The drawings contained in this recommendation have been done in Autocad

Recommendation X.407

MESSAGE HANDLING SYSTEMS: ABSTRACT SERVICE DEFINITION CONVENTIONS[1)]

(Melbourne, 1988)

The establishment in various countries of telematic services and computerÑbased storeÑandÑforward message services in association with public data networks creates a need to produce standards to facilitate international message exchange between subscribers to such services.

The CCITT,

considering

(a) the need for Message Handling Systems;

(b) that Message Handling is a complex distributed information processing task;

(c) that a means for abstractly defining such tasks is required;

(d) that Recommendation X.200 defines the Reference Model of Open Systems Interconnection for CCITT applications;

(e) that Recommendations X.208, X.217, X.218, and X.219 provide the foundation for CCITT applications,

unanimously declares

(1) that conventions for defining abstract services are defined in Section 2;

(2) that techniques for the realization of abstract services so defined are discussed in Section 3.

TABLE OF CONTENTS

SECTION 1 Ñ *Introduction*

0 *Introduction*

1 *Scope*

2 *References*

3 *Definitions*

4 *Abbreviations*

5 *Conventions*

5.1 ASN.1

5.2 Terms

SECTION 2 Ñ *Abstract service definition conventions*

6 *Overview*

7 *Abstract models*

7.1 Abstract objects

7.2 Abstract ports

7.3 Abstract services

7.4 Abstract refinements

---

1) Recommendation X.407 and ISO 10021Ñ3, Information processing systems Ñ Text Communication Ñ MOTIS Ñ Abstract Service Definition Conventions, were developed in close collaboration and are technically aligned.

8 *Abstract services*

8.1 Abstract procedures

8.2 Abstract bind operations

8.3 Abstract unbind operations

8.4 Abstract operations

8.5 Abstract errors

SECTION 3 Ñ *Abstract service realizations*

9 *Overview*

10 *OSI realizations*

10.1 ROS realizations

10.2 NonÑROS realizations

11 *Proprietary realizations*

11.1 Distributed realizations

11.2 NonÑdistributed realizations

Annex A Ñ Example of use of abstract service notation

Annex B Ñ Reference definition of object identifiers

Annex C Ñ Reference definition of notation

Annex D Ñ Differences between CCITT Recommendation and ISO Standard

Annex E Ñ Index

SECTION 1 Ñ INTRODUCTION

0 Introduction

This Recommendation is one of a set of Recommendations for Message Handling. The entire set provides a comprehensive blueprint for a Message Handling System (MHS) realized by any number of cooperating open systems.

The Message Handling System made possible by these Recommendations is a complex distributed information processing task, many of whose components themselves have these characteristics.

This Recommendation specifies the conventions for defining the distributed information processing tasks of Message Handling and may also be useful for other applications.

The text of this Recommendation is the subject of joint CCITTÑISO agreement. The corresponding ISO specification is ISO 10021Ñ3.

1 Scope

This Recommendation specifies the conventions used to specify the distributed information processing tasks that arise in Message Handling.

This Recommendation is structured as follows. Section 1 is this introduction. Section 2 specifies the conventions for defining a distributed information processing task abstractly. Section 3 gives principles for realizing the communication aspects of such tasks concretely, such as by Open Systems Interconnection (OSI) protocols. Annexes provide important supplemental information.

There are no requirements for conformance to this Recommendation.

2 References

This Recommendation cites the documents below.

Recommendation X.200 Reference model of open systems interconnection for CCITT applications (see also ISOÊ7498).

Recommendation X.208 Specification of abstract syntax notation one (ASN.1) (see also ISO 8824).

Recommendation X.209 Specification of basic encoding rules for abstract syntax notation one (ASN.1) (see also ISOÊ8825).

Recommendation X.217 Association control service definition for open systems interconnection for CCITT applications (see also ISO 8649).

Recommendation X.219 Remote operations: Model, notation and service definition (see also ISO 9072Ñ1).

3 Definitions

For the purposes of this Recommendation, the definition of Annex E and below apply.

This Recommendation is based upon the concepts developed in Recommendation X.200 and uses the following terms defined in it:

a) abstract syntax;

b) Application Layer;

c) application protocol data unit (APDU);

d) application protocol;

e) application service element (ASE);

f) concrete transfer syntax;

g) distributed information processing task;

h) layer service;

i) layer;

j) open system;

k) Open Systems Interconnection (OSI);

l) real open system.

This Recommendation uses the following terms defined in Recommendation X.208:

a) Abstract Syntax Notation One (ASN.1);

b) (data) type;

c) (data) value;

d) import;

e) Integer;

f) macro;

g) module;

h) Object Identifier;

i) tag.

This Recommendation uses the following terms defined in Recommendation X.209:

a) Basic Encoding Rules.

This Recommendation uses the following terms defined in Recommendation X.217:

a) application context (AC).

This Recommendation uses the following terms defined in Recommendation X.219:

a) bind operation;

b) error;

c) linked;

d) operation;

e) Remote Operation Service (ROS);

f) Remote Operations;

g) unbind operation.

4 Abbreviations

For the purposes of this Recommendation, the abbreviations of Annex E apply.

5 Conventions

This Recommendation uses the descriptive conventions identified below.

5.1 *ASN.1*

This Recommendation uses for the indicated purposes the following ASN.1Ñbased descriptive conventions:

a) to define the OBJECT, PORT, and REFINE macros, the ASN.1 macro notation of RecommendationÊX.208;

b) to define the ABSTRACTÑBIND, ÑUNBIND, ÑOPERATION, and ÑERROR macros, the BIND, UNBIND, OPERATION, and ERROR macros of Recommendation X.219;

c) to specify the abstract syntax of information objects in the example of Annex A, ASN.1 itself;

d) to specify various *abstract models* in the example of Annex A, the OBJECT, PORT, and REFINE macros of ¤ 7;

e) to specify various *abstract services* in the example of Annex A, the ABSTRACTÑBIND, ÑOPERATION, and ÑERROR macros of ¤ 8.

ASN.1 appears both in the body of this Recommendation to aid the exposition and again, largely redundantly, in Annexes for reference. If differences are found between the two, a specification error is indicated.

Note Ñ ASN.1 tags are implicit throughout the ASN.1 modules in the Annexes; the modules are definitive in that respect.

5.2 *Terms*

Throughout this Recommendation, terms are rendered in **bold** when defined, in *italic* when reference is made to them prior to their definitions, without emphasis upon all other occasions.

Terms that are proper nouns are capitalized, generic terms are not.

SECTION 2 Ñ ABSTRACT SERVICE DEFINITION CONVENTIONS

6 Overview

When faced with the job of describing and specifying a complex distributed information processing task, one is wise to begin by specifying the task in abstract, rather than concrete terms. This approach ensures that the task's functional requirements are stated independently of its concrete realization. Such separation is important, among other reasons, because each aspect of the task may admit of *several* concrete realizations. In a Message Transfer System comprising three message transfer agents, e.g., the first and second might interact using OSI communication, the second and third by proprietary means.

This section specifies the conventions for abstractly describing a distributed information processing task both macroscopically and microscopically. The former description is called an abstract model, the latter an *abstract service*.

Various formal tools for specifying *abstract models* and *services* are defined in this section. A comprehensive example of their use is given in Annex A. The reader may wish to refer to that Annex while reading the present section.

This section covers the following topics:

a) *Abstract models*.

b) *Abstract services*.

Note Ñ The formal tools mentioned above are neither a formal description language nor a substitute for such. They are simply an ASN.1 notation that supports the informal descriptive conventions defined in this section.

7 Abstract models

A macroscopic description of a distributed information processing task is called an **abstract model (model)** of that task and of the environment in which it is carried out. It is based upon the concepts of *abstract objects*, *ports*, *services*, and *refinements*. (The concept of an *abstract service* is much more fully developed in ¤ 8.)

7.1 *Abstract objects*

An **abstract object (object)** is a functional entity, one of perhaps several which interact with one another. Objects are of different types which determine their function and behavior. An object of one type, e.g., might represent a system, multiple objects of another type its users. Objects interact by means of *abstract ports*.

An object type is specified by means of the OBJECT macro. Such a specification lists the types of *abstract ports* that provide access to such an object. For each *asymmetric* port type, the specification indicates whether the ports of that type are *consumer* or *supplier* ports.

```
OBJECT MACRO ::=
BEGIN

TYPE NOTATION     ::=   ÒPORTSÓ Ó{Ó PortList Ò}Ó | empty
VALUE NOTATION    ::=   value (VALUE OBJECT IDENTIFIER)

PortList          ::=   Port Ò,Ó PortList | Port
Port              ::=   value (PORT) PortType

PortType          ::=   Symmetric | Asymmetric

Symmetric         ::=   empty
Asymmetric        ::=   Consumer | Supplier

Consumer          ::=   Ò[C]Ó

Supplier          ::=   Ò[S]Ó

END
```

A data value of type OBJECT is an Object Identifier that unambiguously and uniquely identifies the specified object type.

Note Ñ The keyword ÒOBJECTÓ is reserved in ASN.1. Selection of a suitable replacement for use in the present context is for further study.

7.2 *Abstract ports*

An **abstract port (port)** is a point at which an abstract object interacts with another abstract object. Ports are of different types which determine the kinds of interactions they enable. Ports of one type, e.g., might represent the means by which a directory system is accessed, ports of another type the means by which it is administered.

Port types are themselves of the following two varieties:

a) **symmetric**: All instances of a symmetric port type are identical;

b) **asymmetric**: Each instance of an asymmetric port type is of one of two kinds, **supplier** and **consumer**.

Note Ñ A particular allocation of the terms ÒsupplierÓ and ÒconsumerÓ is often intuitive. One might naturally consider a file system, e.g., to present supplier ports to its users and administrators. Strictly speaking, however, the assignment of the two terms is arbitrary.

Two objects can interact with one another by means of a port in one and a port in the other only while those ports are in contact with one another, or **bound**. The actions by means of which this state is initiated and terminated for one or more port pairs are called **binding** and **unbinding**, respectively.

Two ports can be bound only if they **match**. Any two ports of the same, symmetric type match. Two ports of the same, asymmetric type match if and only if one is a supplier, the other a consumer.

A port type is specified by means of the PORT macro. Such a specification identifies the *abstract operations* that represent the interactions possible while two such ports are bound. If none is listed, the *abstract operations* shall be considered unspecified.

```
PORT MACRO ::=
BEGIN

TYPE NOTATION     ::=   Operations | empty
VALUE NOTATION    ::=   value (VALUE OBJECT IDENTIFIER)

Operations        ::=   Symmetrical | Asymmetrical

Symmetrical       ::=   ÒABSTRACTÓ ÒOPERATIONSÓ Ò{Ó OperationList Ò}Ó
Asymmetrical      ::=   OneSided | TwoSided

OneSided          ::=   Consumer | Supplier
TwoSided          ::=   Consumer Supplier | Supplier Consumer

Consumer          ::=   ÒCONSUMERÓ ÒINVOKESÓ Ò{Ó OperationList Ò}Ó
Supplier          ::=   ÒSUPPLIERÓ ÒINVOKESÓ Ò{Ó OperationList Ò}Ó

OperationList     ::=   Operation Ò,Ó OperationList | Operation
Operation         ::=   value (ABSTRACTÑOPERATION) | ÑÑ identifying the abstract
                                                      operation by data value
                        type ÑÑ identifying the abstract operation by data type

END
```

If the port type is symmetric, both objects offer all listed *abstract operations*. If the port type is asymmetric, the macro distinguishes between the *abstract operations* an object with the consumer port offers and those an object with the supplier port offers.

A data value of type PORT is an Object Identifier that unambiguously and uniquely identifies the specified port type.

7.3 *Abstract services*

An **abstract service** is the set of capabilities that one object offers to another by means of one or more of its ports. The former object is called an **abstract service provider (provider)**, the latter an **abstract service user (user)**. Each port in question may be either symmetric or asymmetric and, if the latter, either consumer or supplier.

An abstract service may have any number of users and providers.

Whenever the abstract service ports of a provider are bound to the matching ports of a user, an **abstract association** (or **association**) is said to exist between the two objects.

An abstract service is specified as indicated in ¤ 8.

Note Ñ An abstract service serves much the same purpose within the Application Layer as does one of the layer services of lower OSI layers.

7.4 *Abstract refinements*

An object can be viewed in different ways at different times. On some occasions it is convenient to think of an object as atomic. This is the case, e.g., when describing how an object interacts with other objects external to it, i.e., when specifying its abstract service. On other occasions, it may be more convenient to think of an object as composite, i.e., constructed from other objects. This might be the case, e.g., when describing how an object is realized.

Like any objects, component objects have ports. Some are those visible on the ÒsurfaceÓ of the constructed object. Others enable the component objects to interact, thus supporting the provision and use of lesser abstract services among the component objects, which cooperate to provide the overall abstract service of the constructed object.

The functional decomposition of an object into several lesser objects is called the **abstract refinement (refinement)** of that object.

The technique of refinement can be applied recursively. A component object can itself be refined to reveal *its* internal structure. This can continue until one reaches component objects best considered atomic.

A refinement is specified by means of the REFINE macro. It identifies the object whose internal structure is being revealed and the component objects used in its construction. Each component object is characterized as either unique or recurring. The macro also indicates which ports of component objects are bound to ports of other component objects, and which are visible at the surface of the composite object.

```
REFINE MACRO ::=
BEGIN

TYPE NOTATION     ::=   Object ÒASÓ ComponentList
VALUE NOTATION    ::=   value (VALUE OBJECT IDENTIFIER)

ComponentList     ::=   Component ComponentList | Component
Component         ::=   ObjectSpec Ports

ObjectSpec        ::=   Object | Object ÒRECURRINGÓ

Ports             ::=   PortSpecList | empty
PortSpecList      ::=   PortSpec PortSpecList | PortSpec
PortSpec          ::=   value (PORT) PortSide PortStatus

PortSide          ::=   Consumer | Supplier | empty
Consumer          ::=   Ò[C]Ó
Supplier          ::=   Ò[S]Ó

PortStatus        ::=   ÒVISIBLEÓ | ÒPAIREDÓ ÒWITHÓ ObjectList

ObjectList        ::=   Object Ò,Ó ObjectList | Object
Object            ::=   value (OBJECT)

END
```

A data value of type REFINE is an Object Identifier.

Note Ñ As with objects themselves, ports, in principle, can be viewed in different ways at different times. On some occasions, it is convenient to think of a port (pair) as atomic. However, one can imagine refining a port itself to examine how communication of this type can be provided. In this view, a port pair is itself viewed as being supported by a collection of objects. This would enhance the ability to specify communications capabilities. This Òport refinementÓ concept is not pursued further in this version of this Recommendation.

8 Abstract services

A microscopic description of a distributed information processing task is a specification of the abstract service that defines how the task is initiated, controlled, and terminated. It is based upon the concepts of *abstract bind operations*, *unbind operations*, *operations*, and *errors*, as well as the enabling concept of *abstract procedures*.

Note Ñ The macros defined below imply use of ASN.1 to specify arguments, results, and parameters. Any contextÑspecific tags, e.g., assigned in the course of the specifications, although meaningless in that context, play an important role in a ROS realization of the abstract service.

8.1 *Abstract procedures*

An **abstract procedure (procedure)** is a task that one object carries out at another's request. The making of the request and the carrying out of the task are called the **invocation** and **performance** of the procedure. The objects that issue and act upon the request are called the **invoker** and **performer**, respectively.

A procedure may (but need not) require that an invoker, upon invocation, supply to the performer a single information object of a prescribed type, which is called the procedure's **argument**.

Every performance of every procedure has an outcome, success or failure. A procedure is considered to succeed if it is carried out in full, to fail if it is terminated prematurely.

A procedure may (but need not) require that the performer apprise the invoker of success. It may (but need not) further require that it supply, when reporting success, a single information object of a prescribed type, which is called the procedure's **result**.

A procedure may (but need not) require that the performer apprise the invoker of failure. It may (but need not) further require that it supply certain information when reporting failure.

Note Ñ In subsequent ¤ ASN.1 is prescribed as the means for specifying the abstract syntax of the arguments and results of procedures (as well as of the parameters of *abstract errors*). These uses of ASN.1 do not imply that these information objects are *necessarily* transported between open systems. In particular, the fact that the information objects, by virtue of their description in ASN.1 and of its Basic Encoding Rules, have concrete transfer syntaxes is immaterial in the present context. ASN.1 is simply a convenient tool for formally describing the information objects' abstract syntax.

8.2 *Abstract bind operations*

An **abstract bind operation** is a procedure whose successful performance binds one or more pairs of abstract ports. The object which invokes an abstract bind operation is said to be the **initiator**, that which performs it the **responder**.

An abstract bind operation is suitable for binding a particular set of ports of the initiator to a matching set of the responder. Where one or more ports in the set are asymmetric, the abstract bind operation may be suitable for binding to the consumer side only, the supplier side only, or to either.

An abstract bind operation is a fully general procedure except that, if information is conveyed to the invoker upon failure, it is constrained to a single information object, called **error information**.

An abstract bind operation is specified by means of the ABSTRACTÑBIND macro whose definition is as follows:

```
ABSTRACTÑBIND MACRO ::=
BEGIN

TYPE NOTATION     ::=   Ports Bind
VALUE NOTATION    ::=   value (VALUE BindType)

Ports             ::=   ÒTOÓ Ò{Ó PortList Ò}Ó | empty
PortLis           ::=   Port Ò,Ó PortList | Port
Port              ::=   value (PORT) PortSide
PortSide          ::=   Consumer | Supplier | empty
Consumer          ::=   Ò[C]Ó
Supplier          ::=   Ò[S]Ó

Bind              ::=   type (BindType) ÑÑ must be a BIND type
                        | empty <BindType ::= BIND>

END
```

The ÒPortsÓ clause, introduced by the keyword ÒTOÓ, lists the ports of a responder which this abstract bind operation will bind. If an asymmetric port is listed there, without being qualified by Ò[S]Ó or Ò[C]Ó, this means that the abstract bind operation is suitable for use in binding such a port in either direction.

Note that the specification of the argument, result, and/or error information is accomplished by means of an (embedded) BIND macro of Remote Operations, defined in Recommendation X.219, and it is a value of such a type that the macro returns. If none is provided, the default ÒBINDÓ is returned.

Note Ñ The relationship of ABSTRACTÑBIND and BIND can help make trivial the ROS realization of an abstract service; see ¤ 10.1.

An abstract service typically comprises an abstract bind operation for each type of port involved in its provision. When several port types are involved, their abstract bind operations may but need not be distinct.

8.3 *Abstract unbind operations*

An **abstract unbind operation** is a procedure whose performance, successful or not, unbinds two ports. It is invoked by the object which invoked the corresponding abstract bind (i.e., the initiator) and performed by the responder.

An abstract unbind operation is suitable for unbinding a particular set of ports of the initiator from a matching set of the responder. Where one or more ports in the set are asymmetric, the abstract unbind operation may be suitable for unbinding from the consumer side only, the supplier side only, or either.

An abstract unbind operation is a fully general procedure except that, if information is conveyed to the invoker upon failure, it is constrained to a single information object, called **error information**.

An abstract unbind operation is specified by means of the ABSTRACTÑUNBIND macro whose definition is as follows:

```
ABSTRACTÑUNBIND MACRO ::=
BEGIN

TYPE NOTATION     ::=   Ports Unbind
VALUE NOTATION    ::=   value (VALUE UnbindType)

Ports             ::=   ÒFROMÓ Ò{Ó PortList Ò}Ó
PortList          ::=   Port Ò,Ó PortList | Port
Port              ::=   value (PORT) PortSide
PortSide          ::=   Consumer | Supplier | empty
Consumer          ::=   Ò[C]Ó
Supplier          ::=   Ò[S]Ó

Unbind            ::=   type (UnbindType) |
                        ÑÑ must be an UNBIND type
                        empty <UnbindType ::= UNBIND>.

END
```

The ÒPortsÓ clause, introduced by the keyword ÒFROMÓ, lists the ports of a responder from which this abstract unbind operation will unbind. If an asymmetric port is listed there, without being qualified by Ò[S]Ó or Ò[C]Ó, this means that the abstract unbind operation is suitable for use in unbinding such a port in either direction (although the actual direction is determined by the direction in which the bind took place).

Note that the specification of the argument, result, and/or error information is accomplished by means of an (embedded) UNBIND macro of Remote Operations, defined in Recommendation X.219, and it is a value of such a type that the macro returns. If none is provided, the default ÒUNBINDÓ is returned.

Note Ñ The relationship of ABSTRACTÑUNBIND and UNBIND helps make trivial the ROS realization of an abstract service; see ¤ 10.1.

An abstract service typically comprises an abstract unbind operation for each type of port involved in its provision. When several port types are involved, their abstract unbind operations may but need not be distinct.

8.4 *Abstract operations*

An **abstract operation** is a procedure that may be invoked in the context of two bound ports. Its failure has no effect upon the binding. If the ports are assymmetric, whether the invoker is the object having the consumer port, the object having the supplier port, or either is prescribed by the port. If the ports are symmetric, the invoker may be either object. Whether the ports are symmetric or asymmetric, the remaining object is the performer.

An abstract operation is a fully general procedure except for the information conveyed to the invoker upon failure. An abstract operation fails when it encounters an *abstract error*, and the information conveyed is constrained to that required to report that abstract error. Whether failure is reported and, if so, which abstract errors can be encountered are prescribed for each abstract operation.

An abstract operation is specified by means of the ABSTRACTÑOPERATION macro. Its definition is identical to that of the OPERATION macro of Remote Operations, specified in Recommendation X.219.

```
ABSTRACTÑOPERATION MACRO ::= OPERATION
```

An abstract service comprises zero or more abstract operations for each type of port involved in its provision. When several port types are involved, they may but need not have abstract operations in common.

Note Ñ The equivalence of ABSTRACTÑOPERATION and OPERATION helps make trivial the ROS realization of an abstract service; see ¤ 10.1.

8.5 *Abstract errors*

An **abstract error** is an exceptional condition that may arise during the performance of an abstract operation, causing it to fail.

When an abstract error is reported, the performer conveys to the invoker the identity of the abstract error and possibly a single information object called its **parameter**. Whether a parameter is returned and, if so, its type are prescribed for each abstract error.

An abstract error is specified by means of the ABSTRACTÑERROR macro. Its definition is identical to that of the ERROR macro of Remote Operations, specified in Recommendation X.219.

ABSTRACTÑERROR; MACRO ::= ERROR

An abstract service comprises the zero or more abstract errors reported by its abstract operations.

Note Ñ The equivalence of ABSTRACTÑERROR and ERROR helps make trivial the ROS realization of an abstract service; see ¤ 10.1.

SECTION 3 Ñ ABSTRACT SERVICE REALIZATIONS

9 Overview

Once a distributed information processing task has been described and specified in abstract terms, the manner in which each aspect of the task is to be concretely realized must be prescribed. As suggested previously, each aspect may admit of several concrete realizations.

This section specifies principles for concretely realizing abstract models and services. A **real** *x* is the computer process or system, or the real open system that concretely realizes an abstract object of type *x*.

This section covers the following topics:

a) OSI realizations.

b) Proprietary realizations.

Note Ñ The aspects of an abstract model stressed here are abstract ports and their bindings. This is because abstract ports mark the boundary not only between abstract objects but also between the physical systems that concretely realize those abstract objects. Thus abstract ports and bindings are the parts of an abstract model that must be constructed or constructable with OSI tools if open systems interworking is to occur.

10 OSI realizations

A primary objective of CCITT Recommendations and ISO Standards is to specify how distributed information processing tasks are realized when carried out by several cooperating real open systems.

In the OSI environment, objects are realized by means of application processes, with, in general, a manyÑtoÑmany mapping of objects to application processes. Communication among objects which are realized by application processes in different open systems is accomplished by OSI application protocols (consisting of application contexts). An application context thus realizes the binding, use, and unbinding of a number of port pairs.

The specification of an application context is in terms of the coordinated operation of a number of applicationÑserviceÑelements. Realization is therefore particularly straightforward to specify if an applicationÑserviceÑelement is defined to correspond to each port whose communication is to be supported.

The realization of abstract ports and bindings by means of ASEs and ACs is discussed below. Both ROS and nonÑROS realizations are considered.

10.1 *ROS realizations*

The concrete realization of ports and bindings is often trivial when accomplished by means of Remote Operations.

This is true because it is straightforward to define an abstract service which is such that there exists a ROSÑbased application protocol that is functionally identical to it. This is true in turn because the framework for the specification of abstract services is isomorphic to that for the specification of ROSÑbased application protocols. The correspondences behind the isomorphism are listed in Table 1/X.407.

TABLE 1/X.407

Correspondences of abstract services and ROSÑbased protocols

Aspect of Abstract service	Aspect of ROSÑbased protocol
Abstract bind operation	Bind operation
Abstract unbind operation	Unbind operation
Abstract operation	Operation
Abstract error	Error

The correspondences of the table arise from the fact that corresponding aspects are formally specified using closelyÑrelated, or equivalent macros, as summarized in Table 2/X.407.

TABLE 2/X.407

Equivalent abstract service and ROS macros

Abstract service macro	ROS macro
ABSTRACTÑBIND	BIND
ABSTRACTÑUNBIND	UNBIND
ABSTRACTÑOPERATION	OPERATION
ABSTRACTÑERROR	ERROR

The definition of ROSÑbased ASEs and ACs that concretely realize abstract ports is explored in Annex A by means of an example.

For the realization to be trivial, it is necessary that there be an abstract bind operation which binds all of the ports which must be paired.

Note Ñ Where there is more than one port (pair) involved in the abstract service, this requires that the abstract bind operation be designed for the particular ports involved. There is (currently) no provision for the automatic synthesis of a suitable abstract bind based, e.g., upon the definitions of abstract bind operations defined for the individual ports.

10.2 *NonÑROS realizations*

The concrete realization of ports and bindings is a more substantial task when attempted by means other than Remote Operations, and little can be said about the general propositions.

Despite the above, the following two observations are relevant:

a) The concrete realization of an abstract service as an application protocol is greatly simplified by using ASN.1 to define its APDUs. This is so because the protocol specification can simply import relevant types and values from the abstract service specification.

b) The concrete realization of an abstract service whose abstract operations do not report their outcomes is conceptually simple. This is so because each such abstract operation represents an interaction comprising a single APDU. From this simplest of all possible interactions, arbitrarily complex ones can be constructed.

11 Proprietary realizations

A secondary objective of CCITT Recommendations and ISO Standards is to ensure that those portions of a distributed information processing task that are carried out by proprietary means are accomplished in such a way that the intended overall functionality of the system is upheld.

The realization of abstract ports and bindings by proprietary means is briefly discussed below. Both distributed and nonÑdistributed realizations are considered.

11.1 *Distributed realizations*

The concrete realization of ports and bindings by means of proprietary computer communication protocols is a local matter. The specification of the visible functionality embodied in the abstract service provides a guide to the implementors of the proprietary realizations, so that, where such realizations are appropriate, they may play the appropriate role in the overall task.

11.2 *NonÑdistributed realizations*

The concrete realization of ports and bindings by means of mechanisms wholly within a single computer is a local matter. As with the case considered in ¤ 11.1, the abstract service specification serves as a guide to the implementor in ensuring that the proprietary realization can nonetheless play the appropriate role in the overall task.

ANNEX A

(to Recommendation X.407)

Example of use of abstract service notation

This Annex is not a part of this Recommendation.

This Annex illustrates the use of the abstract model and service notation by means of an example. The example involves two systems, the *Yellow* and *Green Systems*, and their environments, the **Yellow** and **Green Environments**.

It uses the abstract model notation to describe the environments separately (¤ A.2 and A.4) and to show how their systems are related: one is constructed from the other (¤ A.6). It uses the abstract service notation to describe the capabilities of each system (¤ A.3 and A.5). The example concludes by realizing the systems' ports as ACs and ASEs using the ROS notation of Recommendation X.219, as might be appropriate for OSI communication (¤ A.7 and A.8).

A.1 *Assignment of object identifiers*

The ASN.1 modules defined in this Annex require the assignment of a variety of Object Identifiers. All are defined below using ASN.1. The assignments are definitive except for those for ASN.1 modules and the subject of application service definition conventions itself. The definitive assignments for the former occur in the modules themselves; other references to them appear in IMPORT clauses. The latter is fixed.

```
ExampleObjectIdentifiers { jointÑisoÑccitt
    mhsÑmotis(6) asdc(2) example(1) modules(0) objectÑidentifiers(0) }
DEFINITIONS IMPLICIT TAGS ::=
BEGIN
```

ÑÑ *Prologue*
ÑÑ *Exports everything.*

IMPORTS ÑÑ *nothing* ÑÑ ;

ID ::= OBJECT IDENTIFIER

ÑÑ *Abstract Service Definition Conventions Example (not definitive)*

idÑasdcÑex ID ::= { jointÑisoÑccitt mhsÑmotis (6) asdc(2) example(1) }
ÑÑ *not definitive*

ÑÑ *Categories*

idÑmod ID ::= { idÑasdcÑex 0 } ÑÑ *modules; not definitive*
idÑot ID ::= { idÑasdcÑex 1 } ÑÑ *object types*
idÑpt ID ::= { idÑasdcÑex 2 } ÑÑ *port types*
idÑref ID ::= { idÑasdcÑex 3 } ÑÑ *refinements*
idÑac ID ::= { idÑasdcÑex 4 } ÑÑ *application contexts*
idÑase ID ::= { idÑasdcÑex 5 } ÑÑ *application service elements*
idÑas ID ::= { idÑasdcÑex 6 } ÑÑ *abstract syntaxes*

ÑÑ *Modules*

idÑmodÑobjectÑidentifiers ID ::= { idÑmod 0 } ÑÑ *not definitive*
idÑmodÑyeÑrefinement ID ::= { idÑmod 1 } ÑÑ *not definitive*
idÑmodÑyÑabstractÑservice ID ::= { idÑmod 2 } ÑÑ *not definitive*
idÑmodÑgeÑrefinement ID ::= { idÑmod 3 } ÑÑ *not definitive*
idÑmodÑgÑabstractÑservice ID ::= { idÑmod 4 } ÑÑ *not definitive*
idÑmodÑysÑrefinement ID ::= { idÑmod 5 } ÑÑ *not definitive*
idÑmodÑysÑrealization ID ::= { idÑmod 6 } ÑÑ *not definitive*
idÑmodÑgsÑrealization ID ::= { idÑmod 7 } ÑÑ *not definitive*

ÑÑ *Object types*

idÑotÑyÑenvironment ID ::= { idÑot 0 }
idÑotÑyÑuser ID ::= { idÑot 1 }
idÑotÑyÑsystem ID ::= { idÑot 2 }
idÑotÑgÑenvironment ID ::= { idÑot 3 }
idÑotÑgÑuser ID ::= { idÑot 4 }
idÑotÑgÑmanager ID ::= { idÑot 5 }
idÑotÑgÑsystem ID ::= { idÑot 6 }
idÑotÑagent ID ::= { idÑot 7 }

ÑÑ *Port types*

idÑptÑyÑuse ID ID ::= { idÑpt 0 }
idÑptÑgÑuse ID ID ::= { idÑpt 1 }
idÑptÑgÑmanagement ID ::= { idÑpt 2 }

ÑÑ *Refinements*

idÑrefÑyÑenvironment ID ::= { idÑref 0 }
idÑrefÑgÑenvironment ID ::= { idÑref 1 }
idÑrefÑyÑsystem ID ::= { idÑref 2 }

ÑÑ *Application contexts*

idÑacÑyÑuse ID ::= { idÑac 0 }
idÑacÑgÑuse ID ::= { idÑac 1 }
idÑacÑgÑmanagement ID ::= { idÑac 2 }

ÑÑ *Application service elements*

idÑaseÑyÑuse ID ::= { idÑase 0 }
idÑaseÑgÑuse ID ::= { idÑase 1 }
idÑaseÑgÑmanagement ID ::= { idÑase 2 }

ÑÑ *Abstract syntaxes*

```
idÑasÑyÑuse             ID ::= { idÑas 0 }
idÑasÑgÑuse             ID ::= { idÑas 1 }
idÑasÑgÑmanagement      ID ::= { idÑas 2 }

END ÑÑ of ExampleObjectIdentifiers
```

A.2 *Refinement of yellow environment*

The Yellow Environment, depicted in Figure AÑ1/X.407, is formally refined below using the OBJECT and REFINE macros.

Fig. AÑ1/X.407/T0704110-88 = 8 cm

As Figure AÑ1/X.407 indicates and the ASN.1 specification below confirms, the Yellow Environment can be modeled as an object which can be decomposed into one central object, the **Yellow System**, and any number of other, peripheral objects, **yellow users**. The Yellow System interacts with yellow users by means of its **yellowÑuse** ports.

```
YellowEnvironmentRefinement { jointÑisoÑccitt
    mhsÑmotis(6) asdc(2) example(1) modules(0) yeÑrefinement(1) }
DEFINITIONS IMPLICIT TAGS ::=
BEGIN
ÑÑ Prologue

EXPORTS
    yellowÑenvironment, yellowÑenvironmentÑrefinement,
    yellowÑsystem, yellowÑuser;

IMPORTS
    ÑÑ Yellow Abstract Service
        yellowÑuse
        . . . .
        FROM YellowAbstractService { jointÑisoÑccitt
            mhsÑmotis(6) asdc(2) example(1) modules(0) yÑabstractÑservice(2) }

    ÑÑ Example Object Identifiers

        idÑotÑyÑenvironment, idÑotÑyÑsystem, idÑotÑyÑuser,
        idÑrefÑyÑenvironment
        . . . .
        FROM ExampleObjectIdentifiers { jointÑisoÑccitt
            mhsÑmotis(6) asdc(2) example(1) modules(0) objectÑidentifiers(0) }

    ÑÑ Abstract Service Notation
        OBJECT, REFINE
        . . . .
        FROM AbstractServiceNotation { jointÑisoÑccitt
            mhsÑmotis(6) asdc(2) modules(0) notation(1) };

ÑÑ Yellow Environment

yellowÑenvironment OBJECT
    ::= idÑotÑyÑenvironment

ÑÑ Yellow Environment refinement

yellowÑenvironmentÑrefinement REFINE yellowÑenvironment AS
    yellowÑuser RECURRING
```

```
        yellowÑsystem
            yellowÑuse [S] PAIRED WITH yellowÑuser
        ::= idÑrefÑyÑenvironment

ÑÑ Component object types

yellowÑuser OBJECT
        PORTS {
                yellowÑuse [C] }
        ::= idÑotÑyÑuser

yellowÑsystem OBJECT

        PORTS {
                yellowÑuse [S] }
        ::= idÑotÑyÑsystem

END ÑÑ of YellowEnvironmentRefinement
```

A.3 *Definition of yellow abstract service*

The abstract service that the Yellow System provides to its users is formally defined below using the PORT and ABSTRACTÑBIND, ÑOPERATION, and ÑERROR macros.

As the ASN.1 specification indicates, the abstract service that the Yellow System provides comprises ports of a single kind, yellowÑuse. Each port comprises a number of abstract operations which collectively report a number of abstract errors. The Yellow System guards its ports by means of an abstract bind operation, **YellowBind**, which demands that users identify themselves convincingly before further interaction occurs. An abstract unbind operation, **YellowUnbind**, which constitutes the finalization step required to conclude an interaction.

```
YellowAbstractService { jointÑisoÑccitt
        mhsÑmotis(6) asdc(2) example(1) modules(0) yÑabstractÑservice(2) }
DEFINITIONS IMPLICIT TAGS ::=
BEGIN
ÑÑ Prologue

EXPORTS
        AuthenticateUser, YellowÑoperationÑ1, . . . yellowÑuse;

IMPORTS
        ÑÑ Example Object Identifiers
                idÑptÑyÑuse
                . . . .
                FROM ExampleObjectIdentifiers { jointÑisoÑccitt
                    mhsÑmotis(6) asdc(2) example(1) modules(0) objectÑidentifiers(0) }

        ÑÑ Abstract Service Notation
                ABSTRACTÑBIND, ABSTRACTÑERROR, ABSTRACTÑOPERATION, PORT
                . . . .
                FROM AbstractServiceNotation { jointÑisoÑccitt
                    mhsÑmotis(6) asdc(2) modules(0) notation(1) };

ÑÑ Port type

yellowÑuse PORT
        CONSUMER INVOKES {
                YellowÑoperationÑ1, . . . }
        ::= idÑptÑyÑuse

ÑÑ Abstract bind operation

Credentials ::= SET {
                name  [0] IA5String,
                password        [1] IA5String }

YellowBind ::= ABSTRACTÑBIND
                TO { yellowÑuse[S] }
```

```
        BIND

                ARGUMENT credentials Credentials
                    BINDÑERROR ENUMERATED {
                      nameÑorÑpasswordÑinvalid(0) }
```

ÑÑ *Abstract unbind operation*

```
YellowUnbind ::= ABSTRACTÑUNBIND
        FROM { yellowÑuse[S] }
```

ÑÑ *Abstract operations*

```
YellowÑoperationÑ1 ::= ABSTRACTÑOPERATION
                ARGUMENT . . .
                RESULT . . .
                ERRORS {
                    yellowÑerrorÑ1, . . . }

. . .
```

ÑÑ *Abstract errors*

```
yellowÑerrorÑ1 ABSTRACTÑERROR
                PARAMETER . . .
                ::= 1

. . .
```

END ÑÑ *of YellowAbstractService*

A.4 *Refinement of green environment*

The Green Environment, depicted in Figure AÑ2/X.407, is formally refined below using the OBJECT and REFINE macros.

Fig. AÑ2/X.407/T0704120-88 = 8cm

As Figure AÑ2/X.407 indicates and the ASN.1 specification below confirms, the Green Environment can be modeled as an object which can be decomposed into one central object, the **Green System**; any number of other, peripheral objects, **green users**; and any number of yet additional objects, **green managers**. The Green System interacts with green users and managers by means of its **greenÑuse** ports, and with green managers (alone) by means of its **greenÑmanagement** ports.

```
GreenEnvironmentRefinement { jointÑisoÑccitt
        mhsÑmotis(6) asdc(2) example(1) modules(0) geÑrefinement(3) }
DEFINITIONS IMPLICIT TAGS ::=
BEGIN
ÑÑ Prologue

EXPORTS
        greenÑenvironment, greenÑenvironmentÑrefinement,
        greenÑmanager, greenÑsystem, greenÑuser;

IMPORTS

        ÑÑ Green Abstract Service
                greenÑuse, greenÑmanagement
                . . . .
                FROM GreenAbstractService { jointÑisoÑccitt
                    mhsÑmotis(6) asdc(2) example(1) modules(0) gÑabstractÑservice(4) }

        ÑÑ Example Object Identifiers
                idÑotÑgÑenvironment, idÑotÑgÑmanager, idÑrefÑgÑenvironment
                idÑotÑgÑuser, idÑotÑgÑsystem,
                . . . .
                FROM ExampleObjectIdentifiers { jointÑisoÑccitt
                    mhsÑmotis(6) asdc(2) example(1) modules(0) objectÑidentifiers(0) }

        ÑÑ Abstract Service Notation
                OBJECT, REFINE
                . . . .
                FROM AbstractServiceNotation { jointÑisoÑccitt
                    mhsÑmotis(6) asdc(2) modules(0) notation(1) };
```

ÑÑ *Green Environment*

```
greenÑenvironment OBJECT
        ::= idÑotÑgÑenvironment
```

ÑÑ *Green Environment refinement*

```
greenÑenvironmentÑrefinement REFINE greenÑenvironment AS
        greenÑuser      RECURRING
        greenÑmanager RECURRING
        greenÑsystem
            greenÑuse           [S] PAIRED WITH greenÑuser, greenÑmanager
            greenÑmanagement [S] PAIRED WITH greenÑmanager
        ::= idÑrefÑgÑenvironment
```

ÑÑ *Component object types*

```
greenÑuser OBJECT
        PORTS {
                greenÑuse           [C] }
        ::= idÑotÑgÑuser

greenÑmanager OBJECT
        PORTS {
                greenÑuse           [C],
                greenÑmanagement   [C] }
        ::= idÑotÑgÑmanager

greenÑsystem OBJECT
        PORTS {
                greenÑuse           [S],
                greenÑmanagement   [S] }
        ::= idÑotÑgÑsystem
```

END ÑÑ *of GreenEnvironmentRefinement*

A.5 *Definition of green abstract service*

The abstract service that the Green System provides to its users and managers is formally defined below using the PORT and ABSTRACTÑBIND, ÑOPERATION, and ÑERROR macros.

As the ASN.1 specification indicates, the abstract service that the Green System provides comprises ports of two kinds, greenÑuse and greenÑmanagement. A port of either kind comprises a number of abstract operations which collectively report a number of abstract errors. The Green System guards its ports by means of abstract bind operations, **AuthenticateUser** and **AuthenticateManager**, which demand that users and managers identify themselves convincingly before further interaction can occur. No abstract unbind operations are specified, indicating that no finalization step is required to conclude an interaction.

```
GreenAbstractService { jointÑisoÑccitt
        mhsÑmotis(6) asdc(2) example(1) modules(0) gÑabstractÑservice(4) }
DEFINITIONS IMPLICIT TAGS ::=
BEGIN
ÑÑ Prologue

EXPORTS
        AuthenticateManager, AuthenticateUser, greenÑmanagement,
        GreenÑmanagementÑoperationÑ1, . . . greenÑuse,
        GreenÑuseÑoperationÑ1, . . .;

IMPORTS
        ÑÑ Example Object Identifiers
                idÑptÑgÑuse, idÑptÑgÑmanagement
                . . . .
```

```
                FROM ExampleObjectIdentifiers { jointÑisoÑccitt
                    mhsÑmotis(6) asdc(2) example(1) modules(0) objectÑidentifiers(0) }

        ÑÑ Abstract Service Notation
                PORT, ABSTRACTÑBIND, ABSTRACTÑOPERATION, ABSTRACTÑERROR
                . . . .
                FROM AbstractServiceNotation { jointÑisoÑccitt
                    mhsÑmotis(6) asdc(2) modules(0) notation(1) };

ÑÑ Port types

greenÑuse PORT
        CONSUMER INVOKES {
                GreenÑuseÑoperationÑ1, . . . }
        ::= idÑptÑgÑuse

greenÑmanagement PORT
        CONSUMER INVOKES {
                GreenÑmanagementÑoperationÑ1, . . . }
        ::= idÑptÑgÑmanagement

ÑÑ Abstract bind operations

Credentials ::= SET {
        name     [0] IA5String,
        password [1] IA5String }

AuthenticateUser ::= ABSTRACTÑBIND
        ARGUMENT credentials Credentials
        BINDÑERROR ENUMERATED {
            nameÑorÑpasswordÑinvalid(0) }

AuthenticateManager ::= ABSTRACTÑBIND
        ARGUMENT credentials Credentials
        BINDÑERROR ENUMERATED {
            nameÑorÑpasswordÑinvalid(0),
            notÑaÑmanager              (1) }

ÑÑ Abstract operations

GreenÑuseÑoperationÑ1 ::= ABSTRACTÑOPERATION
        ARGUMENT . . .
        RESULT . . .
        ERRORS {
                greenÑerrorÑ1, . . . }

. . .

GreenÑmanagementÑoperationÑ1 ::= ABSTRACTÑOPERATION
        ARGUMENT . . .
        RESULT . . .
        ERRORS {
            greenÑerrorÑ1, . . . }

. . .

ÑÑ Abstract errors

greenÑerrorÑ1 ABSTRACTÑERROR
        PARAMETER . . .
        ::= 1

. . .

END ÑÑ of GreenAbstractService
```

A.6 *Refinement of yellow system*

The Yellow System, depicted in Figure AÑ3/X.407, is formally refined below using the OBJECT and REFINE macros.

Fig. AÑ3/X.407/T0704130-88 = 8cm

As the figure indicates and the ASN.1 specification confirms, the Yellow System, when examined closely, has components. In particular, the Yellow System comprises the Green System and green managers, augmented by objects of an as yet unseen variety, *agent*. An **agent** serves as an intermediary between the Green System and a yellow user. It might be thought of as adding value to the Green System. In any case, it is a provider of a yellowÑuse port and a consumer of a greenÑuse port.

```
YellowSystemRefinement { jointÑisoÑccitt
        mhsÑmotis(6) asdc(2) example(1) modules(0) ysÑrefinement(5) }
DEFINITIONS IMPLICIT TAGS ::=
BEGIN
ÑÑ Prologue

EXPORTS
        agent, yellowÑsystemÑrefinement;

IMPORTS
        ÑÑ Yellow Environment Refinement
                yellowÑsystem, yellowÑuse
                . . . .
                FROM YellowEnvironmentRefinement { jointÑisoÑccitt
                    mhsÑmotis(6) asdc(2) example(1) modules(0) yeÑrefinement(1) }

        ÑÑ Green Environment Refinement
                greenÑmanagement, greenÑmanager, greenÑsystem,
                greenÑuse
                . . . .
                FROM GreenEnvironmentRefinement { jointÑisoÑccitt
                    mhsÑmotis(6) asdc(2) example(1) modules(0) geÑrefinement(3) }

        ÑÑ Example Object Identifiers
                idÑotÑagent, idÑrefÑyÑsystem
                . . . .
                FROM ExampleObjectIdentifiers { jointÑisoÑccitt
                    mhsÑmotis(6) asdc(2) example(1) modules(0) objectÑidentifiers(0) }

        ÑÑ Abstract Service Notation
        OBJECT, REFINE
            FROM AbstractServiceNotation { jointÑisoÑccitt
                mhsÑmotis(6) asdc(2) modules(0) notation(1) };

ÑÑ Yellow System refinement

yellowÑsystemÑrefinement REFINE yellowÑsystem AS
        agent RECURRING
                yellowÑuse      [S] VISIBLE
        greenÑmanager RECURRING
        greenÑsystem
                greenÑuse       [S] PAIRED WITH agent, greenÑmanager
                greenÑmanagement        [S] PAIRED WITH greenÑmanager
        ::= idÑrefÑyÑsystem

ÑÑ Component object type

agent OBJECT
        PORTS {
```

```
            yellowÑuse      [S],
            greenÑuse       [C] }
    ::= idÑotÑagent

END ÑÑ of YellowSystemRefinement
```

A.7 *Realization of yellow system*

The abstract service of the Yellow System is formally realized below, by means of ROS, using the APPLICATIONÑCONTEXT and APPLICATIONÑSERVICEÑELEMENT macros of Recommendation X.219.

As the ASN.1 specification indicates, the abstract service that the Yellow System provides is realized as a single ASE, **yellowÑuseÑASE**, and a single and corresponding AC, **yellowÑuseÑAC**. Each abstract bind operation, abstract operation, or abstract error in the abstract service has a corresponding and equivalent bind operation, operation, or error, respectively, in its ROSÑbased realization.

Note that Integer values are assigned to the operations; the corresponding abstract operations require and received no such values.

```
YellowSystemRealization { jointÑisoÑccitt
    mhsÑmotis(6) asdc(2) example(1) modules(0) ysÑrealization(6) }
DEFINITIONS IMPLICIT TAGS ::=
BEGIN
ÑÑ Prologue

EXPORTS
    yellowÑuseÑAC, yellowÑuseÑASE;

IMPORTS
    ÑÑ Yellow Abstract Service
        YellowÑoperationÑ1, . . . yellowÑuse, YellowBind,
        YellowUnbind
        . . . .
        FROM YellowAbstractService { jointÑisoÑccitt
            mhsÑmotis(6) asdc(2) example(1) modules(0) yÑabstractÑservice(2) }

    ÑÑ Example Object Identifiers
        idÑacÑyÑuse, idÑasÑyÑuse, idÑaseÑyÑuse
        . . . .
        FROM ExampleObjectIdentifiers { jointÑisoÑccitt
            mhsÑmotis(6) asdc(2) example(1) modules(0) objectÑidentifiers(0) }

    ÑÑ Remote Operations APDUs
        rOSE
        . . . .
        FROM RemoteÑOperationsÑAPDUs { jointÑisoÑccitt
            remoteÑoperations(4) apdus(1) }

    ÑÑ Association Control
        aCSE
        . . . .
        FROM RemoteÑOperationÑNotationÑextension
            { jointÑisoÑccitt remoteÑoperations(4)
            notationÑextension(2) }

    ÑÑ Remote Operations Notation Extension
        APPLICATIONÑCONTEXT, APPLICATIONÑSERVICEÑELEMENT
        . . . .
        FROM RemoteÑOperationsÑNotationÑextension { jointÑisoÑccitt
            remoteÑoperations(4) notationÑextension(2) };

ACSEÑAS OBJECT IDENTIFIER ::=
        { jointÑisoÑccitt associationÑcontrol(2)
        abstractSyntax(1) apdus(0) version1(1) }
```

ÑÑ *Application context*

yellowÑuseÑAC APPLICATIONÑCONTEXT
APPLICATION SERVICE ELEMENTS { aCSE }
BIND YellowBind
UNBIND YellowUnbind
REMOTE OPERATIONS { rOSE }
INITIATOR CONSUMER OF { yellowÑuseÑASE }
ABSTRACT SYNTAXES { yellowÑuseÑAS, aCSEÑAS }
::= idÑacÑyÑuse

ÑÑ *Application service element*

yellowÑuseÑASE APPLICATIONÑSERVICEÑELEMENT
CONSUMER INVOKES {
yellowÑoperationÑ1, . . . }
::= idÑaseÑyÑuse

yellowÑoperationÑ1 YellowÑoperationÑ1 ::= 1
. . .

ÑÑ *Abstract syntax*

yellowÑuseÑAS OBJECT IDENTIFIER ::= idÑasÑyÑuse

END ÑÑ *of YellowSystemRealization*

A.8 *Realization of green system*

The abstract service of the Green System is formally realized below, by means of ROS, using the APPLICATIONÑCONTEXT and APPLICATIONÑSERVICEÑELEMENT macros of Recommendation X.219.

As the ASN.1 specification indicates, the abstract service that the Green System provides is realized as two ASEs, **greenÑuseÑASE** and **greenÑmanagementÑASE**, and two, corresponding ACs, **greenÑuseÑAC** and **greenÑmanagementÑAC**. Each abstract bind operation, abstract operation, or abstract error in the abstract service has a corresponding and equivalent bind operation, operation, or error, respectively, in its ROSÑbased realization.

Note that Integer values are assigned to the operations; the corresponding abstract operations require and received no such values.

GreenSystemRealization { jointÑisoÑccitt
mhsÑmotis(6) asdc(2) example(1) modules(0) gsÑrealization(7) }
DEFINITIONS IMPLICIT TAGS ::=
BEGIN
ÑÑ *Prologue*

EXPORTS
greenÑmanagementÑAC, greenÑmanagementÑASE, greenÑuseÑAC,
greenÑuseÑASE;

IMPORTS
ÑÑ *Green Abstract Service*
AuthenticateManager, AuthenticateUser,
greenÑmanagement, GreenÑmanagementÑoperationÑ1, . . .
greenÑuse, GreenÑuseÑoperationÑ1, . . .
. . . .
FROM GreenAbstractService { jointÑisoÑccitt
mhsÑmotis(6) asdc(2) example(1) modules(0) gÑabstractÑservice(4) }

ÑÑ *Example Object Identifiers*
idÑacÑgÑuse, idÑaseÑgÑuse, idÑasÑgÑuse,
idÑacÑgÑmanagement, idÑaseÑgÑmanagement, idÑasÑgÑmanagement
. . . .
FROM ExampleObjectIdentifiers { jointÑisoÑccitt
mhsÑmotis(6) asdc(2) example(1) modules(0) objectÑidentifiers(0) }

```
        ÑÑ Remote Operations APDUs
                rOSE
                . . . .
                FROM RemoteÑOperationsÑAPDUs { jointÑisoÑccitt
                    remoteÑoperations(4) apdus(1) }

        ÑÑ Association Control
                aCSE
                . . . .
                FROM RemoteÑOperationsÑNotationÑextension { jointÑisoÑccitt
                    remoteÑoperations(4) notationÑextension(2) }

        ÑÑ Remote Operations Notation Extension
                APPLICATIONÑCONTEXT, APPLICATIONÑSERVICEÑELEMENT
                . . . .
                FROM RemoteÑOperationsÑNotationÑextension { jointÑisoÑccitt
                    remoteÑoperations(4) notationÑextension(2) };

aCSEÑAS OBJECT IDENTIFIER ::=
                { jointÑisoÑccitt associationÑcontrol(2)
                abstractSyntax(1) apdus(0) version1(1) }

ÑÑ Application contexts

greenÑuseÑAC APPLICATIONÑCONTEXT
            APPLICATION SERVICE ELEMENTS { aCSE }
            BIND AuthenticateUser
            UNBIND NoOperation
            REMOTE OPERATIONS { rOSE }
            INITIATOR CONSUMER OF { greenÑuseÑASE }
            ABSTRACT SYNTAXES { greenÑuseÑAS, aCSEÑAS }
            ::= idÑacÑgÑuse

greenÑmanagementÑAC APPLICATIONÑCONTEXT
            APPLICATION SERVICE ELEMENTS { aCSE }
            BIND AuthenticateManager
            UNBIND NoOperation
            REMOTE OPERATIONS { rOSE }
            INITIATOR CONSUMER OF { greenÑmanagementÑASE }
            ABSTRACT SYNTAXES { greenÑmanagementÑAS, aCSEÑAS }
            ::= idÑacÑgÑmanagement

NoOperation ::= UNBIND

ÑÑ Application service elements

greenÑuseÑASE APPLICATIONÑSERVICEÑELEMENT
            CONSUMER INVOKES {
                    greenÑuseÑoperationÑ1, . . . }
            ::= idÑaseÑgÑuse

greenÑmanagementÑASE APPLICATIONÑSERVICEÑELEMENT
            CONSUMER INVOKES {
                greenÑmanagementÑoperationÑ1, . . .}
            ::= idÑaseÑgÑmanagement

greenÑuseÑoperationÑ1 GreenÑuseÑoperationÑ1 ::= 1
. . .

greenÑmanagementÑoperationÑ1 GreenÑmanagementÑoperationÑ1 ::= 50 ...

ÑÑ Abstract syntaxes

greenÑuseÑAS OBJECT IDENTIFIER ::= idÑasÑgÑuse

greenÑmanagementÑAS OBJECT IDENTIFIER ::= idÑasÑgÑmanagement
```

END ÑÑ *of GreenSystemRealization*

ANNEX B

(to Recommendation X.407)

Reference definition of object identifiers

This Annex is an integral part of this Recommendation.

This Annex defines for reference purposes various Object Identifiers cited in the ASN.1 modules of Annex C. It uses ASN.1.

With the exception of those assigned in Annex A, all Object Identifiers this Recommendation assigns are assigned in this Annex. The Annex is definitive for all but those for ASN.1 modules and the subject of application service definition conventions itself. The definitive assignments for the former occur in the modules themselves; other references to them appear in IMPORT clauses. The latter is fixed.

```
ASDCObjectIdentifiers { jointÑisoÑccitt
        mhsÑmotis(6) asdc(2) modules(0) objectÑidentifiers(0) }
DEFINITIONS IMPLICIT TAGS ::=
BEGIN
ÑÑ Prologue

ÑÑ Exports everything.

IMPORTS ÑÑ nothingÑÑ;

ID ::= OBJECT IDENTIFIER

ÑÑ Abstract Service Definition Conventions (not definitive)

idÑasdc  ID ::= { jointÑisoÑccitt mhsÑmotis(6) asdc(2) } ÑÑ not definitive

ÑÑ Categories

idÑmod  ID ::= { idÑasdc 0 } ÑÑ modules; not definitive
idÑex   ID ::= { idÑasdc 1 } ÑÑ example; not definitive

ÑÑ Modules

idÑmodÑobjectÑidentifiers      ID ::= { idÑmod 0 } ÑÑ not definitive
idÑmodÑnotation                ID ::= { idÑmod 1 } ÑÑ not definitive

END ÑÑ of ASDCObjectIdentifiers
```

ANNEX C

(to Recommendation X.407)

Reference definition of notation

This Annex is an integral part of this Recommendation.

This Annex, a supplement to section 2, defines for reference purposes the notation for specifying abstract models and services. It employs ASN.1.

```
AbstractServiceNotation { jointÑisoÑccitt
        mhsÑmotis(6) asdc(2) modules(0) notation(1) }
```

```
DEFINITIONS IMPLICIT TAGS ::=
BEGIN
ÑÑ Prologue

EXPORTS

        ABSTRACTÑBIND, ABSTRACTÑERROR, ABSTRACTÑOPERATION,
        ABSTRACTÑUNBIND, OBJECT, PORT, REFINE;

IMPORTS

        ÑÑ Remote Operations Notation
                BIND, ERROR, OPERATION, UNBIND
                . . . .
                FROM RemoteÑOperationÑNotation { jointÑisoÑccitt
                    remoteÑoperations(4) notation(0) };

ÑÑ Object macro

OBJECT; MACRO ::=
BEGIN

TYPE NOTATION       ::=     ÒPORTSÓ Ò{Ó PortList Ò}Ó | empty
VALUE NOTATION      ::=     value (VALUE OBJECT IDENTIFIER)

PortList            ::=     Port Ò , Ó PortList | Port
Port                ::=     value (PORT) PortType

PortType            ::=     Symmetric | Asymmetric

Symmetric           ::=     empty
Asymmetric          ::=     Consumer | Supplier

Consumer            ::=     Ò[C]Ó
Supplier            ::=     Ò[S]Ó

END

ÑÑ Port macro

PORT MACRO ::=
BEGIN

TYPE NOTATION       ::=     Operations | empty
VALUE NOTATION      ::=     value (VALUE OBJECT IDENTIFIER)

Operations          ::=     Symmetrical | Asymmetrical

Symmetrical         ::=     ÓABSTRACTÓ ÒOPERATIONSÓ Ò{Ó OperationList Ò}Ó
Asymmetrical        ::=     OneSided | TwoSided

OneSided            ::=     Consumer | Supplier
TwoSided            ::=     Consumer Supplier | Supplier Consumer

Consumer            ::=     ÒCONSUMERÓ ÒINVOKESÓ Ò{Ó OperationList Ò}Ó
Supplier            ::=     ÒSUPPLIERÓ ÒINVOKESÓ Ò{Ó OperationList Ò}Ó

OperationList       ::=     Operation Ò , Ó OperationList | Operation
Operation           ::=     value (ABSTRACTÑOPERATION)

END

ÑÑ Refine macro

REFINE MACRO ::=
BEGIN

TYPE NOTATION       ::=     Object ÒASÓ ComponentList
VALUE NOTATION      ::=     value (VALUE OBJECT IDENTIFIER)
```

```
ComponentList       ::=   Component ComponentList | Component
Component           ::=   ObjectSpec PortSpecList

ObjectSpec          ::=   Object | Object ÓRECURRINGÓ

PortSpecList        ::=   PortSpec PortSpecList | PortSpec
PortSpec            ::=   value (PORT) PortType PortStatus

PortType            ::=   Consumer | Supplier | empty
Consumer            ::=   Ò[C]Ó
Supplier            ::=   Ò[S]Ó

PortStatus          ::=   ÒVISIBLEÓ | ÒPAIREDÓ ÒWITHÓ ObjectList

ObjectList          ::=   Object Ò,Ó ObjectList | Object
Object              ::=   value (OBJECT)

END
```

ÑÑ *Abstract bind, unbind, operation, and error macros*

```
ABSTRACTÑBIND MACRO ::=
BEGIN

TYPE NOTATION       ::=   Ports Bind
VALUE NOTATION      ::=   value (VALUE BindType)

Ports               ::=   ÒTOÓ Ò{Ó PortList Ò}Ó | empty
PortList            ::=   Port Ò , Ó PortList | Port
Port                ::=   value (PORT) PortSide
PortSide            ::=   Consumer | Supplier | empty
Consumer            ::=   Ò[C]Ó
Supplier            ::=   Ò[S]Ó

Bind                ::=   type (BindType) ÑÑ must be a BIND type
                          | empty <BindType ::= BIND>

END

ABSTRACTÑUNBIND MACRO ::=
BEGIN

TYPE NOTATION       ::=   Ports Unbind
VALUE NOTATION      ::=   value (VALUE UnbindType)

Ports               ::=   ÒFROMÓ Ò{Ó PortList Ò}Ó | empty
PortList            ::=   Port Ò , Ó PortList | Port
Port                ::=   value (PORT) PortSide
PortSide            ::=   Consumer | Supplier | empty
Consumer            ::=   Ò[C]Ó
Supplier            ::=   Ò[S]Ó

Unbind              ::=   type (UnbindType) ÑÑ an UNBIND type
                          | empty <UnbindType ::= UNBIND>

END

ABSTRACTÑOPERATION MACRO ::= OPERATION

ABSTRACTÑERROR MACRO ::= ERROR

END ÑÑ of AbstractServiceNotation
```

ANNEX D

(to Recommendation X.407)

Differences between CCITT Recommendation and ISO Standard

This Annex is not a part of this Recommendation.

This Annex lists all but the purely stylistic differences between this Recommendation and the corresponding ISO International Standard.

No differences between the two specifications exist.

ANNEX E

(to Recommendation X.407)

Index

This Annex indexes this Recommendation. It gives the number(s) of the section(s) on which each item in each of several categories is defined. Its coverage of each category is exhaustive.

This Annex indexes items (if any) in the following categories:

a) abbreviations;

b) terms;

c) information items;

d) ASN.1 modules;

e) ASN.1 macros;

f) ASN.1 types;

g) ASN.1 values;

h) bilateral agreements;

i) items for further study;

j) items to be supplied.

E.1 *Abbreviations*

AC 3

APDU 3

ASE 3

ASN.1 3

OSI 3

ROS 3

E.2 *Terms*

abstract association 7.3

abstract bind operation 8.2

abstract error 8.5

abstract model 7

abstract object 7.1

abstract operation 8.4

abstract port 7.2

abstract procedure 8.1

abstract refinement 7.4

abstract service 7.3

abstract service provider 7.3

abstract service user 7.3

abstract unbind operation 8.3

argument 8.1

association 7.3

asymmetric 7.2

binding 7.2

bound 7.2

consumer 7.2

error information 8.3

Blancinitiator 8.2
invocation 8.1
invoker 8.1
match 7.2
model 7
object 7.1
parameter 8.5
performance 8.1
performer 8.1
port 7.2
procedure 8.1
provider 7.3
real 9
refinement 7.4
responder 8.2
result 8.1
supplier 7.2
symmetric 7.2
unbinding 7.2
user 7.3