sec_context)		

C.3 Security Context Deletion

The sequence of events for orderly termination is as follows:

Initiator		Responder
Call GSS_Delete_sec_context		
(returns GSS_COMPLETE)		
(may provide an output-context-token)		
Make ACM-Release.req	>>>>>>	Receive ACM-Release.ind
		Call GSS_Process_context_token
		(returns GSS_COMPLETE)
Receive ACM-Release.cnf	<<<<<<	Make ACM-Release.rsp

GSS-API provides no direct means to do this. It could be done by a sequence similar to the release sequence above, discarding the output-token, but this solution is not ideal, since the abort may be urgent and GSS_Delete_sec_context can block while accessing network security servers. A better method, provided that such servers can cope with both parties deleting the context independently, is the following sequence:

Initiator		Responder
Make ACM-Abort.req	>>>>>>	Receive ACM-Abort.ind
Call GSS_Delete_sec_context		Call GSS_Delete_sec_context
Discard output-token		Discard output-token

C.4 Security Context Modification

GSS-API does not provide the means to do this.

C.5 GSS-API Output Token

The GSS-API output token can be carried in the ACM protocol either as a single context parameter or preferably as a complete ACM PDU.

Annex D
(Informative)
Relationship to OSI-TP

Clause 13 of this Standard maps ACM protocol on to ACSE, ROSE and RPC, which are all "point-to-point" communication types. It is the intent of ACM also to be mappable to multi-dialogue type communications like OSI-TP dialogue types.

Standard ISO 10026 - OSI-TP - defines a standard for distributed transactions. A transaction is a set of related operations characterised by the ACID concept:

- Atomicity: none or all operations performed
- Consistency: accurate, correct and valid with respect to application semantics
- Isolation: no intermediate or partial result
- Durability: the set of related operations is completed and a result is committed.

The communication aspect of OSI-TP is three-fold:

- TP Service Users (TPSU) and Dialogues between them [Applications]
- TP Service Providers (TPSP) and application-associations between them [TP monitors]
- OSI Communications based on ACSE, CCR and Presentation [OSI Communications].

Certain rules are valid as regards TP-dialogues on ACSE:

- a TP dialogue can last within or beyond an application-association;
- only one TP-dialogue can use an application-association at a time

Two main "levels" of mapping of ACM can be modelled:

- "Upper" level TPSU context related to TP-dialog (ACSE-association independent)
- "Lower" level TPSP context related to application-associations between TP monitors.

A TP mapping must specify whether the "upper" and "lower" levels are clearly isolated or if they are tightly coupled.

- If well isolated, one could use the ACSE-mapping of ACM for the "lower" level part and define a ACM mapping to the "upper" part.
- If coupled, a more complex ACM mapping is foreseen where two levels of context have to be managed simultaneously.

The resolution of this issue depends on the way in which real TP systems operate. This is beyond the scope of this Standard. It should be considered in the context of a Work Item on OSI-TP Security.

Annex E
(Informative)
Managed objects

E.1 Introduction

This annex specifies one possibility for specifying managed objects (MOs) for ACM. It includes definitions of the following managed object classes according to the Guidelines for Definition of Managed Objects (ISO 10165-4):

- association manager
- abstract-association.

Both managed object classes have a basic set of attributes that are used to contain their identity and their type.

The association manager MO class additionally supports an administrative state (enabled/disabled) and an operative state (busy, idle, ...). On both states a GET operation can be performed ; the administrative state also allows a SET operation.

The abstract-association MO class additionally to the basic attributes mentioned above supports attributes for the abstract-association context.

On all of these attributes only GET operations can be performed. Modification of context attributes of abstract-associations is part of ACM functionality and can be performed using the ACM-Modify service primitive. (See also clause 10.7).

E.2 acmService Class

```
acmService MANAGED OBJECT CLASS
  DERIVED FROM securityService;
  CHARACTERIZED BY acmServicePackage PACKAGE
    BEHAVIOUR acmServiceBehaviour BEHAVIOUR
      DEFINED AS
           -- An instance of this class refers to a
           -- specific security service provided to
           -- network users in order to establish,
           -- control and release abstract
           -- associations.
      ;
    ;
    ATTRIBUTES
      acmServiceId          GET,
           -- used for naming, i.e. this
           -- attribute must be specified in the
           -- name binding attribute inherited from top.
      acmServiceType        GET,
           -- this is an OCTET STRING used to
           -- differ between ACM versions
      typeText              GET-REPLACE,
           -- for additional description,
        operationalState          GET,
                  -- current ACM state (busy / idle)
        administrativeState       GET, SET
                  -- enabled or disabled
      ;
      NOTIFICATIONS
        objectCreation,
        objectDeletion,
        attributeValueChange
      ;
    ;
    REGISTERED AS { ... } ;
```

E.3 abstractAssociation Class

```
abstractAssociation MANAGED OBJECT CLASS
    DERIVED FROM top;
    CHARACTERIZED BY abstractAssociationPackage PACKAGE
      BEHAVIOUR abstractAssociationBehaviour BEHAVIOUR
        DEFINED AS
                  -- An instance of this class refers to an
                  -- abstract-association in the sense of
                  -- the ACM standard.
        ;
      ;
      ATTRIBUTES
        abstractAssociationId          GET,
                  -- used for naming, i.e. this
                  -- attribute must be specified in the
                  -- name binding attribute inherited
                  -- from top. Abstract-associations
                  -- must be contained in the
                  -- acmService managed object.
        abstractAssociationType        GET,
                  -- this is an OCTET STRING
        typeText                       GET-REPLACE,
                  -- for additional description,
        contextAttributes              GET
      ;
      NOTIFICATIONS
```

```
        objectCreation,
        objectDeletion,
        attributeValueChange
    ;
;
CONDITIONAL PACKAGES
    ecmaSecurityAttributePackage        PACKAGE
        PRESENT IF "ECMA attributes supported",
    culturalConventionsPackage          PACKAGE
        PRESENT IF "cultural conventions supported";
REGISTERED AS { ... } ;
```