SERVICE ORDER HANDLING

A DISSERTATION IN
Computer Networking
and
Telecommunications

Presented to the Faculty of the University
of Missouri-Kansas City in partial fulfillment of
the requirements for the degree

DOCTOR OF PHILOSOPHY

by

YOUNG BAE CHOI

B.S., Chonnam National University, Korea, 1982
M.S., Korea Advanced Institute of Science and Technology, 1985
M.S., University of Missouri-Kansas City, 1991

Kansas City, Missouri
1995

SERVICE ORDER HANDLING

Young Bae Choi, Doctor of Philosophy

University of Missouri-Kansas City, 1995

ABSTRACT

Current service order handling systems in telecommunications industry have problems such as user-unfriendliness, long service waiting time, uneasy customization of services, limited capability of in-house services, and inefficient human resources allocation. Consequently, the current solutions for the ordering process cannot be used to meet the business needs of service providers. To solve this problem, a generic service order handling model is suggested. Based on the TINA-C information modeling and computation modeling concepts, service order handling information object types and interfaces were defined.

The three ordering interfaces studied in this model are: service negotiation, service ordering and order tracking for the main service provider and subcontracted service provider.

To support the transfer of ordering information, the Store-and-Forward Paradigm is proposed. This Store-and-Forward paradigm is based on the use of e-mail (e.g., X.400) to transport non-realtime information.

This abstract of 135 words is approved as to form and content.

Dr. Adrian Tang, Professor

Computer Science Telecommunications

The undersigned, appointed by the Dean of the School of Graduate Studies, have examined a dissertation titled "Service Order Handling," presented by Young Bae Choi, candidate for the Doctor of Philosophy, and hereby certify that in their opinion it is worthy of acceptance.

12/15/95

Adrian Tang, Ph.D.
Computer Science Telecommunications

Date

Dec 14, 1995

Jonathan C. Oh, Ph.D.
Computer Science Telecommunications

Date

12/14/95

James K. Blundell, Ph.D.
Computer Science Telecommunications

Date

12/14/95

Lein Harn, Ph.D.
Computer Science Telecommunications

Date

12/14/95

Xiaojun Shen, Ph.D.
Computer Science Telecommunications

Date

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure

ACKNOWLEDGMENTS

I would like to express my thanks to all the committee members for their advice on this dissertation, especially my academic advisor, Dr. Adrian Tang, for his generous and continuous support throughout my research work and completing this dissertation.

I owe a very special thanks to my wife [redacted] for her patience, sacrifice, deep love and understanding, and my two lovely daughters [redacted] and [redacted] Without their emotional support, I couldn't have finished my study.

I wish to express my many thanks to all the OSE Lab members: Deokjai Choi, Taesang Choi, Ohyoung Kang and Hai Feng Weng for their help, good suggestions and encouragements to finish my study successfully.

Also, I cannot forget the cheers of all the members of the Korean Catholic Community of the Greater Kansas City. Especially, I wish to say thanks to Father Daniel Schneider who supported me spiritually during my study. I wish he is always healthy.

I cannot forget all the emotional and financial support of my brothers and sisters, relatives and friends for my successful study in the States.

I appreciate Ms. Elizabeth S. Smock, Thesis/Dissertation Format Advisor of the School of Graduate Studies, University of Missouri-Kansas City for her excellent advice on a very painful dissertation formatting work.

Finally, I would like to dedicate this dissertation to my parents who sacrificed their whole lives for their eight sons and daughters during very hard times and passed away, and my mother-in-law in Kwangju who encouraged me a lot when I was depressed.

CHAPTER 1

INTRODUCTION

1.1 Service Order Handling Problem

A fast and accurate ordering process will be crucial for service providers in an increasingly competitive telecommunications market. The fragmentation of the market with more actors involved in the provision of services also emphasizes the need for efficient order handling and tracking across company borders. The current ordering process in most service provider organizations was developed for a totally different environment:

- It was not customer friendly because most service providers operated in a regulated, non-competitive environment.
- Customers were prepared to wait for weeks and even months for a new service.
- There were only a few services to order and customization was almost unknown.
- Most of the services were produced in-house, minimizing the need for external interactions.
- Service providers had lots of personnel that could handle activities manually.
- Technology was not mature enough to deal with the complex tasks of the ordering process.

As a consequence of this the current solutions for the ordering process cannot be used to meet the business needs of service providers. The ordering process has a lot in common with the environment it was developed for:

- It is too slow. Information does not travel fast enough in the organization to meet customer requirements.
- It contains too many steps. Fragmentation has gone too far in the ordering process leading to many hand-overs between specialized departments.

- It requires too much re-keying of data. Information is moved manually between support systems which means time consuming typing of data.

- It is difficult to track the order to know what the status is. Order tracking is important to be able to answer customer questions and to ensure that the delivery of the order is following the plans.

- It is too inflexible. The process is adapted to the type of service being ordered. New services usually require organizational as well as support system changes.

- The external interfaces are inefficient. Phone, fax and mail constitute the point of contact between service providers and between service providers and customers.

- The accuracy is low. Many errors are made which lead to costly changes and customer complaints.

- There is a risk for misunderstanding. The terminology is not consistent between different service providers, leading to many mistakes.

These problems with the current ordering process are especially serious when several service providers are involved in providing an end user service.

1.2 Service Order Handling Requirements

The business problems described in Section 1.1 have emerged from the fact that the traditional ordering process has become obsolete when applied to the new market environment. When developing new solutions for this process it is not enough to address the current problems. It is also necessary to look into the current trends and try to predict the way the market situation will evolve. The following enumerates a couple of extra requirements that are important to consider:

- In a competitive environment, service orders generally come out of a discussion, comparing different alternatives, with the customer.

- A pre-order process is needed that can handle several service design proposals in a short time, with low cost, even if several service/network providers are involved in the proposals.

- The recommendation must be applicable in a real multi-service provider market. Having a situation with several dozens of providers involved in an offer will not be unusual. Automatization and real time capabilities are paramount when the number of parties involved increases.

- Many different types of service provider roles must be supported. New roles will be created in the ordering process, addressing other niches than today (for example service brokers, companies doing resale of capacity, etc.). Roles may also change over time and between different markets.

- The entire ordering process must be taken into account when describing process and information models, otherwise it will not be possible to get overall efficiency. To address the service provider to service provider interface in isolation will only lead to sub-optimal solutions.

- During the ordering process, the best efficiency is obtained when the different parties of customer and service provider are involved and can address information to the right peer party. Therefore, to ensure the best quality of service, the customer and service provider parties exchanging the information have to identify and to recognize each other. Assuming the customer doesn't have to make any changes to its legacy ordering systems to address the order to the service provider, the ordering information model must include an identification for the principle parties (or external interfaces to the "human" organization tree) involved in the ordering process. This information can be represented as an attribute of a "customer" object subclass.

1.3 Order Handling Process Model

The overall goal of the ordering process model is to meet customers need for new services as efficient as possible. This means that a whole range of different type of cases from customers wanting a simple quote on a single service to customers going though a complete design and negotiation phase of a complex offering. The ordering process can stop with a proposal to be considered by the customer or go the whole way to firm orders and implementation of the offer. This puts some extra demands on the process model which must be reflected in the model as a high degree of flexibility. The generic ordering process model [44] is given in the Figure 1 below.

It is important to notice that the process model only focuses on the activities involved in ordering. There are other views of ordering that are important, like the information flow, that is not depicted in this model. There is a need to add more dimensions to this model to show a more complete picture of the ordering area.

The meaning of the signs and symbols in this figure can shortly be described like this:

Boxes: A process, i.e., a set of activities with a common goal, a well defined start and end point.

Arrows: The hand over from one process to another, i.e., the trigger for another process to start. They are not showing the information flow.

Broken lines: Interfaces between the main actors involved in ordering are shown as broken lines in the figure. Besides these main interfaces there may be other interfaces depending on the actual market situation.

Each of the processes in ordering are described in more detail in the sections below.

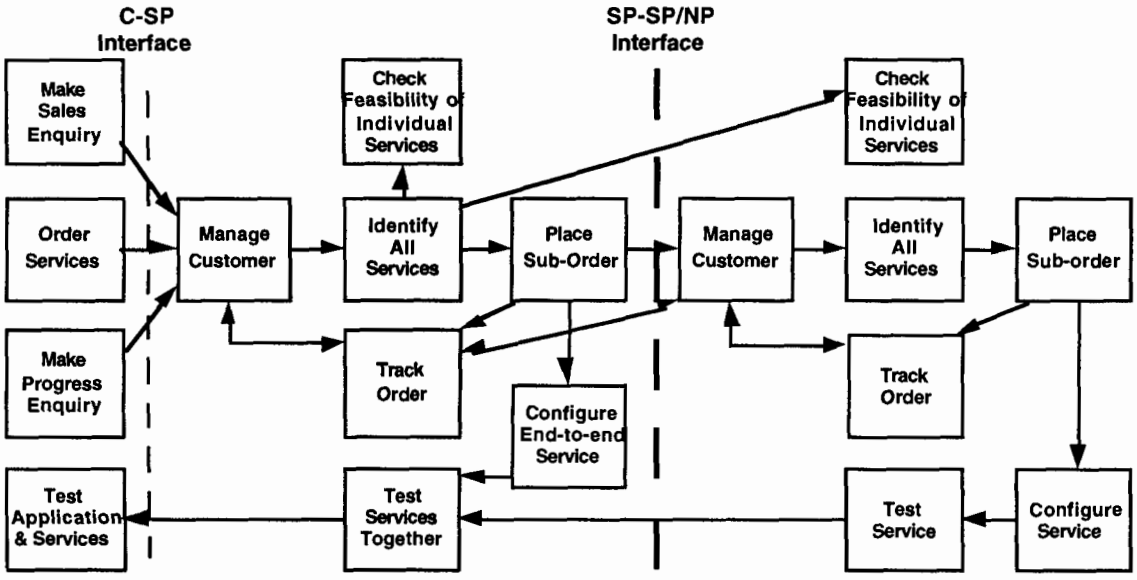

Figure 1 Generic Ordering Process Model [44]

1.3.1 The Customer Processes

The ordering process is triggered by processes within the customer organization. Due to the wide range of customers that will use the ordering process, the type of customer processes will vary. A private customer will not have any fixed processes at all, while a large corporation will have their own internal processes to deal with ordering. The four basic customer processes that form the interfaces to the ordering process are the followings:

- Make Sales Enquiry
- Order Services
- Make Progress Enquiry

- Test Application & Services.

Make Sales Enquiry is the process where the customer is looking for proposals for who to fulfil their communication needs. The result is a proposal that the service provider saves to make it possible for the customer to come back later with a firm order.

Order Services is the customer process where firm orders are generated. It usually refers back to an offer produced earlier for the Make Sales Enquiry process.

The Make Progress Enquiry process is used by customers that want to know what is happening with an enquiry or an order that has been submitted to a service provider. Progress enquiries can be made during the entire ordering process from the production of offers to the implementation of orders.

The Test Application and Services process is the receiving process in the customers' organization. When the offer has been implemented it has to go through acceptance testing at the customers site. In the case of simple services like plain old telephony this process is reduced to a minimum.

1.3.2 Manage Customer

The Manage Customer process handles the management of the customer and act as an interface to processes like ordering. This process is the same for both the main service providers and sub-contracted service/network providers. The three basic subprocesses of the Manage Customer process are the followings:

- Obtain Customer Information
- Negotiate Contract
- Inform the Customer about Changes.

1.3.2.1 Obtain Customer Information

All necessary customer information must be gathered to be able to continue with the preparation of the offer to the customer. How much information that is gathered depends on

the situation. If the customer is just asking for a quote or asking about the progress of an order, less information is needed than if a complete offer is ordered.

1.3.2.2 Negotiate Contract

The exact details of the customer contract is negotiated when the offer is agreed. This process starts when a firm order is placed. This process can continue in parallel with the actual implementation of the order until all necessary information in the contract is in place. This process may comprise an initial credit check to make sure that the customer is worth doing business with.

1.3.2.3 Inform the Customer about Changes

Any changes to what has been promised has to be reported back to the customer. This process is triggered by the order tracking process and means sending information to the customer or taking up a discussion about further changes or consequences.

1.3.3 Identify All Services

The Identify All Services process is the main part of the ordering process where the offer is put together, using internal and external services, to meet the customer's needs. This process is the same for both the main service providers and sub-contracted service/network providers. The four basic subprocesses of the Identify All Services process are the followings:

- Identify Business Problem
- Design Solution
- Ask for/Negotiate Options
- Select Services.

1.3.3.1 Identify Business Problem

Customer requirements and needs are collected and analyzed to get a picture of how the customer is going to use the services. This information is mainly expressed in terms used by

the customer and not in technical telecommunications terms. The result of this process is a list of business, communications and information management problems that the final offer has to meet as close as possible.

1.3.3.2 Design Solution

In this process, the customer needs are transformed into a general solution to meet the customer needs. The solution is expressed as a mix of general services to meet the requirements. These services then have to be instantiated to form a real offer.

1.3.3.3 Ask for/Negotiate Options

In this process the service provider makes a search to gather all possible options that exist to meet the general services that have been selected in the design process. This involves internal as well as external services. The offer to the customer is completed with price estimates and delivery details. In simple cases the instantiation can be made just by searching standard services where all information is already available in information bases. In more complicated cases the "Check Feasibility of Individual Services" process is triggered. This feasibility check is performed either externally in other service providers or internally checking the existence of in-house services.

1.3.3.4 Select Services

In this process the various options of internal and external services are compared to the requirements of the solution being designed. The best services are selected and a description of the offer is stored for future use. This process might start an iteration back to the Ask for/Negotiate Options process if the customer is not content or the requirements are not met.

1.3.4 Check Feasibility of Individual Services

This process takes a request for proposal and check whether services can fit the requirements. If they do, a description of these services is sent back to the process that issued

the request. The reason for having this process is that enquiries about services and the production of different proposals to the customer must be handled as efficiently as possible. Therefore a separate process is formed to handle it. The goal is to produce proposals that comprise enough information to meet the customers' needs. In the case of simple services this process is not needed. In that case the required information should be available directly in the "Ask for/negotiate Options" process.

In some cases the proposal will be stored to be used in the future when the customer comes back with a firm order based on the proposal. There could also be a need for pre-booking of resources and services to be able to respond quickly to firm orders.

1.3.5 Place Sub-order

The Place Sub-order process deals with the tasks that have to be done to start implementing the offer when the customer has decided to make a firm order. The three basic subprocesses of the Identify All Services process are the followings:

- Perform Credit Checks
- Develop Order Plan
- Order Internal/External Services.

1.3.5.1 Perform Credit Checks

The customers ability to pay for the services being ordered is checked. This is a detailed credit check that is made when the entire order (including e.g., prices, credits, etc.) is known. An initial credit check could be made as a part of the contract negotiation in the "Manage Customer" process.

1.3.5.2 Develop Order Plan

When the customer has chosen to accept the offer from the service provider an order plan is developed to guide the realization, testing and delivery of the offer.

1.3.5.3 Order Internal/External Services

The internal and external services selected in the "Identify All Services" process are ordered.

1.3.6 Track Order

A central part of the order process is the Track Order process where the progress is monitored to be able to manage the order process efficiently and to be able to give customers the feedback that they want. The Track Order process can be triggered by any of the three processes Order Services, Make Progress Enquiry and Place Sub-order. Different parts of the process are started depending on which process that triggered the action. The two basic subprocesses of the Track Order process are the followings:

- Track Progress
- Provide Customer Feedback on Progress.

1.3.6.1 Track Progress

All services being ordered, both internally and externally should be tracked to be able to control the implementation of the offer to the customer. Tracking can be both regular, getting standard status reports from those delivering the services, and spontaneous triggered by a service provider. The goal is to be able to handle all queries from the customers about the status and progress of the implementation of the order.

In the case of more complicated offers to the customer that cannot be handled in real time, it is also necessary to track the production of an offer. Information is gathered to be used to meet customer queries about the status and progress. Different solutions are possible from making routine collections of information, to only collecting the relevant information when the customer asks.

1.3.6.2 Provide Customer Feedback on Progress

This process handles customer queries on ordering process progress. Information from the other tracking processes is used. Sometimes this information has to be completed with extra information collected internally and externally from the sub-contracted service/network providers. Any deviations from the plan is reported back to the customer. When the order is fulfilled, this process will see that a final confirmation is sent to the customer.

1.3.7 Configure Services

When a firm order has been placed the configuration of services has to take place according to the order plan. It is a complex process in itself that is essential to look at to reach overall efficiency of ordering.

1.3.8 Configure End-to-end Services

The main service provider will configure the total end-to-end service with external and internal services. This process is similar to the Configure Services process.

1.3.9 Test Services

Service are tested before they are delivered to the service provider that has ordered them.

1.3.10 Test Services Together

The main service provider tests services together to ensure that they can interact correctly.

1.4 Order Handling Interfaces

Interactions between the main service provider and subcontracted providers are related to different processes in ordering. Three main interfaces need to be defined in order to achieve efficiency throughout the ordering process. These interfaces are:

- Ask for/Negotiate Options -> Check Feasibility of Individual Services: This interface deals with the interactions between a main service provider and a subcontracted service provider involved in designing the solution to be offered to the customer. This interface has to handle large transactions of information describing service requirements and possible solutions to meet these requirements. It also has to handle haggling information to be able to agree on the final price for the solution. This interface only deals with the external relations to other service providers, not the internal one between processes within the same organization.
- Place Sub-Order -> Manage Customer: An order, cancellation or change request is issued to the subcontracted service provider's manage customer process. Any changes to the order also have to be reported over this interface. Reports on changes may require quite large amounts of information to be sent over this interface. But, if the changes requires complete redesign of the solution it has to be passed back to the "Ask for/Negotiate Options" process again.
- Track Order -> Manage Customer: The delivery of all the subcontracted services has to be monitored by the main service provider. This interface can be built to send status reports with a certain frequency or to report exceptions if anything happens.

The contents sent over these interfaces will look different depending on what kind of providers that are involved. The type of information passed over the interface depends on how much of the design process that is handled by the main contractor and the subcontractor. If the main contractor has the role of a broker, it will mean that the "Design Solution" process is almost empty and that the "Ask for/Negotiate Options" deals with customer requirements rather than technical descriptions.

1.5 Scenarios

The three interfaces defined above will handle all interactions needed to create a total offer to an end user with several service providers involved. The message sequences that have to take place over these interfaces give a framework for how the information model will be used to support these interactions.

1.5.1 A Scenario for Ask for/Negotiate Options -> Check Feasibility of Individual Services

This interface handles the interactions needed to put together an offer to a customer based on various services from different service providers. Several interactions are needed in the case of a more complex offer. The following shows one possible scenario which can be occurred in this interface. Here, the SP1 and SP2 represents a main service provider and a subcontracted service provider respectively.

1. SP1 -> SP2: A service request, including a description of the requirements of the service, i.e., a general service description is sent.
2. SP1 <- SP2: A proposal to meet the requirements (if possible) is sent. This message contains a service or package description and tariffs.
3. SP1 -> SP2: A request for additional information, changes to the proposal and haggling information is sent.
4. SP1 <-> SP2: Steps 2 - 3 are repeated until an agreement is made.
5. SP1 <- SP2: A description of the final offer is sent. This message contains a service or package description, tariffs and a point of contact for the subcontracted service provider
6. SP1 -> SP2: An acknowledgement is sent.

1.5.2 A Scenario for Place Order -> Manage Customer

This is the formal ordering of a service that is defined by the negotiation process. The receiving service provider may come back with change requests for things that cannot be met. The following shows one possible scenario which can be occurred in this interface. Here, the SP1 and SP2 represents a main service provider and a subcontracted service provider respectively.

1. SP1 -> SP2: An order of a service that either has been negotiated in the "Ask for/Negotiate Options" Process or is contained in the "Contractual Agreements" information base. This message contains customer identifier, an order, a contract, a SLA, point of contacts, customer account identifier and a test plan.
2. SP1 <- SP2: Exceptions from the order. This includes descriptions of things that cannot be fulfilled. This message comprises references to the order information that it concerns.
3. SP1 -> SP2: A new order containing the relevant change requests from the subcontractor is sent.
4. SP1 <- SP2: An acknowledgement is sent.
5. SP1 -> SP2: Notifications of changes and cancellations that may arise during the implementation of the solution are sent.
6. SP1 <- SP2: Notification of final delivery is sent.

1.5.3 A Scenario for Track Order -> Manage Customer

The main service provider always needs to have information of the order status. This information can be sent with a certain frequency or at the service providers request. It can also be a spontaneous report sent from the subcontracted service provider if anything important happens. Here, the SP1 and SP2 represents a main service provider and a subcontracted service provider respectively.

1. SP1 -> SP2: Request for order status is sent. This is launched when the main service provider needs to know the status and progress of an offer being put together.

2. SP1 <- SP2: Status reports on demand or at a defined frequency during the delivery of the service are sent.

3. SP1 <- SP2: Reports on exceptions that might arise during the delivery that can influence the overall offer to the customer are sent.

CHAPTER 2

OVERVIEW OF SP-TO-SP ORDER HANDLING INFORMATION MODEL

2.1 Overview

The following information object types are relevant to the ordering handling interfaces:

- rFP (Request for Proposal),
- pR (Proposal),
- oP (Option),
- oR (Order) and
- sR (Service).

The first three object types are used in the pre-ordering phase, and the last two object types are used in the ordering phase.

2.1.1 Information Object Types in the Pre-ordering Phase

The pre-ordering phase is made up of order handling activities which may occur before an order is placed. Let us assume that an MSP, say A, tries to gather all possible options that exist to meet some service requirements. First, A submits a request for proposal to an SSP B. The following information is provided by A in its request:

- additionalInfo: this provides additional information (e.g., vendor qualification requirement, proposal evaluation procedure) on the project which this request for proposal is about.
- dueDate: this indicates the last day for B to respond to the request for proposal.
- expectedResponsibility: this specifies B's responsibilities such as buying insurance, getting permission and license for the project, etc.
- generalInfo: this provides the principal features of the project so that B can quickly determine which, if any, of their services meet the requirements. This may include the

type of service being requested, the required completion date, the schedule of major events, etc.

- projectDescription: this provides a narrative description of the organization (i.e., A) requesting the service and the environment in which it will be used. This may include the objectives of the project, the primary lines of business of A, the addresses of principal locations, the description of existing services and the reasons why they are to be replaced.
- projectRequirements: this explains what the project must accomplish. A should state clearly which requirements are mandatory and which are optional. B is supposed to check feasibility of services fitting these requirements.
- purchaseTermsAndConditions: this specifies warranty, payment terms, criteria of system acceptance, performance bonding, etc.
- requestor: this provides information on the person issuing the request for proposal.
- rFPName: this is the name assigned to the request for proposal by the MSP.

After B receives the request for proposal, it will create an rFP object. In addition to the information supplied by A, the following information can be found in an rFP object:

- cancelRequestedIndicator: this indicates whether A has requested for the cancellation of the request for proposal (this is initialized with the FALSE value).
- enteredDate: this indicates the date on which the rFP object was created.
- rFPID: this is the identifier assigned to the rFP object at the creation time.
- rFPState: this gives the current state of the rFP object. The rFP state model will be discussed later in this section.
- rFPStatus: this describes the progress which B has made in processing the request for proposal.
- rFPStatusTime: this indicates the time at which the status was last modified or validated.

The above information is also returned to A.

After reviewing the request for proposal, let us assume that B is in a position to respond with a proposal consisting of one or more options. A pR object along with one or more oP objects will be created by B. The pR object contains the following information:

- expiryTime: this indicates the time at which the terms in the proposal will expire.
- oPList: this is a list of names of the oP objects for the options in the proposal.
- proposer: this provides information on the person offering the proposal.
- pRID: this is the identifier assigned to the pR object at the creation time.
- pRState: this gives the current state of the pR object. The pR state model will be discussed later in this section.
- rFPID: this identifies the rFP object for which this pR object is created.

Each oP object contains the following information:

- hagglingInfo: this indicates the items (e.g., price) which are subject to negotiation between A and B.
- oPID: this is the identifier assigned to the oP object at the creation time.
- oPState: this identifies the current state of the oP object. The oP state model will be discussed later in this section.
- pRID: this identifies the pR object to which this oP object is linked.
- serviceItemList: this is a list of services proposed in the option to meet the project requirements in the request for proposal.

After A has received the proposal, it will examine the various options. For any option on which A wants to negotiate with B, A can counteroffer by requesting one or more change(s) in the serviceItemList and/or hagglingInfo attribute(s). After B has received the counteroffer, it too can counteroffer. The negotiation will last until either both sides have agreed, B has determined to withdraw the option (e.g., when negotiation comes to a standstill), or A has explicitly indicated no interest to continue the negotiation. If A is not interested in an option, it

does not need to respond. When B finds out that there is no response to an option in due time (which may be determined by the business agreement), it should assume that A is no longer interested in the option.

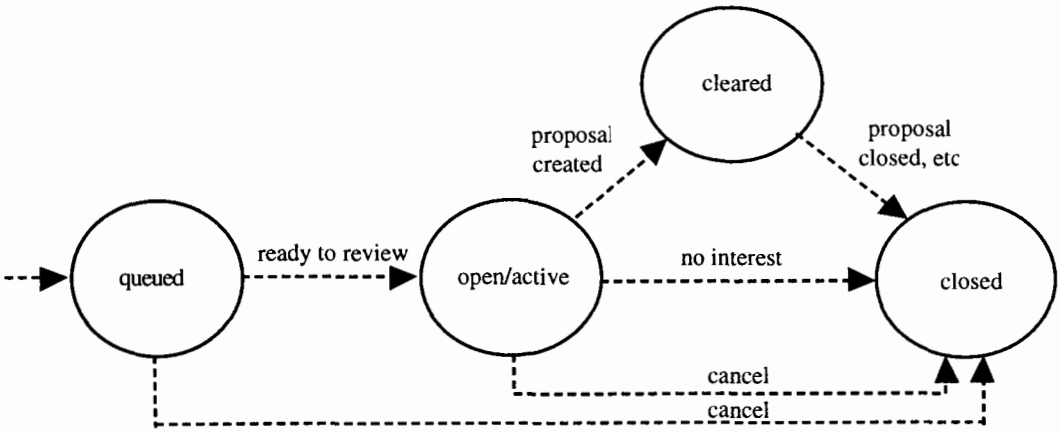

Figure 2 rFP State Model

Next, we examine the state model of each information object type. We begin with the rFP information object type (Figure 2). When an rFP object is created, it is in the QUEUED state. When B starts to review the request for proposal, its state is changed to OPEN/ACTIVE. After reviewing, B may decide to prepare a proposal or give up. If B decides to prepare a proposal and create a pR object, the state of the rFP object is changed to the CLEARED state. As long as the rFP object is in the CLEARED state, the rFP object can be referenced by a pR object. After the pR object is closed, there is no reason to maintain the rFP object, hence the state of the rFP object is changed to the CLOSED state. When the state of the rFP object is either QUEUED or OPEN/ACTIVE, it can be also changed to the CLOSED state if the MSP requests to cancel the request for proposal.

Next, we examine the oP state model (Figure 3). When an oP object is created, it is in the NEGOTIATING state, i.e., the option is subject to negotiation. During negotiation, the oP

object remains in the NEGOTIATING state as long as both sides have not agreed to the terms. When both sides have agreed with each other, the state is changed to NEGOTIATION_OVER. When the expiryDate of the proposal expires, the state is changed to CLOSED. When the oP object is in the NEGOTIATING state, its state can be changed to CLOSED for other reasons, e.g.,

- the SSP wants to cancel the option (e.g., it finds out during the negotiation that the option does not fit the needs of the MSP) or
- the MSP does not respond to the option in a time period which may be determined by the business agreement between the MSP and the SSP.

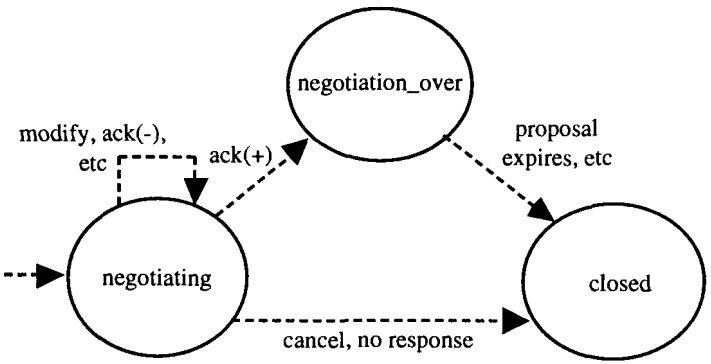

Figure 3 oP State Model

The pR state model is defined in terms of the state model of the oP information object type. For example, if a pR object has specified four options, then the current state of the pR object is given by the current states of the four oP objects linked to the pR object.

2.1.2 Information Object Types in the Ordering Phase

The ordering phase is made up of order handling activities which may occur when an order is placed. Let us assume that A is ready to place an order with B. In the order request, A may provide the following information:

- billingInfo: this provides billing information (e.g., billing account number, billing address, billing contact person) on the order.
- chargeCode: this identifies A's charge code for the order.
- contractID: this identifies the contract (between A and B) specifying the ordering terms and conditions.
- initialORID: this identifies the order of which this order forms a part.
- oPID: this identifies an oP object which triggers the issuance of this order.
- oRDescription: this gives a summary of the order.
- oRName: this is the name assigned to the order by the MSP.
- oRPriority: this indicates the priority of the order.
- oRRequestor: this provides information on the person placing the order.
- oRType: this indicates whether this is an initial order or a subcontracted order (in our case, this is a subcontracted order).
- projectCode: this is used to associate related orders.
- requestedDate: this indicates the date on which order fulfillment is requested by A.
- serviceItemList: this identifies the services to be ordered.

After B has received the order request, it will create an oR object and an sR object for each service item in the serviceItemList. The oR object will be initialized with the information supplied by A. In addition, it will be initialized with the following information supplied by B:

- additionalORStatusInfo: this provides progress information on the processing of the order when there is no recent status change.
- cancelRequestedIndicator: this indicates whether A has requested for the cancellation of the order.
- chargeDate: this indicates the date on which charges will be billed.
- facilityTestDate: this indicates the date on which cooperative testing (involving A and B) of the services is scheduled.

acceptanceDate: this indicates the date on which fulfillment was accepted by A (this is initialized with the NULL value).

committedDate: this indicates the date to which the order fulfillment is committed by B.

fulfillmentDate: this indicates the date on which order fulfillment occurred (this is initialized with the NULL value).

oRClearancePerson: this provides information on the person who is responsible for either requesting for the cancellation of the order or verifying the fulfillment of the order.

oRCloseOutVerification: this indicates whether A has verified order fulfillment, denied order fulfillment or taken no action.

oRDialog: this enables interaction to take place between A and B at each stage of the ordering process.

oREngineeringPerson: this provides information on the person to be contacted with technical/design queries regarding the service implementation.

oRID: this is the identifier assigned to the oR object at the creation time.

oRInstallationPerson: this provides information on the person to be contacted with queries/information regarding the installation schedule.

oRReceivedDate: this indicates the date on which the order was received.

oRRepresentative: this provides information on the person receiving the order.

oRState: this gives the current state of the oR object. The oR state information object type will be discussed later in this section.

oRStatus: this describes the progress which B has made in processing the order request.

oRStatusTime: this gives the time at which the oRStatus was last validated or changed.

oRStatusWindow: this specifies the frequency for B to generate a progress report on the order.

- readyForServiceDate: this indicates the date on which all the requested services were tested to be operational by B (this is initialized with the NULL value).
- sRList: this gives the names of the sR objects created for the order.

Given the seven different date-related oR attributes, it is worthwhile to understand how these dates are related to each other. The following shows the event sequence on these dates:

- oRReceivedDate: first B receives an order from A which specifies a requestedDate,
- committedDate: then B responds with a committedDate,
- readyForServiceDate: then B indicates to A that the order is ready for service,
- facilityTestDate: then A and B collaborate to test the order,
- fulfillmentDate: then A indicates that the order is fulfilled if the collaborative test passes,
- acceptanceDate: finally A indicates the acceptance of the order.

Note that the dates in the oRReceivedDate, readyForServiceDate, fulfillmentDate and acceptanceDate attributes cannot be modified.

An sR object is created for every service item in the serviceItemList attribute of the oR object. At the creation time, B will initialize an sR object with the following information:

- additionalSRStatusInfo: this is used to provide additional information on the service ordering progress when there is no recent status change.
- cancelRequestedIndicator: this indicates whether A has requested for the cancellation of the service order.
- endUserInfoList: this provides information on the end users (e.g., their names) of the service.
- oRID: this identifies the oR object to which this sR object is linked.
- requestType: this indicates the request type which can be either newService, disconnect, move or change.
- serviceType: this specifies the service being ordered (e.g., PSTN, X.25),

- sRDialog: this enables interaction to take place between A and B at each stage of the service ordering process.
- sRID: this is the identifier assigned to the sR object at the creation time.
- sRLocationInfoList: this provides information on the locations to be served.
- sRPriority: this indicates the priority of the sR object.
- sRSpecificInfoList: this provides information specific to the service order.
- sRState: this gives the current state of the sR object. The sR state model will be discussed shortly.
- sRStatus: this describes the progress which B has made in processing the service order.
- sRStatusTime: this gives the time at which the sR status was last validated or changed.
- sRStatusWindow: this specifies the frequency for B to generate a progress report on the service order.

Next, we examine the sR state model of the sR information object type (Figure 4). When an sR object is created, it may be initially in the QUEUED state, meaning that the service order has not yet been processed. As soon as the SSP has performed some work on the service order, the state of the sR object is changed to OPEN/ACTIVE. When the SSP believes that a facility test on the order is ready, the state of the sR object is changed to CLEARED. After the MSP has verified fulfillment of the service order, the state of the sR object is changed to CLOSED. If the fulfillment is denied, the state is changed to OPEN/ACTIVE. When the state of the sR object is either QUEUED or OPEN/ACTIVE, it can be changed to CLOSED if MSP requests to cancel the service order.

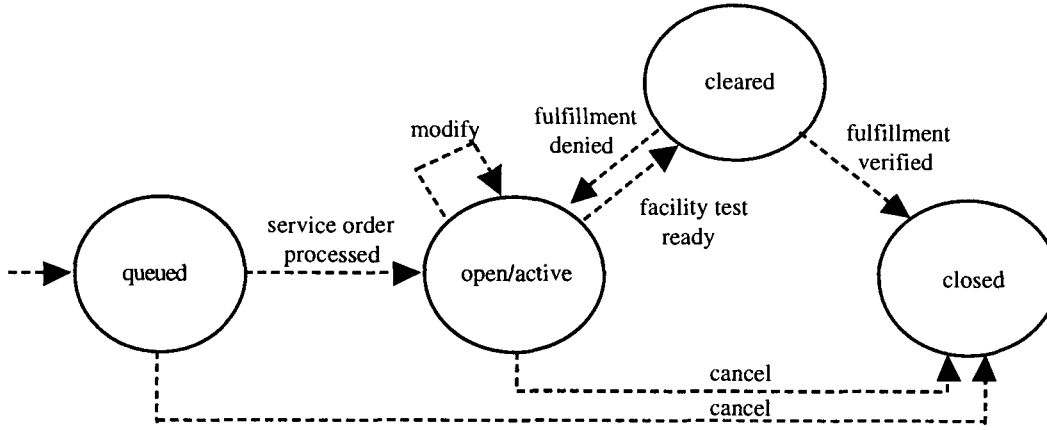

Figure 4 sR State Model

2.2 TINA-C Information Modelling

This section gives an overview of the TINA-C information modelling technique. In the following sections (Sections 2.3-2.7), we sketch a TINA-C specification of each order handling information object type.

The information specification of a distributed application provides the necessary knowledge to interact appropriately within the application. It describes such knowledge in the form of information object types and relationships among them.

TINA-C introduces an information specification language called quasi-GDMO+GRM (so called because of its resemblance to the GDMO [21] and GRM notations [22]). In quasi-GDMO+GRM, an information object type is described in terms of states and actions. The state of an information object type is characterized by attributes. The actions of an information object type are the means to modify the state of an object. The state model does not associate events with an object type, although events are the causes to have actions performed on an object. Instead, events are associated with a computational interface, since the same information object type may be used for different computational interfaces. By associating

events with a computational interface and then mapping events to actions of an information object type, one can give an operational semantics specification of a computational interface.

Conceptually, quasi-GDMO+GRM is different from GDMO [21] because it is not oriented for CMISE [20]. The attribute operations in a GDMO definition are associated with CMISE operations. On the other hand, the attribute operations in a quasi-GDMO+GRM definition are not invocable and should not be associated with operations in any computational interface. Separating the information object design from the computational interface design can be viewed as an advantage of quasi-GDMO+GRM over GDMO, because it encourages re-usability of an information object type in different computational settings. Other subtle differences between quasi-GDMO+GRM and GDMO (such as absence of name binding rules and conditional packages) are discussed in [53].

In quasi-GDMO+GRM, the object type template is used to specify an information object type. It consists of clauses on behavior, attributes, actions and notifications.

The behavior clause specifies the invariant properties of the object type, the semantics of object creation/deletion operations, and the semantics of actions which can be performed on the object. The semantics of creation/deletion operations and actions are specified by means of pre-conditions and post-conditions.

The attribute clause specifies attributes and attribute-operations. Collectively, the attributes define the states of the object type. Attribute-operations (which can be GET, REPLACE, GET-REPLACE, ADD, REMOVE or ADD-REMOVE [21]) are actions on a single attribute.

The actions clause specifies other actions which can cause a state change. Unlike an attribute-operation, an action may cause a side-effect on more than one attribute. The action clause is not used in the specification in this document.

The notification clause specifies notifications. A notification is a special case of an action. Every notification has a triggering-condition which when true, makes the object

"fire" a notification. An object creation/deletion or attribute value change normally fires a notification.

2.3 rFP Information Object Type

2.3.1 rFP Attributes

The following gives an informal description of each rFP attribute. The associated attribute-operations are shown after each attribute.

- additionalInfo (GET)

  This provides additional information (such as vendor qualification, proposal evaluation procedure and required documents) on the project which this request for proposal is about.

- cancelRequestedIndicator (GET-REPLACE)

  This indicates whether the MSP has requested to cancel the request for proposal. This attribute is initialized with the FALSE value.

- dueDate (GET-REPLACE)

  This indicates the last day for the SSP to respond with a proposal to the request for proposal.

- enteredDate (GET)

  This indicates the date on which the rFP object was created.

- expectedResponsibility (GET)

  This indicates the subcontracted service provider's responsibility such as buying necessary insurance, obtaining permits and licenses for the project, etc.

- generalInfo (GET)

This indicates the principal features of the project so that the SSP can quickly determine which, if any, of their services meet the requirements. This may include type of service being requested, required completion date, schedule of major events, etc.

projectDescription (GET)

This gives a narrative description of the organization (of the MSP) requesting the service and the environment in which it will be used. This may include the objectives of the project, the primary lines of business, the addresses of principal location, the description of existing systems and reasons why they are to be replaced.

projectRequirements (GET)

This explains what the project must accomplish. The MSP should state clearly which requirements are mandatory and which are optional. The SSP is supposed to check feasibility of services fitting these requirements.

purchaseTermsAndConditions (GET)

This includes warranty, payment terms, criteria for system acceptance, performance bonding etc.

requestor (GET)

This provides information on the person issuing the request for proposal.

rFPID (GET)

This is the identifier assigned to the rFP object at the creation time.

rFPName (GET)

This is the name assigned to the request for proposal by the MSP.

rFPState (GET-REPLACE)

This gives the current state of the rFP object (Figure 2).

rFPStatus (GET-REPLACE)

This gives the current status of the rFP object. Examples of status values are:

- pendingForReview,

- reviewingInProgress,
- reviewingComplete,
- proposalPreparing,
- proposalPreparationDone,
- proposalDispatched,
- closeOutNoInterest,
- closeOutProposalCancelled,
- closeOutRFPCancelled,
- closeOutNoResponseToProposal and
- closeOutProposalExpired.

- rFPStatusTime (GET-REPLACE)

  This gives the time at which the rFPStatus was last validated or changed.

2.3.2 rFP Notifications

An rFP notification is treated as a special case of an rFP action. It has a triggering-condition that when true, makes an rFP object emit a notification. Here are the rFP notifications:

- rFP-avc-notification():(rFP-avc-notification-list:RFPAttrList)

  The triggering-condition is that there is an attribute value change of an rFP object. The rFP-avc-notification-list output parameter is used to specify the rFP attribute(s) which were modified.

- rFP-create-notification():(rFP-create-notification-list:RFPAttrList)

  The triggering-condition is that an rFP object has been created using the rFP-create action. The rFP attributes in the rFP-create-notification-list output parameter are those which are supplied at the creation time.

- rFP-delete-notification():(rFPID:rFPID)

The triggering-condition is that an rFP object has been deleted using the rFP-delete action. The rFPID output parameter identifies the deleted rFP object.

- rFP-historyEvent-notification():(rFP-historyEvent-notification-list:RFPAttrList)

  The triggering-condition is that the state of the rFP object has been changed to CLOSED. The rFP-historyEvent-notification-list shows the rFP attributes which may be stored in an rFP history log record.

2.4 pR Information Object Type

2.4.1 pR Attributes

The following gives an informal description of each pR attribute. The associated attribute-operations are shown after each attribute.

- expiryTime (GET-REPLACE)

  This indicates the time at which the terms in the proposal will expire.

- oPList (GET)

  This is a list of names of the oP objects for the options in the proposal.

- pRID (GET)

  This is the identifier assigned to the pR object at the creation time.

- pRState (GET-REPLACE)

  This gives the current state of the pR object. The state of the pR object is defined in terms of the states of the oP objects (Figure 2.2) linked to the proposal.

- proposer (GET-REPLACE)

  This provides information on the person offering the proposal.

- rFPID (GET)

  This identifies the rFP object for which this pR object is created.

2.4.2 pR Notifications

A pR notification is treated as a special case of a pR action. It has a triggering-condition that when true, makes a pR object emit a notification. Here are the pR notifications:

- pR-avc-notification():(pR-avc-notification-list:PRAttrList)

  The triggering-condition is that there is an attribute value change of a pR object. The pR-avc-notification-list output parameter is used to specify the pR attribute(s) which were modified.

- pR-create-notification():(pR-create-notification-list:PRAttrList)

  The triggering-condition is that a pR object has been created using the pR-create action. The pR attributes in the pR-create-notification-list output parameter are those which are supplied at the creation time.

- pR-delete-notification():(pRID:PRID)

  The triggering-condition is that a pR object has been deleted using the pR-delete action. The pRID output parameter identifies the deleted pR object.

- pR-historyEvent-notification():(pR-historyEvent-notification-list:PRAttrList)

  The triggering-condition is that the state of the pR object has been changed to CLOSED. The pR-historyEvent-notification-list shows the pR attributes which may be stored in a pR history log record.

2.5 oP Information Object Type

2.5.1 oP Attributes

The following gives an informal description of each oP attribute. The associated attribute-operations are shown after each attribute.

- hagglingInfo (GET-REPLACE)

This indicates the items (e.g., price) which are subject to negotiation between the MSP and the SSP.

- oPID (GET)

  This is the identifier assigned to the oP object at the creation time.

- oPState (GET-REPLACE)

  This gives the current state of the oP object (Figure 2.3).

- pRID (GET)

  This identifies the pR object to which this oP object is linked.

- serviceItemList (GET-REPLACE, ADD-REMOVE)

  This is a list of service orders proposed in the option to meet the requirements in the request for proposal.

2.5.2 oP Notifications

An oP notification is treated as a special case of an oP action. It has a triggering-condition that when true, makes an oP object emit a notification. Here are the oP notifications:

- oP-avc-notification():(oP-avc-notification-list:OPAttrList)

  The triggering-condition is that there is an attribute value change of an oP object. The oP-avc-notification-list output parameter is used to specify the oP attribute(s) which were modified.

- oP-create-notification():(oP-create-notification-list:OPAttrList)

  The triggering-condition is that an oP object has been created using the oP-create action. The oP attributes in the oP-create-notification-list output parameter are those which are supplied at the creation time.

- oP-delete-notification():(oPID:OPID)

  The triggering-condition is that an oP object has been deleted using the oP-delete action. The oPID output parameter identifies the deleted oP object.

- oP-historyEvent-notification():(oP-historyEvent-notification-list:OPAttrList)

  The triggering-condition is that the state of the oP object has been changed to CLOSED. The oP-historyEvent-notification-list shows the oP attributes which may be stored in an oP history log record.

2.6 oR Information Object Type

2.6.1 oR Attributes

The following gives an informal description of each oR attribute. The associated attribute-operations are shown after each attribute.

- acceptanceDate (GET)

  This indicates the date on which fulfillment was accepted by the MSP (if supplied, this attribute is initialized with the NULL value).

- additionalORStatusInfo (GET-REPLACE)

  This provides progress information on the ordering process when there is no recent status change.

- billingInfo (GET-REPLACE)

  This provides billing information (e.g., billing account number, billing address, billing contact person) on the order.

- cancelRequestedIndicator (GET-REPLACE)

  This indicates whether the MSP has requested to cancel the order (if supplied, this attribute is initialized with the FALSE value).

- chargeDate (GET-REPLACE)

  This indicates the date on which charges will be billed.

- chargeCode (GET-REPLACE)

  This identifies the MSP's charge code (i.e., purchase order number) for the order.

committedDate (GET-REPLACE)

This indicates the date to which order fulfillment is committed by the SSP.

contractID (GET)

This identifies the contract specifying the terms and conditions of ordering.

facilityTestDate (GET-REPLACE)

This indicates the date on which cooperative testing of the services (involving the MSP and the SSP) is scheduled.

fulfillmentDate (GET)

This indicates the date on which the order fulfillment occurred (if supplied, this attribute is initialized with the NULL value).

initialORID (GET)

This identifies the order of which this order forms a part.

oPID (GET)

This identifies the oP object which triggers the issuance of this order.

oRClearancePerson (GET-REPLACE)

This provides information on the person who is responsible for either requesting for the cancellation of the order or verifying the fulfillment of the order.

oRCloseOutVerification (GET-REPLACE)

This indicates whether the MSP has verified order fulfillment, denied order fulfillment or taken no action (if supplied, this attribute is initialized with the NO_ACTION value).

oRDescription (GET)

This gives a summary of the order.

oRDialog (GET-REPLACE)

This enables interaction to take place between the MSP and the SSP at each stage of the ordering process. The "dialog text" is free format text. The contents are replaced by the new "dialog text" as the dialog progresses during the service ordering process.

oREngineeringPerson (GET-REPLACE)

This provides information on the person to be contacted with technical/design queries regarding the service implementation.

oRID (GET)

This is the identifier assigned to the oR object at the creation time.

oRInstallationPerson (GET-REPLACE)

This provides information on the person to be contacted with queries/information regarding the installation schedule.

oRName (GET)

This is a name assigned to the order by the MSP.

oRPriority (GET-REPLACE)

This indicates the priority of the order.

oRReceivedDate (GET)

This indicates the date on which the order was received.

oRRepresentative (GET-REPLACE)

This provides information on the person who received the order.

oRRequestor (GET-REPLACE)

This provides information on the person placing the order.

oRState (GET-REPLACE)

This gives the current state of the oR object. The state of an oR object is determined by the states of the sR objects (Figure 4) linked to it.

oRStatus (GET-REPLACE)

This describes the progress which the SSP has made in processing the order.

oRStatusTime (GET-REPLACE)

This gives the time at which the oRStatus was last validated or changed.

oRStatusWindow (GET-REPLACE)

This specifies the frequency for the SSP to generate a progress report on the order.

oRType (GET)

This indicates whether this is an initial order or a subcontracted order (in our case, this is a subcontracted order).

projectCode (GET)

This is used to associate related orders.

readyForServiceDate (GET)

This indicates the date at which all the requested services were tested (by the SSP) to be operational (if supplied, this attribute is initialized with the NULL value).

requestedDate (GET-REPLACE)

This provides information on the date on which order fulfillment was requested by the MSP.

serviceItemList (GET)

This describes the service orders to be ordered. The MSP may provide the following information to each service order:

- endUserInfoList,
- requestType,
- serviceType,
- sRDialog,
- sRLocationInfoList,
- sRName,
- sRPriority,
- sRSpecificInfoList and
- sRStatusWindow.

For each service order in the serviceItemList, the SSP will create an sR object at the time the oR object is created. Each sR object is initialized with the attributes supplied

by the MSP as well as attributes supplied by the SSP. See Section 2.7.1 for sR attributes.

- sRList (GET)

  This gives the identifiers of the sR objects created for the order.

- sRName (GET)

  This is a name assigned to the service order by the MSP.

2.6.2 oR Notifications

An oR notification is treated as a special case of an oR action. It has a triggering-condition that when true, makes an oR object emit a notification. Here are the oR notifications:

- oR-avc-notification():(oR-avc-notification-list:ORAttrList)

  The triggering-condition is that there is an attribute value change of an oR object. The oR-avc-notification-list output parameter is used to specify the oR attribute(s) which were modified.

- oR-create-notification():(oR-create-notification-list:ORAttrList)

  The triggering-condition is that an oR object has been created using the oR-create action. The oR attributes in the oR-create-notification-list output parameter are those which are supplied at the creation time.

- oR-delete-notification():(oRID:ORID)

  The triggering-condition is that an oR object has been deleted using the oR-delete action. The oRID output parameter identifies the deleted oR object.

- oR-historyEvent-notification():(oR-historyEvent-notification-list:ORAttrList)

  The triggering-condition is that the state of the oR object has been changed to CLOSED. The oR-historyEvent-notification-list shows the oR attributes which may be stored in an oR history log record.

- oR-progress-notification():(oR-progress-notification-list:ORAttrList)

The triggering-condition is that the window size (as defined by the oRStatusWindow attribute) is zero. The oR-progress-notification-list output parameter is used to specify the oR attributes which may appear in a status progress report.

2.7 sR Information Object Type

2.7.1 sR Attributes

The following gives an informal description of each sR attribute. The associated attribute-operations are shown after each attribute.

- additionalSRStatusInfo (GET-REPLACE)

  This is used to provide progress information on the service ordering process when there is no recent status change.

- cancelRequestedIndicator (GET-REPLACE)

  This indicates whether the MSP has requested to cancel the service order.

- endUserInfoList (GET-REPLACE)

  This provides information on the end users to be served.

- oRID (GET)

  This identifies the oR object to which this sR object is linked.

- requestType (GET)

  This indicates the request type which can be either newService, disconnect, move or change.

- serviceType (GET)

  This specifies the type of the service being ordered.

- sRDialog (GET-REPLACE)

  This enables interaction to take place between the MSP and the SSP at each stage of the service ordering process. The "dialog text" is free format text. The contents are replaced by the new "dialog text" as the dialog progresses during the service ordering process.

- sRID (GET)

  This is the identifier assigned to the sR object at the creation time.

sRLocationInfoList (GET)

This provides information on the locations to be served.

sRName (GET)

This is the name assigned to the service order by the MSP.

sRPriority (GET-REPLACE)

This indicates the priority of the sR object.

sRSpecificInfoList (GET-REPLACE)

This provides information specific to the service order.

sRState (GET-REPLACE)

This gives the current state of the sR object.

sRStatus (GET-REPLACE)

This describes the progress which the SSP has made in processing the service order. Examples of status values are:

- installPending,
- installInProgress,
- installSuccessful,
- installFail,
- serviceTestPending,
- serviceTestInProgress,
- serviceTestSuccessful,
- serviceTestFail,
- facilityTestPending,
- facilityTestInProgress,
- facilityTestSuccessful,
- facilityTestFail,
- readyForService and

- closeOut.

- sRStatusTime (GET-REPLACE)

  This gives the time at which the sRStatus was last validated or changed.

- sRStatusWindow (GET-REPLACE)

  This specifies the frequency for the SSP to generate a progress report on the service order.

2.7.2 sR Notifications

An sR notification is treated as a special case of an sR action. It has a triggering-condition that when true, makes an sR object emit a notification. Here are the sR notifications:

- sR-avc-notification():(sR-avc-notification-list:SRAttrList)

  The triggering-condition is that there is an attribute value change of an sR object. The sR-avc-notification-list output parameter is used to specify the sR attribute(s) which were modified.

- sR-create-notification():(sR-create-notification-list:SRAttrList)

  The triggering-condition is that an sR object has been created using the sR-create action. The sR attributes in the sR-create-notification-list output parameter are those which are supplied at the creation time.

- sR-delete-notification():(sRID:SRID)

  The triggering-condition is that an sR object has been deleted using the sR-delete action. The sRID output parameter identifies the deleted sR object.

- sR-historyEvent-notification():(sR-historyEvent-notification-list:SRAttrList)

  The triggering-condition is that the state of the sR object has been changed to CLOSED. The sR-historyEvent-notification-list shows the sR attributes which may be stored in an sR history log record.

- sR-progress-notification():(sR-progress-notification-list:SRAttrList)

The triggering-condition is that the window size (as defined by the sRStatusWindow attribute) is zero. The sR-progress-notification-list output parameter is used to specify the sR attributes which may appear in a status progress report.

CHAPER 3

OVERVIEW OF SP-TO-SP ORDER HANDLING INTERFACES

3.1 Overview

3.1.1 Service Negotiation Interfaces

Used in the pre-ordering phase, the two interfaces here provide capabilities to negotiate on an option (in a proposal) and track a request for proposal.

There are two interfaces: SN_MSP and SN_SSP. The SN_MSP interface is comprised of operations which are invoked by the MSP, and the SN_SSP interface is comprised of notifications which are generated by the SSP either directly via a notification server.

The operations in the SN_MSP interface allow the MSP to:

- enter a request for proposal (rFPCreate),
- view an option or a proposal (oPView, pRView),
- modify an option or a proposal (oPModify, pRModify),
- acknowledge an SSP either positively or negatively (oPAcknowledge) and
- track the status of a request for proposal (rFPStatusTrack).

The operations in the SN_SSP interface allow the SSP to inform an MSP of the following events:

- creation of a proposal (pRCreationNotify).
- modification of an option or a proposal (oPModificationNotify, pRModificationNotify),
- acknowledgement of an MSP either positively or negatively (oPAcknowledgeNotify),
- cancellation of an option or a proposal (oPCancelNotify, pRCancelNotify) and
- notification of a status update of the request for proposal (rFPStatusUpdate).

Let us explain how the operations above are applied during the negotiation process. Initially, B (an SSP) presents an option to A (an MSP). If A agrees to the terms in the option, it can respond by invoking oPAcknowledge with the "AGREED" disposition. If A does not agree to the terms, there are three choices. In the first choice, A invokes oPAcknowledge with the "DISAGREED" disposition. In the second choice, A invokes oPModify to suggest a different set of values for the negotiable parameters (e.g., hagglingInfo or serviceItemList). In the third choice, A does not respond.

Next, we examine how B responds. If A agrees to the terms, B would invoke oPAcknowledgeNotify with the "AGREED" disposition. This invocation would change the state of the oP object to NEGOTIATION_OVER. If A does not agree to the terms and selects the first or third choice, B has two choices - note that B normally waits for a time period (which may be defined in the business agreement) before it determines that A selects the third choice. B can either invoke oPModificationNotify to suggest a different set of values for the negotiable parameters or notify A that the option is cancelled; the latter action is normally taken when B realizes that the negotiation has come to a standstill. If A does not agree to the terms and selects the second choice, B has four choices. B can either agree with A on the proposed modification by invoking oPAcknowledgeNotify with the "AGREED" disposition, or disagree with A by invoking oPAcknowledgeNotify with the "DISAGREED" disposition, or propose a different set of values of the negotiable parameters by invoking oPModificationNotify, or notify A that the option is cancelled. In general, the negotiation continues until either side acknowledges with the "AGREED" disposition. To prevent the negotiation continue forever without success, B can invoke oPCancel to cancel an option (as long as it follows the constraints in the business agreement). If the cancelled option is the only option in the proposal, then cancelling the option has the same effect as cancelling the entire proposal.

The interfaces also provide the capability for the MSP to track the status of a request for proposal. There are three general methods to support tracking. In the first method, A can

invoke rFPStatusTrack to poll the status of the request for proposal. In the second method, B can invoke rFPStatusUpdate whenever there is a change of status of the request for proposal. In the third method, B can send a status progress report on the request for proposal to A at a pre-defined frequency. Our interfaces do not support the third method since there is no requirement for this support.

3.1.2 Service Ordering Interfaces

In an order, there are normally one or more service orders. The interfaces here provide the capabilities to manage an active order or service order during the ordering phase. There

There are two interfaces: SO_MSP and SO_SSP. The SO_MSP interface is comprised of operations which are invoked by the MSP, and the SO_SSP interface is comprised of notifications which are generated by the SSP either directly via a notification server.

The operations in the SO_MSP allow the MSP to:

- place an order (oRCreate),
- modify an order or a service order (oRModify, sRModify),
- view an order or a service order (oRView, sRView),
- cancel an order or a service order (oRCancel, sRCancel) and
- verify the fulfillment of an order or a service order (oRVerify, sRVerify).

The operations in the SO_SSP interface allow the SSP to inform the MSP of the following events:

- modification of an order or a service order (oRModificatioNotify, sRModificationNotify).
- creation of a history log record for an order (oRHistoryNotify)
- deletion of an order (oRDeletionNotify)

3.1.3 Order Tracking Interfaces

Used in the ordering phase, the interfaces here support tracking of an order or a service order in three different ways. First, they allow the MSP to poll the status. Second, they allow the SSP to inform the MSP of a status change. Finally, they allow the SSP to send a status progress report to the MSP at a pre-defined frequency.

There are two interfaces: OT_MSP and OT_SSP. The OT_MSP interface is comprised of operations which are invoked by the MSP, and the OT_SSP interface is comprised of notifications which are generated by the SSP either directly via a notification server.

The operations in the OT_MSP interface allow the MSP to:

- track the status of an order or a service order (oRStatusTrack, sRStatusTrack) and
- set the status window size of an order or a service order (oRWindowSet, sRWindowSet).

The operations in the OT_MSP interface allow the SSP to inform the MSP of the following events:

- status update of an order or a service order (oRStatusUpdate, sRStatusUpdate) and
- status update of an order or a service order at a pre-defined frequency (oRStatusWindowUpdate, sRStatusWindowUpdate).

3.2 TINA-C Computational Modelling

TINA-C architecture [55] is decomposed into four main subsets: service architecture, network architecture, management architecture and computing architecture. Among these architectures, TINA-C computing architecture [52] defines concepts and principles for designing and building software and software support environment in a telecommunications information network (e.g., Bellcore's Advanced Intelligent Network). The concepts are based on the Open

Distributed Processing (ODP) standard [9] which is a standard for the definition of generic, non-telecommunications specific distributed systems.

The major TINA computational modeling concepts [52] are:

- computational object: The components of a distributed application are represented as computational objects. Computational objects are the units of structure and distribution.
- computational interface: While computational objects are the units of structure and encapsulation of (application-specific) services, interfaces are the units of provision of services; they are the places at which computational objects can interact and obtain services.

ODL-95 [54] supports separation of computational object design from computational interface design. The separation offers freedom to application developers for independent declaration of objects and interfaces. An interface is not necessarily designed for a specific object. It can be used by any object where appropriate. Applying this separation principle to our context, we can develop customer-SP order handling interfaces independent of application developers designing customer or SP order handling objects. Specification of computational objects for order handling is beyond the scope of discussion.

The ODL-95 interface template is a template for specifying a computational interface. It consists of a behavior specification, a service attribute specification, and either an operational interface signature or a stream interface signature. The behavior specification gives an informal specification of interface operations, and includes a specification of ordering constraints on the operations. The service attribute describes the non-functional properties of an interface, such as qualities of service provided by the interface. This attribute is not used in our interface specification.

3.3 Service Negotiation Interfaces

3.3.1 SN_MSP Interface

The operations in this interface are described here.

- void rFPCreate(

in	AdditionalInfo	additionalInfo,
in	DueDate	dueDate,
in	ExpectedResponsibility	expectedResponsibility,
in	GeneralInfo	generalInfo,
in	ProjectDescription	description
in	ProjectRequirements	requirements,
in	PurchaseTermsAndConditions	termsAndConditions,
in	Requestor	requestor,
in	RFPName	name,
out	CancelRequestedIndicator	cancelRequestedIndicator,
out	EnteredDate	enteredDate,
out	RFPID	rFPID,
out	RFPState	state,
out	RFPStatus	status,
out	RFPStatusTime	statusTime,

);

The MSP invokes this operation to enter a request for proposal. it is optional to supply the name input parameter. In case it is not supplied, it will be initialized with the NULL value. After the operation has been performed, an rFP object is created.

- void rFPStatusTrack(

in	RFPID	rFPID,
out	RFPStatus	status,

out RFPStatusTime statusTime,

raises (UnknownRFPID)

);

The MSP invokes this operation to track the current status of a request for proposal. The rFPStatus output parameter gives the current status, and the rFPStatusTime output parameter indicates when this status was last validated or updated.

- void pRModify(

in PRID pRID,

in PRReplaceList replace,

raises (ChangeDenied,

ReplaceDenied,

UnknownPRID)

);

The MSP invokes this operation to request to modify a proposal. The only pR attribute which can be requested to be modified by the MSP is expiryTime. The replaceDenied exception is raised if the SSP does not agree to the modification. The changeDenied exception is raised if the modification request does not obey the constraints in the pR state model.

- void pRView(

in PRID pRID,

in PRAttrIDList pRAttrIDList,

out PRAttrList pRAttrList,

raises (UnknownPRID)

);

The MSP invokes this operation to view one or more pR attributes specified in the pRAttrIDList input parameter. These attributes are returned in the pRAttrList output parameter.

- void oPAcknowledge(

in OPID oPID,

in Disposition disposition,

raises (UnknownOPID)

);

The MSP invokes this operation to indicate whether it agrees to the current terms in the option. The disposition value can be either AGREED or DISAGRRED.

- void oPModify(

in OPID oPID,

in OPAddList add,

in OPRemoveList remove,

in OPReplaceList replace,

raises (AddDenied,

ChangeDenied,

NegotiationRejected,

RemoveDenied,

ReplaceDenied,

UnknownOPID)

);

This operation is the primary means for the MSP to negotiate on an option with the SSP. The MSP invokes this operation to request modification of the hagglingInfo and/or serviceItemList attribute(s).

- void oPView(

in OPID oPID,

in OPAttrIDList oPAttrIDList,

out OPAttrList oPAttrList,

raises (UnknownOPID)

);

The MSP invokes this operation to view one or more oP attributes specified in the oPAttrIDList input parameter. These attributes are returned in the oPAttrList output parameter.

3.3.2 SN_SSP Interface

The operations in this interface are described here.

- void oPAcknowledgeNotify(

  in Disposition disposition,

  in OPID oPID,

  raises (UnknownOPID)

  );

The SSP-originated notification is used to inform the MSP that the SSP has agreed to the current terms in the option. The value of the disposition input parameter is either AGREED or DISAGREED. If the dispositon value is AGREED, the invocation would change the state of the oP object to NEGOTIATION_OVER.

- void oPCancelNotify(

  in OPID oPID

  );

The SSP-originated notification is used to inform the MSP that an option has been cancelled.

- void oPModificationNotify(

  in OPID oPID,

  in OPAttrList attrList

  );

This operation is the primary means for the SSP to negotiate on an option with the MSP. It serves to inform the MSP of the modification of an option. The oP attributes which

were modified are specified in the attrList input parameter. These attributes include hagglingInfo and serviceItemList.

- void pRCancelNotify(

  in PRID pRID

  );

The SSP-originated notification is used to inform the MSP of the cancellation of a proposal.

- void pRCreationNotify(

  in PRID pRID,

  in PRAttrList pRAttrList

  );

The SSP-originated notification is used to inform the MSP of the creation of a proposal offered to a request for proposal. The pRAttrList input parameter specifies the pR attributes which are supplied by the SSP at the creation time.

- void pRModificationNotify(

  in PRID pRID,

  in PRAttrList pRAttrList

  );

The SSP-originated notification is used to inform the MSP of the modification of a proposal. The pR attributes which were modified are specified in the pRAttrList input parameter. The pR attributes which can be modified by the SSP are expiryTime, proposer and pRState.

- void rFPStatusUpdate(

  in RFPID rFPID,

  in RFPStatus status,

  in RFPStatusTime statusTime

);

The SSP-originated notification is used to inform the MSP of a status change of a request for proposal. The status input parameter gives the current status, and the statusTime input parameter indicates when the status was changed.

3.4 Service Ordering Interfaces

3.4.1 SO_MSP Interface

The operations in this interface are described here.

- void oRCancel(
  in ORClearancePerson clearancePerson,
  in ORID oRID,
  raises (CancellationDenied,
  UnknownORID)
  );

The MSP invokes this operation to cancel an active order. The cancellationDenied exception can be raised if the state of the oR object at the time of invocation is CLOSED.

- void oRCreate(
  in BillingInfo billingInfo,
  in ContractID contractID,
  in ORAttrList oRAttrList,
  in ORPriority priority,
  in ORRequestor requestor,
  in ORType type,
  in ServiceItemList serviceItemList,
  in ORAttrList inputOrAttrList,

```
out     ORAttrList          outputOrAttrList,
out     ORID                oRID,
out     ORState             state,
out     ORStatus            status,
out     ORStatusTime        statusTime,
out     SRList              sRlist,
);
```

The invoker invokes this operation to place an order. The oR attributes which may be optionally supplied by the MSP are specified in the inputOrAttrList input parameter. They include:

- acceptanceDate,
- additionalORStatusInfo,
- chargeDate,
- chargeCode,
- committedDate,
- fulfillmentDate,
- initialORID,
- oPID,
- oRClearancePerson,
- oRCloseOutVerification,
- oRDescription,
- oRDialog,
- oRInstallationPerson,
- oRName,
- oRRequestor,
- oRStatusWindow,

- projectCode and
- requestedDate.

An execution of this operation will trigger the creation of an oR object using the oR-create action. The attributes of the oR object are initialized using the attributes supplied at the creation time. The oR attributes which may be optionally supplied by the SSP are specified in the outputOrAttrList output parameter. They include:

- cancelRequestedIndicator,
- committedDate,
- facilityTestDate,
- oREngineergingPerson,
- oRRepresentative and
- readyForServiceDate.

- void oRModify(

  in ORID oRID,

  in ORReplaceList replace,

  raises (ChangeDenied,

  ReplaceDenied,

  UnknownORID)

  );

The MSP invokes this operation to request modification of an order. The replace input parameter specifies the oR attributes which are used to replace the existing oR attributes. The MSP can request to have the following oR attributes replaced:

- billingInfo,
- chargeDate,
- committedDate,
- facilityTestDate,

- oRClearancePerson,
- oRCloseOutVerification,
- oRDialog,
- oRInstallationPerson,
- oRPriority and
- oRRequestor.

- void oRVerify(

in	ORClearancePerson	clearancePerson,
in	ORCloseOutVerification	verify,
in	ORID	oRID
raises	(UnknownORID)	

);

The MSP invokes this operation to verify an order fulfillment. Since an order may contain one or more service orders, the MSP needs to verify the fulfillment of all the service orders which have not yet been verified. If the fulfillment of one of the service orders is denied, the fulfillment of the entire order is denied. The clearancePerson input parameter identifies the person who performs the verification. The verify input parameter indicates whether the order fulfillment is verified or denied.

- void oRView(

in	ORID	oRID,
in	ORAttrIdList	attrIdList,
out	ORAttrList	attrList,
raises	(Unknown ORID)	

);

The MSP invokes this operation to view one or more oR attributes specified in the attrIdList input parameter. These attributes are returned in the attrList output parameter.

```
- void sRCancel(
    in      SRClearancePerson       clearancePerson,
    in      SRID                    sRID,
    raises  (CancellationDenied,
             UnknownSRID)
    );
```

The MSP invokes this operation to cancel an outstanding service order. The cancellationDenied exception can be raised if the state of the sR object at the time of invocation is CLOSED.

```
- void sRModify(
    in      SRID                    sRID,
    in      SRAddList               add,
    in      SRRemoveList            remove,
    in      SRReplaceList           replace,
    raises  (AddDenied,
             ChangeDenied,
             RemoveDenied,
             ReplaceDenied,
             UnknownSRID)
    );
```

The MSP invokes this operation to request modification of a service order. The add input parameter specifies the components which may be added to the specified set-valued sR attributes. The remove input parameter specifies the components which may be removed from the specified set-valued sR attributes. And the replace input parameter specifies the sR attribute values which are used to replace the specified sR attributes.

The MSP can request to add a component to or remove a component from the endUserInfoList and/or sRLocationInfoList attribute(s).

The MSP can request to have one or more of the following sR attributes replaced:

- endUserInfoList,
- sRDialog,
- sRLocationInfoList,
- sRPriority,
- sRSpecificInfoList and
- sRStatusWindow.

- void sRVerify(

  in SRClearancePerson clearancePerson,

  in SRCloseOutVerification verify,

  in SRID sRID

  raises (UnknownSRID)

  );

The MSP invokes this operation to verify a service order fulfillment. The clearancePerson input parameter identifies the person who performs the verification. The verify input parameter indicates whether the service order fulfillment is verified or denied.

- void sRView(

  in SRID sRID,

  in SRAttrIdList attrIdList,

  out SRAttrList attrList,

  raises (UnknownSRID)

  );

The MSP invokes this operation to view one or more sR attributes specified in the attrIdList input parameter. These attributes are returned in the attrList output parameter.

3.4.2 SO_SSP Interface

The operations in this interface are described here.

- void oRModificationNotify(

  in ORID oRID,

  in ORAttrList attrList

  );

The SSP-originated notification is used to inform the MSP of the modification of an order. The oR attributes which are modified are specified in the attrList input parameter. They can include:

  - facilityTestDate,
  - oRDialog,
  - oREngineeringPerson,
  - oRRepresentative and
  - oRState.

- void sRModificationNotify(

  in SRID sRID,

  in SRAttrList attrList

  );

The SSP-originated notification is used to inform the MSP of the modification of a service order. The modified sR attributes are specified in the attrList input parameter. They can include:

  - sRDialog,
  - sRSpecificInfoList and
  - sRState.

- void oRHistoryNotify(

in ORAttrList historyList,

);

The SSP-originated notification is used to inform the MSP that an oR history log record is about to be created for the oR object which is in the CLOSED state. The historyList input parameter is used to specify history information of the closed oR object. The oR attributes which may appear in the historyList input parameter may include:

- acceptanceDate,
- cancelRequestedIndicator,
- committeddate,
- facilityTestDate,
- fulfillmentTestDate,
- orClearancePerson
- oRCloseOutVerification,
- oRDescription,
- oRID,
- oRReceivedDate,
- oRRequestor,
- readyForServiceDate,
- requesteddate,
- serviceItemList and
- sRList.

- void oRDeletionNotify(

in ORID oRID

);

This SSP-originated notification is used to inform the MSP of the deletion of a closed oR object from the order handling system. The oRID input parameter specifies the deleted oR object.

3.5 Order Tracking Interfaces

3.5.1 OT_MSP Interface

The operations in this interface are described here.

- void oRWindowSet(
  in ORID oRID,
  in ORStatusWindow statusWindow,
  raises (UnknownORID)
  );

The MSP invokes this operation to set the size of the order status window. The SSP is supposed to generate an order status progress report at a frequency determined by this window size. This report would provide information on the status of each service order described in the order.

- void oRStatusTrack(
  in ORID oRID,
  out ORStatus status,
  out ORStatusTime statusTime
  raises (UnknownORID)
  );

The MSP invokes this operation to track the current status of an order. The status output parameter gives the current status, and the statusTime input parameter indicates when this status was last validated or updated.

- void sRWindowSet(

  in SRID sRID,

  in SRStatusWindow sStatusWindow,

  raises (UnknownSRID)

  );

The MSP invokes this operation to set the size of the service order status window. The SSP is supposed to generate a service order status progress report at a frequency determined by this window size.

- void sRStatusTrack(

  in SRID oRID,

  out SRStatus status,

  out SRStatusTime statusTime

  raises (UnknownSRID)

  );

The MSP invokes this operation to track the current status of a service order. The status output parameter gives the current status, and the statusTime input parameter indicates when this status was last validated or updated.

3.5.2 OT_SSP Interface

The operations in this interface are described here.

- void oRStatusUpdate(

  in ORID oRID,

  in ORStatus status,

  in ORStatusTime statusTime

  );

The SSP-originated notification is used to inform the MSP of a status change of an order. The status input parameter gives the current status, and the statusTime input parameter indicates when the status was changed.

- void oRStatusWindowUpdate(

  in ORID oRID,

  in ORStatus status,

  in ORStatusTime statusTime,

  in AdditionalORStatuaInfo additionalORStatusInfo

  );

The SSP-originated notification is used to inform the MSP of the latest status of an order at a pre-defined frequency. If the status has not changed, the notification should include the additionalORStatusInfo input parameter describing what progress the SSP has made in processing the order.

- void sRStatusUpdate(

  in SRID SRID,

  in SRStatus status,

  in SRStatusTime statusTime

  );

The SSP-originated notification is used to inform the MSP of a status change of a service order. The status input parameter gives the current status, and the statusTime input parameter specifies when the status was changed.

- void sRStatusWindowUpdate(

  in SRID SRID,

  in SRStatus status,

  in SRStatusTime statusTime,

  in AdditionalSRStatusInfo additionalSRStatusInfo

);

The SSP-originated notification is used to inform the MSP of the latest status of a service order at a pre-defined frequency. If the status has not changed, the notification would include the additionalSRStatusInfo input parameter describing what progress the SSP has made in processing the service order.

3.6 Filtering of Notifications

The capability to filter SSP-originated notifications is useful so that the MSP can control which notifications it would want to receive. To support filtering, a notification server object can be deployed. The notification server object would examine the notifications received from the SSP, filter them according to what the MSP would want to receive, and forward the filtered notifications to the MSP. It would support the NS (Notification Service) interface which is defined in [52]. This section briefly describes the NS interface and uses an example to illustrate how the notification server interacts with the MSP and the SSP.

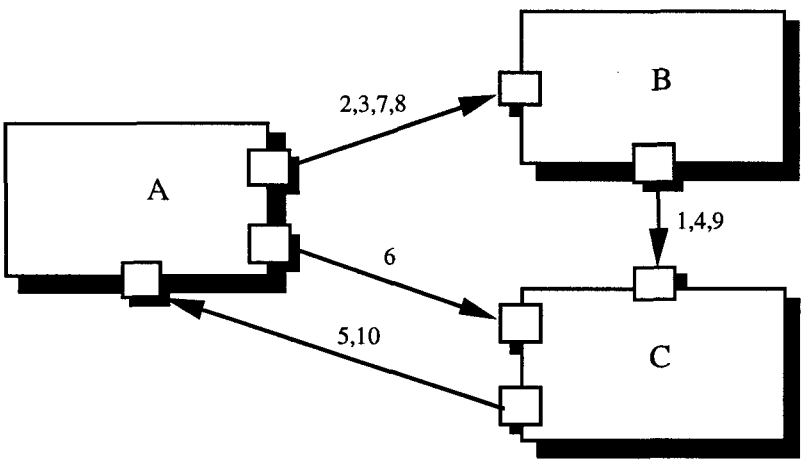

Figure 5 An Example Involving a NotificationServer

The NS interface provides notification management functionalities to receive, filter and forward notifications to recipients that have expressed interest in receiving the notifications. It is characterized by the following operations:

- wantToEmit: this operation allows a computational object (e.g., an agent of rFP, pR or oP) to register its intention of emitting notifications fired by an object type.
- wantToReceive: this operation allows a computational object to register its intention of receiving notifications.

- modifyIntention: this operation allows a computational object to modify the filter or its intention to receive/emit notifications.

Next, we use an example to illustrate how the notification server interacts with the MSP and the SSP. The example illustrates a scenario in the pre-ordering phase. The following computational objects are involved in the example:

- A: This computational object is responsible for pre-ordering in an MSP's organization. It requires the SN_MSP and NS interfaces. It supports the SN_SSP interface.
- B: This computational object is responsible for pre-ordering in an SSP's organization. It requires the SN_SSP and NS interfaces. It supports the SN_MSP interface.
- C: This is a notification server. It requires the SN_SSP interface and supports the the SN_MSP and NS interface.

The following scenario describes how these objects can interact with each other during the pre-ordering phase:

1. B invokes the wantToEmit operation in the NS interface of C to register its intention of forwarding notifications in the SN_MSP interface.

2. A invokes the wantToReceive operation in the NS interface of C to register its intention of receiving notifications in the SN_SSP interface.

3. A invokes the rFPCreate operation in the SN_MSP interface of B to enter a request for proposal.

4. B invokes the pRCreationNotify operation in the SN_SSP interface of C to inform C that a proposal has been created.

5. C invokes the pRCreationNotify operation in the SN_SSP interface of A to inform A that a proposal has been created.

6. Suppose that A only wants to receive notifications on events of the pR object created in step 3. It invokes the modifyIntenion operation in the NS interface of C to specify a filter.

7. A invokes the oPView operation of the SN_MSP interface of B to view an option (for simplicity sake, we can assume that the proposal has only one option).

8. Assume A wants to negotiation on the terms of the option. It invokes the oPModification operation of SN_MSP interface of B to modify some of the terms in the option.

9. Assume that B agrees to the modification. It invokes the oPAcknowledge operation (with the "AGREED" disposition value) of the SN_SSP interface of C to inform C that it agrees to the modification.

10. C then invokes the oPAcknowledge operation (with the "AGREED" disposition value) of the SN_MSP interface of A to inform A that B object agrees to the modification.

CHAPTER 4

SP-TO-SP ORDER HANDLING INFORMATION SPECIFICATION

This section gives the quasi-GDMO+GRM specification of the SP-SP order handling information object types.

4.1 rFP Specification

rFP OBJECT TYPE
CHARACTERIZED BY rFP-package PACKAGE
BEHAVIOUR rFP-Behaviour BEHAVIOUR DEFINED AS
"

COMMENTS: The rFP information object type represents the management view of requests for proposals which are entered by an MSP in the pre-ordering phase. In the request for proposal, the MSP specifies the mandatory and optional requirements of a project. After the SSP has reviewed the request for proposal, it is expected to determine if any of its services meet the requirements. If so, it will respond with a proposal. The rFP information object type has a status attribute allowing the MSP to track the progress which the SSP has made in processing the request for proposal.;

INVARIANT: The invariant properties of the rFP information object type are described by the rFP state model. When an rFP object is created, it is in the QUEUED state. When the SSP starts to review the request for proposal, its state is changed to OPEN/ACTIVE. After reviewing, the SSP may decide to prepare a proposal or give up. If the SSP decides to prepare a proposal and create a pR object, the state of the rFP object is changed to the CLEARED state. As long as the rFP object is in the CLEARED state, the rFP object can be referenced by a pR object. After the pR object is closed, there is no reason to maintain the rFP object, hence the state of the rFP object is changed to the CLOSED state. When the state of the rFP object is either QUEUED or OPEN/ACTIVE, it can be also changed to the CLOSED state if the MSP requests to cancel the request for proposal.;

rFP-create():(x:RFP)
PRECONDS: The MSP has entered a request for proposal.;
POSTCONDS: An rFP object is created and initialized with attributes supplied by the MSP and the SSP at the creation time. The creation triggers the firing of rFP-create-notification.;

rFP-delete(x:RFP):()
PRECONDS: The rFPState is CLOSED.;
POSTCONDS: The rFP object is deleted from the order handling system. The deletion triggers the firing of rFP-delete-notification.;

additionalInfo
This provides additional information (such as vendor qualification, proposal evaluation procedure and required documents) on the project which this request for proposal is about.

additionalInfo-GET():(x:AdditionalInfo)

cancelRequestedIndicator
This indicates whether the MSP has requested to cancel the request for proposal. Initially, this attribute is initialized with the FALSE value.

cancelRequestedIndicator-GET():(x:CancelRequestedIndicator)

cancelRequestedIndicator-REPLACE(x:CancelRequestedIndicator):()
PRECONDS: The cancelRequestedIndicator is FALSE, and the rFPState is neither NEGOTIATION_OVER nor CLOSED.
POSTCONDS: The rFPState is CLOSED.

dueDate
This indicates the last day for the SSP to respond with a proposal to the request for proposal.

dueDate-GET():(x:DueDate)

enteredDate
This indicates the date on which the rFP object was created.

enteredDate-GET():(x:EnteredDate)

expectedResponsibility
This indicates the SSP's responsibility such as buying necessary insurance, obtaining permits and licenses for the project, etc.

expectedResponsibility-GET():(x:ExpectedResponsibility)

generalInfo
This indicates the principal features of the project so that the SSP can quickly determine which, if any, of their services meet the requirements. This may include type of service being requested, required completion date, schedule of major events, etc.

generalInfo-GET():(x:GeneralInfo)

projectDescription
This gives a narrative description of the organization (of the MSP) requesting the service and the environment in which it will be used. This may include the objectives of the project, the primary lines of business, the addresses of principal location, the description of existing systems and reasons why they are to be replaced.

projectDescription-GET():(x:ProjectDescription)

projectRequirements

This explains what the project must accomplish. The MSP should state clearly which requirements are mandatory and which are optional. The SSP is supposed to check feasibility of services fitting these requirements.

projectRequirements-GET():(x:ProjectRequirements)

purchaseTermsAndConditions
This includes warranty, payment terms, criteria for system acceptance, and performance binding etc.

purchaseTermsAndConditions-GET():(x:PurchaseTermsAndConditions)

requestor
This provides information on the person issuing the request for proposal.

requestor-GET():(x:Requestor)

rFPID
This is the identifier assigned to the rFP object by the SSP at the creation time.

rFPID-GET():(x:RFPID)

rFPName
This is the name assigned to the request for proposal by the MSP.

rFPName-GET():(x:RFPName)

rFPState
This gives the current state of the rFP object. The states of the rFP objects are described in the rFP state model.

rFPState-GET():(x:RFPState)

rFPState-REPLACE(x:RFPState):()
PRECONDS: The state change should obey the constraints of the rFP state model.;

rFPStatus
This indicates the progress which the SSP has made in processing the request for proposal Examples of status values are:

- pendingForReview
- reviewingInProgress
- reviewingComplete
- proposalPreparing
- proposalPreparationDone
- proposalDispatched
- closeOutNoInterest
- closeOutProposalCancelled
- closeOutRFPCancelled
- closeOutNoResponseToProposal
- closeOutProposalExpired

rFPStatus-GET():(x:RFPStatus)

rFPStatus-REPLACE(x:RFPStatus):()

rFPStatusTime
This specifies when the rFPStatus was last validated or changed.

rFPStatusTime-GET():(x: RFPStatusTime)

rFPStatusTime-REPLACE(x: RFPStatusTime):()
PRECONDS: The rFPStatus is changed.

rFP-avc-notification():(rFP-avc-notification-list:RFPAttrList)
TRIGGERINGCONDS: The triggering-condition is that there is an attribute value change of an rFP object. The rFP-avc-notification-list output parameter is used to specify the rFP attribute(s) which were modified.;

rFP-create-notification():(rFP-create-notification-list:RFPAttrList)
TRIGGERINGCONDS: The triggering-condition is that an rFP object has been created using the rFP-create action. The rFP attributes in the rFP-create-notification-list output parameter are attributes supplied at the creation time.;

rFP-delete-notification():(rFPID:rFPID)
TRIGGERINGCONDS: The triggering-condition is that an rFP object has been deleted using the rFP-delete action. The rFPID output parameter identifies the deleted rFP object.;
";

ATTRIBUTES

additionalInfo
PERMITTED VALUES: AdditionalInfo
GET,

cancelRequestedIndicator
PERMITTED VALUES: CancelRequestedIndicator
GET-REPLACE,

dueDate
PERMITTED VALUES: DueDate
GET-REPLACE

enteredDate
PERMITTED VALUES: EnteredDate
GET,

expectedResponsibility
PERMITTED VALUES: ExpectedResponsibility
GET,

generalInfo
PERMITTED VALUES: GeneralInfo
GET,

projectDescription
PERMITTED VALUES: ProjectDescription
GET,

projectRequirements
PERMITTED VALUES: ProjectRequirements
GET,

purchaseTermsAndConditions
PERMITTED VALUES: PurchaseTermsAndConditions
GET,

requestor
PERMITTED VALUES: Requestor
GET,

rFPID
PERMITTED VALUES: RFPID
GET,

rFPName
PERMITTED VALUES: RFPName
GET,

rFPState
PERMITTED VALUES: RFPID
GET-REPLACE,

rFPStatus
PERMITTED VALUES: RFPStatus
GET-REPLACE,

rFPStatusTime
PERMITTED VALUES: RFPStatusTime
GET-REPLACE,

ACTIONS ;

NOTIFICATIONS

rFP-avc-notification():(rFP-avc-notification-list:RFPAttrList),

rFP-create-notification():(rFP-create-notification-list:RFPAttrList),

rFP-delete-notification():(rFPID:rFPID);

REGISTERED AS {TBD};

4.2 pR Specification

pR OBJECT TYPE
CHARACTERIZED BY pR-package PACKAGE
BEHAVIOUR pR-Behaviour BEHAVIOUR DEFINED AS
"

COMMENTS: The pR information object type represents the management view of proposals offered by the SSP in the pre-ordering phase. If the SSP decides to respond to a request for proposal, it will offer a proposal containing one or more options. A pR object together with one or more oP objects are created by the SSP.;

INVARIANT: The invariant properties of the pR information object type are described by the pR state model. The state of a proposal is determined by the state of each option in the proposal. Thus the state of a proposal is a tuple, where each tuple element denotes the state of an option in the proposal.;

pR-create():(x:Proposal)
PRECONDS:; The SSP has responded with a proposal.
POSTCONDS: A pR object is created and initialized with attributes supplied by the SSP at the creation time. The creation triggers the firing of pR-create-notification.;
pR-delete(x:Proposal):()
PRECONDS: The state of the pR object is CLOSED.;
POSTCONDS: The pR object is deleted from the order handling system. The deletion triggers the firing of pR-delete-notification.;

expiryTime
This indicates when the terms in the proposal will expire.

expiryTime-GET():(x:ExpiryTime)

expiryTime-REPLACE(x:ExpiryTime):()

oPList
This is a list of names of the oP objects for the options in the proposal.

oPList-GET():(x:OPList)

pRID
This is the identifier assigned to the pR object by the SSP at the creation time.

pRID-GET():(x:PRID)

pRState
This gives the current state of the proposal. The state of the proposal can be specified by a tuple where each tuple element denotes the state of an option in the proposal.

pRState-GET():(x:PRState)

pRState-REPLACE(x:PRState):()

proposer
This provides information on the person offering the proposal.

proposer-GET():(x:Proposer)

proposer-REPLACE(x:Proposer):()

rFPID
This identifies the rFP object for which this pR object is created.

rFPID-GET():(x:RFPID)

pR-avc-notification():(pR-avc-notification-list:PRAttrList)
TRIGGERINGCONDS: The triggering-condition is that there is an attribute value change of a pR object. The pR-avc-notification-list output parameter is used to specify the pR attribute(s) which were modified.;

pR-create-notification():(pR-create-notification-list:PRAttrList)
TRIGGERINGCONDS: The triggering-condition is that a pR object has been created using the pR-create action. The pR attributes in the pR-create-notification-list output parameter are those which are supplied by the SSP at the creation time.;

pR-delete-notification():(pRID:PRID)
TRIGGERINGCONDS: The triggering-condition is that a pR object has been deleted using the pR-delete action. The pRID output parameter identifies the deleted pR object.;
";

ATTRIBUTES

expiryTime
PERMITTED VALUES: ExpiryTime
GET-REPLACE,

oPList
PERMITTED VALUES: oPList
GET,

pRID
PERMITTED VALUES: PRID
GET,

pRState
PERMITTED VALUES: pRState
GET-REPLACE,

proposer

PERMITTED VALUES: Proposer
GET-REPLACE,

rFPID
PERMITTED VALUES: RFPID
GET;

ACTIONS ;

NOTIFICATIONS

pR-avc-notification():(pR-avc-notification-list:PRAttrList),

pR-create-notification():(pR-create-notification-list:PRAttrList),

pR-delete-notification():(pRID:PRID);

REGISTERED AS {TBD};

4.3 oP Specification

oP OBJECT TYPE
CHARACTERIZED BY oP-package PACKAGE
BEHAVIOR oP-Behaviour BEHAVIOR DEFINED AS
"

COMMENTS: The oP information object type represents the management view of options which are created by the SSP in the pre-ordering phase. When the SSP is ready to offer a proposal to a request for proposal, it will create a pR object as well as one or more oP objects which are linked to the pR object. The pR information object type is characterized by attributes which can be negotiated between the MSP and the SSP.; ;

INVARIANT: The invariant of information object type are described by the oP state model. When an oP object is created, it is in the NEGOTIATING state, i.e., the option is subject to negotiation. During the negotiation, it remains in the NEGOTIATING state as long as both sides have not agreed to the terms in the option. When both sides have agreed to the terms, the state is changed to NEGOTIATION_OVER. When the expiry date of the proposal has expired, the state is changed to CLOSED. When the oP object is in the NEGOTIATING state, its state can be changed to CLOSED for other reasons, e.g.,

- the SSP wants to cancel the option, or
- the MSP does not respond to the option in due time (the duration may be determined by the business agreement between the MSP and the SSP).;

oP-create():(x:OP)
PRECONDS: The SSP has offered a proposal containing the option.;
POSTCONDS: An oP object is created and initialized with attributes supplied by the SSP at the creation time. The creation triggers the firing of oP-create-notification.;

oP-delete(x:OP):()
PRECONDS: The oPState is CLOSED. ;
POSTCONDS: The oP object is deleted from the order handling system. The deletion triggers the firing of oP-delete-notification.;

hagglingInfo
This indicates the items (e.g., price) which are subject to negotiation between the MSP and the SSP.

hagglingInfo-GET():(x:HagglingInfo)

hagglingInfo-REPLACE(x:HagglingInfo):()

oPID
This is the identifier assigned to the oP object by the SSP at the creation time.

oPID-GET():(x:oPID)

oPState
This gives the current state of the oP object. The states of the oP object are described in the oP state model.

oPState-GET():(x:OPState)

oPState-REPLACE(x:OPState):()
PRECONDS: The state change should obey the constraints of the oP state model.;

pRID
This identifies the pR object to which this oP object is linked.

pRID-GET():(x:pRID)

serviceItemLIst
This is a list of service orders proposed in the option to meet the requirements in the request for proposal.

serviceItemList-GET():(x:ServiceItemList)

serviceItemList-REPLACE(x:ServiceItemList):()

serviceItemList-ADD(x:ServiceItemList):()

serviceItemList-REMOVE(x:ServiceItemList):()

oP-avc-notification():(oP-avc-notification-list:OPAttrList)
TRIGGERINGCONDS: The triggering-condition is that there is an attribute value change of an oP object. The oP-avc-notification-list output parameter is used to specify the modified oP attribute(s).;

oP-create-notification():(oP-create-notification-list:OPAttrList)

TRIGGERINGCONDS: The triggering-condition is that an oP object has been created using the oP-create action. The oP attributes in the oP-create-notification-list output parameter are those which are supplied at the creation time.;

oP-delete-notification():(oPID:OPID)
TRIGGERINGCONDS: The triggering-condition is that an oP object has been deleted using the oP-delete action. The oPID output parameter identifies the deleted oP object.;
";

ATTRIBUTES

hagglingInfo
PERMITTED VALUES: HagglingInfo
GET-REPLACE,

oPID
PERMITTED VALUES: OPID
GET,

oPState
PERMITTED VALUES: OPState
GET-REPLACE,

pRID
PERMITTED VALUES: PRID
GET,

serviceItemList
PERMITTED VALUES: ServiceItemList
GET-REPLACE ADD-REMOVE;

ACTIONS ;

NOTIFICATIONS

oP-avc-notification():(oP-avc-notification-list:OPAttrList),

oP-create-notification():(oP-create-notification-list:OPAttrList),

oP-delete-notification():(oPID:OPID);

REGISTERED AS {TBD};

4.4 oR Specification

oR OBJECT TYPE
CHARACTERIZED BY oR-package PACKAGE
BEHAVIOUR oR-Behaviour BEHAVIOUR DEFINED AS

"

COMMENTS: The oR information object type represents the management view of orders which are placed by an SSP in the ordering phase. An order placed by the the SSP may be triggered by a customer-originated order. After the SSP has received the order request, it will create an oR object. ;

INVARIANT: The invariant properties of the oR information object types are described by the oR state model. The state of an order is determined by the state of each service order which may appear in the order. Thus the state of an order is a tuple, where each tuple element represents the state of a service order appearing in the order.;

oR-create():(x:OR)
PRECONDS: The MSP has placed an order with the SSP.;
POSTCONDS: An oR object is created and initialized with attributes supplied by the MSP and the SSP at the creation time. An sR object is created for each service order appearing in the order. The creation triggers the firing of oR-create-notification. ;

oR-delete(x:OR):()
PRECONDS: The state of the oR object is CLOSED.;
POSTCONDS: The oR object is deleted from the order handling system. The deletion triggers the firing of oR-delete-notification.;

acceptanceDate

This indicates the date at which fulfillment was accepted by the MSP (if supplied, this attribute value is initialized with the NULL value).

acceptanceDate-GET():(x:AcceptanceDate)

additionalORStatusInfo

This provides progress information on the ordering process when there is no recent status change.

additionalORStatusInfo-GET():(x:AdditionalORStatusInfo)

additionalORStatusInfo-REPLACE(x:AdditionalORStatusInfo):()

billingInfo
This provides billing information (e.g., billing account number, billing address, billing contact person) on the order.

billingInfo-GET():(x:BillingInfo)

billingInfo-REPLACE(x:BillingInfo):()

cancelRequestedIndicator
This indicates whether the MSP has requested for the cancellation of the order (if supplied, this attribute is initialized with the FALSE value).

cancelRequestedIndicator-GET():(x:CancelRequestedIndicator)

cancelRequestedIndicator-REPLACE(x:CancelRequestedIndicator):()

chargeDate
This indicates the date on which charges will be billed.

chargeDate-GET():(x:ChargeDate)

chargeDate-REPLACE(x:ChargeDate):()

chargeCode
This identifies the MSP's charge code (i.e., purchase order number) for the order.

chargeCode-GET():(x:ChargeCode)

chargeCode-REPLACE(x:ChargeCode):()

committedDate
This indicates the date to which order fulfillment is committed by the SSP.

committedDate-GET():(x:CommittedDate)

committedDate-REPLACE(x:CommittedDate):()

contractID
This identifies the contract specifying the ordering terms and conditions.

contractID-GET():(x:ContractID)

facilityTestDate
This indicates the date on which cooperative testing of the services (involving the MSP and SSP) is scheduled.

facilityTestDate-GET():(x:FacilityTestDate)

facilityTestDate-REPLACE(x:FacilityTestDate):()

fulfillmentDate
This indicates the date on which the order fulfillment occurred (if supplied, this attribute is initialized with the NULL value).

fulfillmentDate-GET():(x:FullfillmentDate)

initialORID
This identifies the order of which this order forms a part.

initialORID-GET():(x:InitialORID)

oPID
This identifies the oP object which triggers the issuance of this order.

oPID-GET():(x:OPID)

oRClearancePerson
This provides information on the person who is responsible for either requesting for the cancellation of the order or verifying the fulfillment of the order.

oRClearancePerson-GET():(x:ORClearancePerson)

oRClearancePerson-REPLACE(x:ORClearancePerson):()
PRECONDS: The orCloseOutVerification was denied.;

oRCloseOutVerification
This indicates whether the MSP has verified order fulfillment, denied order fulfillment or taken no action (if supplied, this attribute is initialized with the NO_ACTION value).

oRCloseOutVerification-GET():(x:ORCloseOutVerification)

oRCloseOutVerification-REPLACE(x:ORCloseOutVerification):()

oRDescription
This gives a summary of the order.

oRDescription-GET():(x:ORDescription)

oRDescription-REPLACE(x:ORDescription):()

oRDialog
This enables interaction to take place between the MSP and the SSP at each stage of the ordering process. The "dialog text" is free format text. The contents are replaced by the new "dialog text" as the dialog progresses during the service ordering process.

oRDialog-GET():(x:ORDialog)

oRDialog-REPLACE(x:ORDialog):()

oREngineeringPerson
This provides information on the person to be contacted with technical/design queries regarding the service implementation.

oREngineeringPerson-GET():(x:OREngineeringPerson)

oREngineeringPerson-REPLACE(x:OREngineeringPerson):()

oRID
This is the identifier assigned to the oR object by the SSP at the creation time.

oRID-GET():(x:ORID)

oRInstallationPerson
This provides information on the person to be contacted with queries/information regarding the installation schedule.

oRInstallationPerson-GET():(x:ORInstallationPerson)

oRInstallationPerson-REPLACE(x:ORInstallationPerson):()

oRName
This is a name assigned to the order by the MSP.

oRName-GET():(x:ORName)

oRPriority
This indicates the priority of the order.

oRPriority-GET():(x:ORPriority)

oRPriority-REPLACE(x:ORPriority):()

oRReceivedDate
This indicates the date on which the order was received.

oRReceivedDate-GET():(x:ORReceivedDate)

oRRepresentative
This provides information on the person who received the order.

oRRepresentative-GET():(x:ORRepresentative)

oRRepresentative-REPLACE(x:ORRepresentative):()

oRRequestor
This provides information on the person placing the order.

oRRrequestor-GET():(x:ORRequestor)

oRRequestor-REPLACE(x:ORRequestor):()

oRState
This gives the current state of the oR object. The state of an the order can be specified by a tuple, where each tuple element denotes the state of a service order appearing in the order.

oRState-GET():(x:ORState)

oRState-REPLACE(x:ORState):()

oRStatus
This describes the progress which the SSP has made in processing the order.

oRStatus-GET():(x:ORStatus)

oRStatus-REPLACE(x:ORStatus):()

oRStatusTime
This specifies when the oRStatus was last validated or changed.

oRStatusTime-GET():(x:ORStatusTime)

oRStatusTime-REPLACE(x:ORStatusTime):()

oRStatusWindow
This specifies the frequency for the SSP to generate a progress report on the order.

oRStatusWindow-GET():(x:ORStatusWindow)

oRStatusWindow-REPLACE(x:ORStatusWindow):()

oRType
This indicates whether this is an initial order or a subcontracted order (in our case, this is a subcontracted order).

oRType-GET():(x:ORType)

projectCode
This is used to associate related orders.

projectCode-GET():(x:ProjectCode)

readyForServiceDate
This indicates the date at which all the requested services were tested by the SSP to be operational (if supplied, this attribute is initialized with the NULL value).

readyForServiceDate-GET():(x:ReadyForServiceDate)

requestDate
This is the date which order fulfillment was requested by the MSP.

requestedDate-GET():(x:RequestedDate)

requestedDate-REPLACE(x:RequestedDate):()

serviceItemList
This identifies the service orders appearing in the order request. The MSP may provide the following information to each service order:

- endUserInfoList,
- requestType,
- serviceType,
- sRDialog,

- sRLocationInfoList,
- sRName,
- sRPriority,
- sRSpecificInfoList and
- sRStatusWindow.

For each service order in the serviceItemList, the SSP would create an sR object at the time the oR object is created. Each sR object is initialized with the (above) attributes supplied by the MSP as well as attributes supplied by the SSP.

serviceItemList-GET():(x:ServiceItemList)

sRList
This gives the names of the sR objects created for the order.

sRList-GET():(x:SRList)

sRName
This is a name assigned to the service order by the MSP.

sRName-GET():(x:SRName)

oR-avc-notification():(oR-avc-notification-list:ORAttrList)
TRIGGERINGCONDS: The triggering-condition is that there is an attribute value change of an oR object. The oR-avc-notification-list output parameter is used to specify the modified oRAttribute(s).;

oR-create-notification():(oR-create-notification-list:ORAttrList)
TRIGGERINGCONDS:The triggering-condition is that an oR object has been created using the oR-create action. The oR attributes in the oR-create-notification-list output parameter are those which are supplied at the creation time.;

oR-delete-notification():(oRID:ORID)
TRIGGERINGCONDS:The triggering-condition is that an oR object has been deleted using the oR-delete action. The oRID output parameter identifies the deleted oR object.;

oR-history-notification():(oR-history-notification-list:ORAttrList)
TRIGGERINGCONDS:The triggering-condition is that the state of the oR history log record is about to be created for the closed order. Typically, this occurs when an sR-history-notification is fired for the last service order (in the proposal) which has not yet fired sR-history-notification. The oR-history-notification-list is used to specify the the oR attributes which may appear in an oR history log record.;

oR-progress-notification():(oR-progress-notification-list:ORAttrList)
TRIGGERINGCONDS:The triggering-condition is that the window size (as defined by the oRStatusWindow attribute) was 0. The oR-progress-notification-list output parameter shows the oR attributes which may appear in a progress report.;
".

ATTRIBUTES

acceptanceDate
PERMITTED VALUES: ORAcceptanceDate
GET,

additionalORStatusInfo
PERMITTED VALUES: AdditionalORStatusInfo
GET-REPLACE,

billingInfo
PERMITTED VALUES: BillingInfo
GET-REPLACE,

cancelRequestedIndicator
PERMITTED VALUES: CancelRequestedIndicator
GET-REPLACE,

chargeDate
PERMITTED VALUES: ChargeDate
GET-REPLACE,

chargeCode
PERMITTED VALUES: ChargeCode
GET-REPLACE,

commitedDate
PERMITTED VALUES: ORCommitedDate
GET-REPLACE,

contractID
PERMITTED VALUES: ContractID
GET,

facilityTestDate
PERMITTED VALUES: FacilityTestDate
GET-REPLACE,

fulfillmentDate
PERMITTED VALUES: ORFulfillmentDate
GET,

initialORID
PERMITTED VALUES: InitialORID
GET,

oPID
PERMITTED VALUES: OPID
GET,

oRClearancePerson

PERMITTED VALUES: ORClearancePerson
GET-REPLACE,

oRCloseOutVerification
PERMITTED VALUES: ORCloseOutVerification
GET-REPLACE,

oRDescription
PERMITTED VALUES: ORDescription
GET,

oRDialog
PERMITTED VALUES: ORDialog
GET-REPLACE,

oREngineeringPerson
PERMITTED VALUES: OREngineeringPerson
GET-REPLACE,

oRID
PERMITTED VALUES: ORID
GET,

oRInstallationPerson
PERMITTED VALUES: ORInstallationPerson
GET-REPLACE,

oRName
PERMITTED VALUES: ORName
GET,

oRPriority
PERMITTED VALUES: ORPriority
GET-REPLACE,

oRReceivedDate
PERMITTED VALUES: ORReceivedTime
GET,

oRRepresentative
PERMITTED VALUES: ORRepresentative
GET-REPLACE,

oRRequestor
PERMITTED VALUES: ORRequester
GET-REPLACE,

oRState
PERMITTED VALUES: ORState
GET-REPLACE,

```
    oRStatus
        PERMITTED VALUES: ORStatus
        GET-REPLACE,

    oRStatusTime
        PERMITTED VALUES: ORStatusTime
        GET-REPLACE,

    oRStatusWindow
        PERMITTED VALUES: ORStatusWindow
        GET-REPLACE,

    oRType
        PERMITTED VALUES: ORType
        GET,

    projectCode
        PERMITTED VALUES: ProjectCode
        GET,

    readyForServiceDate
        PERMITTED VALUES: ReadyForServiceDate
        GET,

    requestedDate
        PERMITTED VALUES: RequestedDate
        GET-REPLACE,

    serviceItemList
        PERMITTED VALUES: ServiceItemList
        GET

    sRList
        PERMITTED VALUES: SRList
        GET

    sRName
        PERMITTED VALUES: SRName
    GET;

ACTIONS ;

NOTIFICATIONS

    oR-avc-notification():(oR-avc-notification-list:ORAttrList),

    oR-create-notification():(oR-create-notification-list:ORAttrList),

    oR-delete-notification():(oRID:ORID),

    oR-history-notification():(oR-history-notification-list:ORAttrList),
```

oR-progress-notification():(oR-progress-notification-list:ORAttrList);

REGISTERED AS {TBD};

4.5 sR Specification

sR OBJECT TYPE
CHARACTERIZED BY sR-package PACKAGE
BEHAVIOUR sR-Behaviour BEHAVIOUR DEFINED AS
"

COMMENTS: The sR information object type represents the management view of service orders which are placed by an MSP in an order request during the ordering phase. After the SSP has received an order request, an sR object is created for every service item in the serviceItemList attribute of the order. The sR object provides information on the type of the service, the end-users of the service and locations to be served.;

INVARIANT: The invariant properties of the sR information object type are described by the sR state model. When an sR object is created, it may be initially in the QUEUED state, meaning that the service order has not yet been processed. As soon as the SSP has performed some work on the service order, the state of the sR object is changed to OPEN/ACTIVE. When the SSP believes that a facility test on the order is ready, the state of the sR object is changed to CLEARED. After the MSP has verified fulfillment of the service order, the state of the sR object is changed to CLOSED. If the fulfillment is denied, the state is changed to OPEN/ACTIVE. When the state of the sR object is either QUEUED or OPEN/ACTIVE, it can be changed to CLOSED if MSP requests to cancel the service order.;

sR-create():(x:SR)
PRECONDS: The MSP has placed an order.;
POSTCONDS: An sR object is created and initialized with attributes supplied by the MSP and the SSP at the creation time. The creation triggers the firing of sR-create-notification.;

sR-delete(x:SR):()
PRECONDS: The state of the sR object is CLOSED.;
POSTCONDS: The sR object is deleted from the order handling system. The deletion triggers the firing of sR-delete-notification.;

additionalSRStatusInfo
This is used to provide additional progress information on the service order when there is no recent status change.

additionalSRStatusInfo-GET():(x:AdditionalSRStatusInfo)

additionalSRStatusInfo-REPLACE(x:AdditionalSRStatusInfo):()

cancelRequestedIndicator

This indicates whether the MSP has requested to cancel the service order.

cancelRequestedIndicator-GET():(x:CancelRequestedIndicator)

cancelRequestedIndicator-REPLACE(x:CancelRequestedIndicator):()

endUserInfoList
This provides information on the end users to be served.

endUserInfoList-GET():(x:EndUserInfoList)

endUserInfoList-REPLACE(x:EndUserInfoList):()

oRID
This identifies the oR object to which this sR object is linked.

oRID-GET():(x:ORID)

requestType
This indicates the request type which can be either newService, disconnect, move or change.

requestType-GET():(x:RequestType)

serviceType
This specifies the type of the service being ordered.

serviceType-GET():(x:ServiceType)

sRDialog
This enables interaction to take place between the MSP and the SSP at each stage of the service ordering process. The "dialog text" is free format text. The contents are replaced by the new "dialog text" as the dialog progresses during the service ordering process.

sRDialog-GET():(x:SRDialog)

sRDialog-REPLACE(x:SRDialog):()

sRID
This is the identifier assigned to the sR object by the SSP at the creation time.

sRID-GET():(x:SRID)

sRLocationInfoList
This provides information on the locations to be served.

sRLocationInfoList-GET():(x:SRLocationInfoList)

sRName
This is the name assigned to the service order by the MSP.

sRName-GET():(x:SRName)

sRPriority
This indicates the priority of the sR object.

sRPriority-GET():(x:SRPriority)

sRPriority-REPLACE(x:SRPriority):()

sRSpecificInfoList
This provides information specific to the service order.

sRSpecificInfoList-GET():(x:sRSpecificInfoList)

sRSpecificInfoList-REPLACE(x:sRSpecificInfoList):()

sRState
This gives the current state of the sR object. The states of the sR object are described in the sR state model.

sRState-GET():(x:SRState)

sRState-REPLACE(x:SRState):()

sRStatus
This describes the progress which the SSP has made in processing the service order. Examples of status values are:

- installPending,
- installInProgress,
- installSuccessful,
- installFail,
- serviceTestPending,
- serviceTestInProgress
- serviceTestSuccessful,
- serviceTestFail,
- facilityTestPending,
- facilityTestInProgress,
- facilityTestSuccessful,
- facilityTestFail,
- readyForService and
- closeOut.

sRStatus-GET():(x:SRStatus)

sRStatus-REPLACE(x:SRStatus):()

sRStatusTime
This specifies when the sRStatus was last validated or changed.

sRStatusTime-GET():(x:SRStatusTime)

sRStatusTime-REPLACE(x:SRStatusTime):()

sRStatusWindow
This specifies the frequency for the SSP to generate a progress report on the service order.

sRStatusWindow-GET():(x:SRStatusWindow)

sRStatusWindow-REPLACE(x:SRStatusWindow):()

sR-avc-notification():(sR-avc-notification-list:SRAttrList)
TRIGGERINGCONDS: The triggering-condition is that there is an attribute value change of an sR object. The sR-avc-notification-list output parameter is used to specify the modified sR attribute(s).;

sR-create-notification():(sR-create-notification-list:SRAttrList)
TRIGGERINGCONDS: The triggering-condition is that an sR object has been created using the sR-create action. The sR attributes in the sR-create-notification-list output parameter are those which are supplied at the creation time.;

sR-delete-notification():(sRID:SRID)
TRIGGERINGCONDS: The triggering-condition is that an sR object has been deleted using the sR-delete action. The sRID output parameter identifies the deleted sR object.;

sR-history-notification():(sR-historyEvent-notification-list:SRAttrList)
TRIGGERINGCONDS:The triggering-condition is that an sR history log record is about to be created for the closed service order. The sR-history-notification-list shows the sR attributes which may appear in an sR history log record.;

sR-progress-notification():(sR-progress-notification-list:SRAttrList)
TRIGGERINGCONDS: The triggering-condition is that the window size (as defined by the sRStatusWindow attribute) was 0. The sR-progress-notification-list output parameter shows the sR attributes which may appear in a progress report.;
";

ATTRIBUTES

additionalSRStatusInfo
PERMITTED VALUES:AdditionalSRStatusInfo
GET-REPLACE,

cancelRequestedIndicator
PERMITTED VALUES:CancelRequestedIndicator
GET-REPLACE,

endUserInfoList
PERMITTED VALUES:EndUserInfoList
GET-REPLACE,

oRID
    PERMITTED VALUES:ORID
    GET,

requestType
    PERMITTED VALUES:RequestType
    GET,

serviceType
    PERMITTED VALUES:ServiceType
    GET,

sRDialog
    PERMITTED VALUES:SRDialog
    GET-REPLACE,

sRID
    PERMITTED VALUES:SRID
    GET,

sRLocationInfoList
    PERMITTED VALUES:SRLocationInfoList
    GET-REPLACE ADD-REMOVE,

sRName
    PERMITTED VALUES:SRName
    GET,

sRPriority
    PERMITTED VALUES:SRPriority
    GET-REPLACE,

sRSpecificInfoList
    PERMITTED VALUES:SRSpecificInfoList
    GET-REPLACE,

sRState
    PERMITTED VALUES:SRState
    GET-REPLACE,

sRStatus
    PERMITTED VALUES:SRStatus
    GET-REPLACE,

sRStatusTime
    PERMITTED VALUES:SRStatusTime
    GET-REPLACE,

sRStatusWindow
    PERMITTED VALUES:SRStatusWindow

```
        GET-REPLACE;
    ACTIONS ;
    NOTIFICATIONS
        sR-avc-notification():(sR-avc-notification-list:SRAttrList),
        sR-create-notification():(sR-create-notification-list:SRAttrList),
        sR-delete-notification():(sRID:SRID),
        sR-history-notification():(sR-history-notification-list:SRAttrList),
        sR-progress-notification():(sR-progress-notification-list:SRAttrList);
REGISTERED AS {TBD};
```

CHAPTER 5

SERVICE ORDER HANDLING INTERFACE SPECIFICATION

This section gives the ODL-95 specification of the following SP-SP order handling interfaces:

- SN_MSP,
- SN_SSP,
- SO_MSP,
- SO_SSP,
- OT_MSP and
- OT_SSP.

An ODL interface specification consists of three components:

- behavior component: This component contains the informal behavior specification and the usage specification.
- service attribute component: This component, which describes the non-functional attributes (e.g., quality of service), is not deployed in my specification.
- operational interface signature: This component contains either an operational interface signature or a stream interface signature. My specification only gives operational interface signatures.

In specifying an operational signature, it is necessary to define the main types of the input or output parameters of each operation. In addition, it is necessary to define the supporting types used in the definition of the main types. These main and supporting types are defined in Appendix B.

5.1 SN_MSP Specification

interface SN_MSP {

behavior

"The SN_MSP interface is used by an MSP to track the status of a request for proposal or negotiate on an option with an SSP. The operations in this interface allow a MSP to:

- enter a request for proposal (rFPCreate),
- view an option or a proposal (oPView, pRView),
- modify an option or a proposal (oPModify, pRModify),
- acknowledge the SSP (oPAcknowledge) and
- track the status of a request for proposal (rFPStatusTrack).

The operations are described below:

rFPCreate:
The MSP invokes this operation to enter a request for proposal. If the invocation is successful, an rFP object is created.

rFPStatusTrack:
The MSP invokes this operation to track the current status of a request for proposal. The rFPStatus output parameter gives the current status of the rFP object, and the rFPStatusTime output parameter indicates when this status was last validated or updated.

pRModify:
The MSP invokes this operation to request to modify a proposal. The only pR attribute which the MSP can request to modify is expiryTime. The replaceDenied exception is raised if the SSP does not agree to the modification.

pRView:
The MSP invokes this operation to view one or more pR attributes which are specified in the pRAttrIDList input parameter. These attributes are returned in the pRAttrList output parameter.

oPAcknowledge:
The MSP invokes this operation to indicate whether it agrees to the current terms in the option. The disposition input parameter is used to convey its agreement or disagreement.

oPModify:
This operation is the primary means for the MSP to negotiate on an option with the SSP. The oP attributes which can be negotiated are hagglingInfo and/or serviceItemList.

oPView:
The MSP invokes this operation to view one or more oP attributes which are specified in the oPAttrIDList input parameter. These attributes are returned in the oPAttrList output parameter.

usage

"The MSP can enter a request for proposal by invoking rFPCreate. If the invocation is successful, an rFP object is created. Thereafter, the MSP can track the status of the rFP object by invoking rFPStatusTRack. Using the SN_SSP interface, the SSP can

offer a proposal and create a pR object. The MSP can view the pR object or the oP objects linked to it by invoking pRView or oPView. During negotiation, it can request to modify an oP object by invoking oPModify. It can convey its disposition on an option to the SSP by invoking oPAcknowledge."

```
/* Main and supporting types are defined in the Appendix. */

void        rFPCreate(
    in      AdditionalInfo                  additionalInfo,
    in      DueDate                         dueDate,
    in      ExpectedResponsibility          expectedResponsibility,
    in      GeneralInfo                     generalInfo,
    in      ProjectDescription              description
    in      ProjectRequirements             requirements,
    in      PurchaseTermsAndConditions      termsAndConditions,
    in      Requestor                       requestor,
    in      RFPName                         name,
    out     CancelRequestedIndicator        cancelRequestedIndicator,
    out     EnteredDate                     enteredDate,
    out     RFPID                           rFPID,
    out     RFPState                        state,
    out     RFPStatus                       status,
    out     RFPStatusTime                   statusTime
    );

void        rFPStatusTrack(
    in      RFPID                   rFPID,
    out     RFPStatus               status,
    out     RFPStatusTime           statusTime,
    raises  (UnknownRFPID)
    );

void        pRModify(
    in      PRID            pRID,
    in      PRReplaceList   replace,
    raises  (ChangeDenied,
             ReplaceDenied,
             UnknownPRID)
    );

void        pRView(
    in      PRID                pRID,
    in      PRAttrIDList        pRAttrIDList,
    out     PRAttrList          pRAttrList,
    raises  (UnknownPRID)
    );

void        oPAcknowledge(
    in      OPID            oPID,
    in      Disposition     disposition,
    raises  (UnknownOPID)
```

```
    );

void        oPModify(
    in      OPID                oPID,
    in      OPAddList           add,
    in      OPRemoveList        remove,
    in      OPReplaceList       replace,
    raises  (AddDenied,
             ChangeDenied,
             NegotiationRejected,
             RemoveDenied,
             ReplaceDenied,
             UnknownOPID)
    );

void        oPView(
    in      OPID                oPID,
    in      OPAttrIDList        oPAttrIDList,
    out     OPAttrList          oPAttrList,
    raises  (UnknownOPID)
    );
}
```

5.2 SN_MSP Specification

interface SN_SSP {

behavior

"The SN_SSP interface is used by an SSP to send information of a request for proposal to an MSP, or negotiate on an option with an MSP. Specifically, the operations in this interface allow the SSP to inform the MSP of the following events:

- creation of a proposal (pRCreationNotify).
- modification of an option or a proposal (oPModificationNotify, pRModificationNotify),
- acknowledgement of the MSP either positively or negatively (oPAcknowledgeNotify),
- cancellation of an option or a proposal (oPCancelNotify, pRCancelNotify) and
- notification of a status update of the request for proposal (rFPStatusUpdate).

These operations are described below:

oPAcknowledgeNotify:
This SSP-originated notification is used by the SSP to inform the MSP that it has agreed to the current terms in the option. The value of the disposition input parameter is either AGREED or DISAGREED. If the disposition value is AGREED, the invocation would change the state of the oP object to NEGOTIATION_OVER.

oPCancelNotify:

This SSP-originated notification is used to inform the MSP of the cancellation of an option.

oPModificationNotify:
This notification is the primary means for the SSP to negotiate on an option with the MSP. It serves to inform the MSP of the modification of an option. The oP attributes which were modified are specified in the attrList input parameter. These attributes include hagglingInfo and serviceItemList.

pRCancelNotify:
The SSP-originated notification is used to inform the MSP of the cancellation of a proposal.

pRCreationNotify:
This SSP-originated notification is used to inform the MSP of the creation of a proposal offered to a request for proposal. The pRAttrList input parameter specifies the pR attributes which are supplied by the SSP at the creation time.

pRModificationNotify:
This SSP-originated notification is used to inform the MSP of the modification of a proposal. The pR attributes which were modified are specified in the pRAttrList input parameter. They can include expiryTime, proposer and pRState.

rFPStatusUpdate:
This SSP-originated notification is used to inform the MSP of a status change of a request for proposal. The status input parameter gives the current status, and the statusTime input parameter indicates when the status was changed.";

usage
"The operations in this interface are invoked by the SSP to inform the customer of the creation of a pR object (using pRCreationNotify), the modification of a pR object (using pRModificationNotify) and status update of an rFP object (using rFPStatusUpdate). During negotiation, the SSP can invoke oPAcknowledge to convey its disposition on an option to the MSP. If the SSP has decided to cancel a proposal or an option, it can invoke pRCancelNotify or oPCancelNotify.";

/* Main and supporting types are defined in the Appendix. */

```
oneway    void          oPAcknowledgeNotify(
    in    Disposition             disposition,
    in    OPID                    oPID,
    raises (UnknownOPID)
    );

oneway    void          oPCancelNotify(
    in    OPID                    oPID
    );

oneway    void          oPModificationNotify(
    in    OPID                    oPID,
    in    OPAttrList              attrList
```

```
    );

  oneway    void        pRCancelNotify(
    in      PRID                pRID
    );

  oneway    void        pRCreationNotify(
    in      PRID                pRID,
    in      PRAttrList          pRAttrList
    );

  oneway    void        pRModificationNotify(
    in      PRID                pRID,
    in      PRAttrList          pRAttrList
    );

  oneway    void        rFPStatusUpdate(
    in      RFPID               rFPID,
    in      RFPStatus           status,
    in      RFPStatusTime       statusTime
    );
  }
```

5.3 SO_MSP Specification

interface SO_MSP {

behavior

"The SO_MSP interface is provided by an MSP to place an order which may contain one or more service orders. The operations in this interface allow the MSP to:

- place an order (oRCreate),
- modify an order or a service order (oRModify, sRModify),
- view an order or a service order (oRView, sRView),
- cancel an order or a service order (oRCancel, sRCancel) and
- verify the fulfillment of an order or a service order (oRVerify, sRVerify).

The operations are described below:

oRCancel:
The MSP invokes this operation to cancel an outstanding order. The cancellationDenied exception can be raised if the state of the oR object is CLOSED at the time of invocation.

oRCreate:
The MSP invokes this operation to place an order. The oR attributes which may be optionally supplied by the MSP are specified in the inputOrAttrList input parameter. They can include:

- acceptanceDate,
- additionalORStatusInfo,
- chargeDate,

- chargeCode,
- committedDate,
- fulfillmentDate,
- initialORID,
- oPID,
- oRClearancePerson,
- oRCloseOutVerification,
- oRDescription,
- oRDialog,
- oRInstallationPerson,
- oRName,
- oRRequestor,
- oRStatusWindow,
- projectCode and
- requestedDate.

An execution of this operation will trigger the creation of an oR object using the oR-create action. The attributes of the oR object are initialized using the attributes supplied at the creation time. The oR attributes which may be optionally supplied by the SSP are specified in the outputOrAttrList output parameter. They include:

- cancelRequestedIndicator,
- committedDate,
- facilityTestDate,
- oREngineergingPerson,
- oRRepresentative and
- readyForServiceDate.

oRModify:
The MSP invokes this operation to request modification of an order. The replace input parameter specifies the oR attributes used to replace the existing oR attributes. The MSP can request to have the following oR attributes replaced:

- billingInfo,
- chargeDate,
- committedDate,
- facilityTestDate,
- oRClearancePerson,
- oRCloseOutVerification,
- oRDialog,
- oRInstallationPerson,
- oRPriority and
- oRRequestor.

oRVerify:
The MSP invokes this operation to verify an order fulfillment. The MSP needs to verify the fulfillment of all the service orders (in the order) which have not yet been verified. If the fulfillment of one of the service orders is denied, the fulfillment of the entire order is denied. The clearancePerson input parameter identifies the person who performs the verification. The verify input parameter indicates whether the order fulfillment is verified or denied.

oRView:
The MSP invokes this operation to view one or more oR attributes which are specified in the attrIdList input parameter. These attributes are returned in the attrList output parameter.

sRCancel:
The MSP invokes this operation to cancel an active service order. The cancellationDenied exception can be raised if the state of the sR object at the time of invocation is CLOSED.

sRModify:
The MSP invokes this operation to request modification of a service order. The add input parameter specifies the components which may be added to the specified set-valued sR attributes. The remove input parameter specifies the components which may be removed from the specified set-valued sR attributes. And the replace input parameter specifies the sR attribute values which are used to replace the specified sR attributes.

The MSP can request to add a component to or remove a component from the endUserInfoList and/or sRLocationInfoList attribute(s).

The MSP can request to have one or more of the following sR attributes replaced:

- endUserInfoList,
- sRDialog,
- sRLocationInfoList,
- sRPriority,
- sRSpecificInfoList and
- sRStatusWindow.

sRVerify:
The MSP invokes this operation to verify a service order fulfillment. The clearancePerson input parameter identifies the person who performs the verification. The verify input parameter indicates whether the service order fulfillment is verified or denied.

sRView:
The MSP invokes this operation to view one or more sR attributes specified in the attrIdList input parameter. These attributes are returned in the attrList output parameter.";

usage
"The MSP can place an order by invoking oRCreate. If the invocation is successful, an oR object as well as one or more sR objects are created. Thereafter, the MSP can view these objects by invoking oRView or sRView. If it wants to cancel an order or a service order, it can invoke oRCancel or sRCancel. If an order or a service order is ready to be verified for fulfillment, the MSP can invoke oRVerify or sRVerify."

/* Main and supporting types are defined in the Appendix. */

```
void        oRCancel(
    in      ORClearancePerson       clearancePerson,
    in      ORID                    oRID,
    raises  (CancellationDenied,
             UnknownORID)
    );

void        oRCreate(
    in      BillingInfo             billingInfo,
    in      ContractID              contractID,
    in      ORAttrList              oRAttrList,
    in      ORPriority              priority,
    in      ORRequestor             requestor,
    in      ORType                  type,
    in      ServiceItemList         serviceItemList,
    in      ORAttrList              inputORAttrList,
    out     ORAttrList              outputORAttrList,
    out     ORID                    oRID,
    out     ORState                 state,
    out     ORStatus                status,
    out     ORStatusTime            statusTime,
    out     SRList                  sRList
    );

void        oRModify(
    in      ORID                    oRID,
    in      ORReplaceList           replace,
    raises  (ChangeDenied,
             ReplaceDenied,
             UnknownORID)
    );

void        oRVerify(
    in      ORClearancePerson       clearancePerson,
    in      ORCloseOutVerification  verify,
    in      ORID                    oRID
    raises  (UnknownORID)
    );

void        oRView(
    in      ORID                    oRID,
    in      ORAttrIdList            attrIdList,
    out     ORAttrList              attrList,
    raises  (UnknownORID)
    );

void        sRCancel(
    in      SRClearancePerson       clearancePerson,
    in      SRID                        sRID,
    raises  (CancellationDenied,
             UnknownSRID)
```

```
    );

void        sRModify(
    in      SRID                sRID,
    in      SRAddList           add,
    in      SRRemoveList        remove,
    in      SRReplaceList       replace,
    raises  (AddDenied,
             ChangeDenied,
             RemoveDenied,
             ReplaceDenied,
             UnknownSRID)
    );

void        sRVerify(
    in      SRClearancePerson           clearancePerson,
    in      SRCloseOutVerification      verify,
    in      SRID                        sRID
    raises  (UnknownSRID)
    );

void        sRView(
    in      SRID                sRID,
    in      SRAttrIdList        attrIdList,
    out     SRAttrList          attrList,
    raises  (UnknownSRID)
    );
}
```

5.4 SO_SSP Specification

interface SO_SSP {

behavior

"The SO_MSP interface is used by the SSP to inform the MSP of events relating to the placement of an order or a service order. These events include:

- modification of an active order or a service order (oRModificatioNotify, sRModificationNotify),
- history information of a closed order (oRHistoryNotify) and
- deletion of a closed order (oRDeletionNotify)

The operations in the SO_MSP interface are described below:

oRModificationNotify:
The SSP invokes this operation to inform the MSP that it has modified one or more oR attributes specified in the attrList input parameter. These attributes can include:

- facilityTestDate,
- oRDialog,
- oREngineeringPerson,

- oRRepresentative and
- oRState.

sRModificationNotify:
The SSP invokes this operation to inform the MSP that it has modified one or more sR attributes specified in the attrList input parameter. These attributes can include:

- sRDialog,
- sRSpecificInfoList and
- sRState.

oRHistoryNotify:
The SSP invokes this operation to inform the MSP that an oR history log record is about to be created for the oR object which is in the closed state. The historyList input parameter is used to specify history information of the closed oR object. The following oR attributes may appear in the historyList input parameter:

- acceptanceDate,
- cancelRequestedIndicator,
- committeddate,
- facilityTestDate,
- fulfillmentTestDate,
- orClearancePerson
- oRCloseOutVerification,
- oRDescription,
- oRID,
- oRReceivedDate,
- oRRequestor,
- readyForServiceDate,
- requesteddate,
- serviceItemList and
- sRList.

oRDeletionNotify:
The SSP invokes this operation to inform the MSP that an oR object has been deleted from the order handling system. The oRID input parameter specifies the deleted oR object.

usage
"The operations in this interface are invoked by the SSP to inform the MSP of modification of an oR/sR object (using oRModificationNotify or sRModificationNotify). After the state of an OR object has been changed to CLOSED, the SSP can invoke oRHistoryNotify to inform the MSP that an oR history log record is about to be created, and eventually oRDelete to inform the MSP that the oR object and its associated sR object(s) have been removed from the order handling system.";

/* Main and supporting types are defined in the Appendix. */

```
oneway    void        oRModificationNotify(
    in    ORID                oRID,
```

```
        in      ORAttrList          attrList
        );

    oneway  void    sRModificationNotify(
        in      SRID                sRID,
        in      SRAttrList          attrList
        );

    oneway  void    oRHistoryNotify(
        in      ORAttrList          historyList
        );

    oneway  void    oRDeletionNotify(
        in      ORID                oRID,
        );

    }
```

5.5 OT_MSP Specification

interface OT_MSP {

behavior

"The OT_MSP interface is used by an MSP to track the status of an active order or service order. The operations in this interface allow the MSP to:

- track the status of an order or a service order (oRStatusTrack, sRStatusTrack) and
- set the status window size of an order or a service order (oRWindowSet, sRWindowSet).

The operations are described below:

oRWindowSet:
The MSP invokes this operation to set the size of the order status window. The SSP is supposed to generate an order status progress report at a frequency determined by this window size. This report would provide information of each service order in the order.

oRStatusTrack:
The MSP invokes this operation to track the current status of an order. The status output parameter gives the current status, and the statusTime input parameter indicates when this status was last validated or updated.

sRWindowSet:
The MSP invokes this operation to set the size of the service order status window. The SSP is supposed to generate a service order status progress report at a frequency determined by this window size.

sRStatusTrack:
The MSP invokes this operation to track the current status of a service order. The status output parameter gives the current status, and the statusTime input parameter indicates when this status was last validated or updated.

```
    usage
        "The MSP can invoke oRStatusTrack or sRStatusTrack to track the status of an order
        or a service order. If it needs a periodic status update, it can invoke oRWindowSet or
        sRWindowSet to set the size of an order status window or service order status
        window."

    /* Main and supporting types are defined in the Appendix. */

    void        oRWindowSet(
        in      ORID                oRID,
        in      ORStatusWindow      statusWindow,
        raises  (UnknownORID)
        );

    void        oRStatusTrack(
        in      ORID                oRID,
        out     ORStatus            status,
        out     ORStatusTime        statusTime
        raises  (UnknownORID)
        );

    void        sRWindowSet(
        in      SRID                sRID,
        in      SRStatusWindow      sRStatusWindow,
        raises  (UnknownSRID)
        );

    void        sRStatusTrack(
        in      SRID                oRID,
        out     SRStatus            status,
        out     SRStatusTime        statusTime
        raises  (UnknownSRID)
        );
    }
```

5.6 OT_SSP Specification

```
interface OT_SSP {
    behavior
        "The OT_SSP interface is used by an SSP to inform the MSP of events relating to the
        status of an order or service order. Specifically, these events include:

            - status update of an order or a service order (oRStatusUpdate, sRStatusUpdate)
              and
            - status update of an order or a service order at a pre-defined frequency
              (oRStatusWindowUpdate, sRStatusWindowUpdate).

        The operations are described below:
```

oRStatusUpdate:
The SSP-originated notification is used to inform the MSP of a status change of an order. The status input parameter gives the current status, and the statusTime input parameter indicates when the status was changed.

oRStatusWindowUpdate:
This SSP-originated notification is used to inform the MSP of the latest status of an order at a pre-defined frequency. If the status has not changed, the notification should include the additionalORStatusInfo input parameter describing what progress the SSP has made in processing the order.

sRStatusUpdate:
The SSP-originated notification is used to inform the MSP of a status change of a service order. The status input parameter gives the current status, and the statusTime input parameter specifies when the status was changed.

sRStatusWindowUpdate:
This SSP-originated notification is used to inform the MSP of the latest status of a service order at a pre-defined frequency. If the status has not changed, the notification should include the additionalSRStatusInfo input parameter describing what progress the SSP has made in processing the service order.

usage
"The operations in this interface are invoked by the SSP to inform the MSP of a status change of an order or service order (using oRStatusUpdate or sRStatusUpdate). If the MSP has set an order status update the MSP of the status of an order or a service order (using oRWindowUpdate or sRWindowUpdate).";

/* Main and supporting types are defined in the Appendix. */

```
oneway    void        oRStatusUpdate(
    in    ORID                    oRID,
    in    ORStatus                status,
    in    ORStatusTime            statusTime
    );

oneway    void        oRStatusWindowUpdate(
    in    ORID                    oRID,
    in    ORStatus                status,
    in    ORStatusTime            statusTime,
    in    AdditionalORStatuaInfo  additionalORStatusInfo
    );

oneway    void        sRStatusUpdate(
    in    SRID                    SRID,
    in    SRStatus                status,
    in    SRStatusTime            statusTime
    );

oneway    void        sRStatusWindowUpdate(
    in    SRID                    SRID,
```

```
    in      SRStatus                status,
    in      SRStatusTime            statusTime,
    in      AdditionalSRStatusInfo  additionalSRStatusInfo
    );
}
```

CHAPTER 6

STORE-AND-FORWARD PARADIGM

6.1 Store-and-Forward

The objective of this chapter is to introduce a management paradigm for network management applications which do not have real time constraints. This paradigm, called the Store-and-Forward Management Paradigm, deploys the store-and-forward mode of communication. It is meant to complement existing management paradigms which deploy connection-oriented or connectionless modes of communication.

Network management applications which fall into the accounting and configuration functional areas do not have real time constraints in general. Let us consider an example of such an application in the configuration functional area: a service ordering tracking application by means of which a customer can track an order. After the customer orders a service from a service provider, the service provider normally designs a service plan for the order, assigns resources to the order and performs testing. In the meantime, the customer can track the status of the order to find out which step the service provider is currently executing. Since the steps take time, the status of the order does not normally change rapidly. Because of this, the customer and the service provider can communicate with each other about the status of the order using a store-and-forward transport mechanism.

Every management paradigm needs to address the two fundamental issues: the transport of management information, and the manipulation of management information. Let us first examine how the Store-and-Forward Management Paradigm addresses these two issues.

On the transport, SNMP [48] uses the connectionless mode of communication, and CMIP [20] uses the connection-oriented mode of communication. The store-and-forward mode of communication lies in between the connectionless mode and the connection-oriented mode. It involves the use of intermediate or relay entities where connection establishment is required

only for the adjacent relay entities. It facilitates communication of applications which do not have real time constraints. As an example, all e-mail applications, also known as mail-enabled applications (MEAs), use the store-and-forward mode of communication. Because the MEAs do not have real-time constraints, their management also should not have real-time constraints. Since the Store-and-Forward Management Paradigm is intended for management of applications which do not have real-time constraints, its transport should be based on the store-and-forward mode of communication. As a matter of fact, the store-and-forward transport carrying the application information should be the same store-and-forward transport carrying the management information.

The manipulation of management information addresses what management operations can be applied and how management information is represented. Since the major concern of management paradigm is the deployment of the store-and-forward communication mode, we should not impose constraints on how the management information is manipulated, which should be determined by the end applications processing the management information. The store-and-forward transport mechanism of management paradigm must be extendible to be able to carry all possible management operations and management information, including those to be introduced in the future.

There are a number of e-mail transport systems. To illustrate store-and-forward management paradigm, we use the X.400 MTS (Message Transfer System). Note that paradigm can be extended to other e-mail systems. Because the X.400 MTS supports multicasting via distribution lists, the paradigm can use this support to provide asynchronous group communication among the management entities in a distributed management application. Both SNMP and CMIP assume point-to-point communication among the management entities. By changing the mode of communication to the store-and-forward mode, the paradigm can still keep the SNMP or CMIP style of representation of management information but extend the communication from point-to-point to point-to-multipoint.

This chapter is organized as follows. Section 6.2 explains the Store-and-Forward Management Functional Model which introduces functional entities such as M-UAs (Management User Agents) and MEs (Management Entities). Section 6.3 defines the Pm content type which is designed to carry all possible representations of management information.

6.2 Store-and-Forward Management Functional Model

The Store-and-Forward Management Functional Model is shown in Figure 6. The Management Messaging System (MMS) consists of M-UAs and the MTS. The MTS is made up of X.400 MTAs (Message Transfer Agents) which have functionalities such as message transfer, message delivery, message submission, message splitting, message conversion, etc.

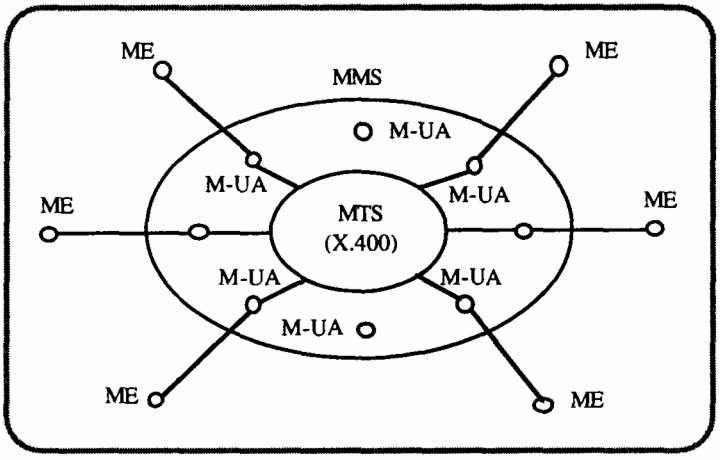

Figure 6 Store-and-Forward Management Functional Model

An ME is a component of a distributed management application. It processes management information. It can play the manager role, the agent role, or both. MEs are typed according to the management protocols they can process. Thus, we can have SNMP-MEs and

CMIP-MEs. The same ME can be an SNMP-ME and a CMIP-ME if it can process both SNMP and CMIP.

M-UAs (Management User Agents) act on behalf of MEs (Management Entities). The major function of an M-UA is to process the Pm content which is defined in Section 6.3. An M-UA interacts with the MTS using the X.400 P3 protocol (known as the MTS access protocol). Thus the MTS sees an M-UA as an MTS user. On the receiving side, when an M-UA receives a message of the Pm content type from an MTA, it processes the content and transforms it into one or more management operations in a management protocol which the recipient ME would understand. On the sending side, after an ME sends one or more management operations to its associated M-UA, the M-UA would create a Pm message and submit the message to the MTS.

We assume a one-to-one correspondence between MEs and M-UAs. This means that every ME has an ORAddress or perhaps an ORName (based on the X.500 standard). The 1988 X.400 standard defines a number of ORAddresses. In the functional model, we assume that every ME has a mnemonic ORAddress, i.e., one which is characterized by attributes such as country name, ADMD name, PRMD name, organization name, organizational unit name, common name, and perhaps domain-defined attributes (DDAs). The DDAs are useful when the management environment can be structured into management domains. Because of the one-to-one correspondence between MEs and M-UAs, it is likely in practice that an ME and its associated M-UA are co-located in the same system. When the M-UA and the ME are co-located, they can communicate with each other using the IPC (Interprocess Communication) facilities offered by the underlying operating system.

Figure 7 shows how an M-UA interacts with its ME. Suppose that the M-UA receives a Pm message, i.e., a message of the Pm content type. The M-UA first parses the content to find out the management operation(s) embedded in the content. For example, if a CMIP operation is embedded in the content, the M-UA will extract the CMIP operation and deliver it

in a message format meaningful to its ME. The design of such a message format for communication between the ME and its associated M-UA is a local issue.

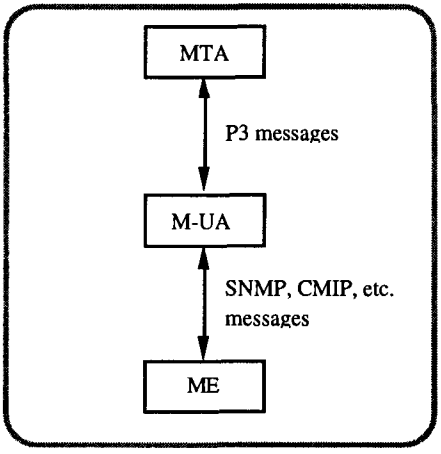

Figure 7 Messages between MTA, M-UA and ME

The X.400 standard defines the concept of a IPM (Interpersonal Message) notification [1] for the IPM (also known as P22) content type. The notion of a Pm notification for the Pm content type needs some explanation. A Pm notification can be either a Pm receipt notification or a Pm non-receipt notification. The purpose of a Pm receipt notification is to inform the originating ME that the recipient ME has received the management operation(s) from its associated M-UA. The issue here is when the recipient M-UA generates a Pm receipt notification. Let us assume that the recipient M-UA has received a Pm message. It would parse the Pm content, extract the management operation(s), and deliver them to the recipient ME. If it encounters an error during the parsing stage or the extraction stage, a Pm non-receipt notification is generated. If no error is encountered, a Pm receipt notification can be generated immediately although some implementation may make sure that the recipient ME has received the management operation(s) before the Pm receipt notification is generated. In Section 6.3, we will elaborate on the structure of the Pm notification.

For the rest of this section, we examine how the X.500 standard [18] can be used to store management information relevant to our management functional model. The following shows three different ways to use the X.500 directory.

The ORName/ORAddress of a recipient does not indicate whether the recipient is an M-UA, an IPM-UA, or a EDI-UA [11-17]. An M-UA can register its management capability to process the Pm content type with the X.500 directory. The MTAs can then use this X.500 knowledge to avoid routing Pm messages to M-UAs which are not management capable.

An ME can register the ORName/ORAddress of its associated M-UA with the X.500 directory. From the AE-title of the recipient ME, the originating M-UA can use this X.500 knowledge to derive the values of the ORName or ORAddress fields of the Pm-header.

While the role of an ME may change over its lifetime, it is unlikely that the agents working for a manager may change dynamically. Therefore, it is useful to store in the X.500 directory the AE-titles of the management agents working on behalf of a manager. A recipient ME can use this X.500 knowledge to avoid performing a management operation requested by an originator ME which is not a manager of the recipient ME.

6.3 Store-and-Forward Management Information Model

The Store-and-Forward Management Information Model defines the structure of a Pm message and the structure of a Pm notification. As mentioned in the introduction, a Pm message must be extendible to be able to carry all possible management operations and management information.

A Pm message has an envelope and a Pm content which is of the Pm content type. The Pm content is used to convey one or more management request or response primitives which can be SNMP, CMIP, DMI (Desktop Management Interface [5]) etc. The Pm content consists of a Pm-header and a Pm-body.

Among the Pm-header fields, there are the IPM header fields which are found in a IPM header. These fields include the mandatory messageId field and the optional fields such as originator, recipient, and repliedTo.

The originator field specifies the originating ME. It may include the AE-title of the ME, and the ORName/ORAddress of the associated M-UA.

The recipient field specifies the recipient ME. In addition to the AE-title of the ME and the ORName/ORAddress of the associated M-UA, it includes three flags: a reply flag to indicate whether a reply is requested, a Pm-receipt-notification flag to indicate whether a Pm receipt notification is expected, and a Pm-non-receipt-notification flag to indicate whether a Pm non-receipt notification is expected. The reply flag must be set if the management operation conveyed by the Pm message is a confirmed operation, e.g., the M-GET operation in CMIP. Consequently, the recipient field is mandatory if the conveyed management operation is confirmed. Some management operations (such as the M-EVENT-REPORT operation in CMIP) can be unconfirmed, meaning that a response is optional. If the originating M-UA sets the reply flag for an unconfirmed management operation, then a response from the recipient ME and hence a reply from the recipient M-UA are required.

The repliedTo field is found in a Pm reply message. It has a messageId value which matches the messageId value of the Pm message to which the Pm reply message is a reply.

Apart from the IPM-like header fields, there are two mandatory fields specific to Pm. They are the tag field and the request/response field. The tag field identifies the management protocol for management operations appearing in the body of the Pm message. Defined as an ASN.1 objectIdentifier type, it has values for management protocols such as SNMP and CMIP, and any management protocol to be designed in the future. It is this field which makes our Pm content type extendible.

The request/response field (0 for request and 1 for response) indicates whether the management operations within the body are either all request primitives or all response

primitives. If the request/response field is set, the Pm message is effectively a Pm reply message to an originating Pm message. The request/response field also has a special value, ERROR. Consider the case when an originating ME sends an M-SET.request primitive to an ME which is not an agent of the originating ME. In this case, the recipient M-UA would create a Pm reply message in which the request/response field has the ERROR value. In the current version of the Pm content type, it is not allowed to have request primitives mixed with response primitives within the Pm-body.

The Pm-body specifies one or more management operations. It consists of a sequence of Pm-body-parts. Contained in each Pm-body-part is a primitive of a management operation. For example, a M-SET request and a M-GET request can appear in the same Pm message as two distinct Pm-body-parts. The management primitives in the message are processed by the recipient ME in the order of appearance.

Pm-body-parts are typed. The current version of the Pm content type supports the following Pm-body-part-types: SNMP, CMIP and DMI [5]. The types of the Pm-body-parts in a Pm message should be consistent with the tag field in the Pm header. For example, if the tag field of a Pm message indicates the CMIP protocol, then all the Pm-body-parts within the message must be of the same type, namely, the CMIP type.

Next let's think about the structure of a Pm-body-part. A Pm-body-part has a sequenceNumber field indicating the relative position of the Pm-body-part within the Pm-body. For example, if the Pm-body has five Pm-body-parts, the sequenceNumbers of the five Pm-body-parts can range from 1 to 5.

The most important field of a Pm-body-part is the managementPrimitive field which is used to specify a management primitive and associated parameters. In the case of CMIP, the associated parameters of a management primitive do not include the invocationIdentifier parameter because the messageId field of the Pm-header essentially plays the same role as the invocationIdentifier parameter.

If the request/response field in the Pm-header is set, the Pm-body-part contains the matchingRequest field. Since the Pm-body-part is meant to behave as a response to an originating Pm message which may contain multiple Pm-body-parts corresponding to multiple request primitives (e.g., multiple M-GET request primitives), the matchingRequest field identifies the Pm-body-part to which this Pm-body-part is a response. For example, if the Pm-body-part is a response to the Pm-body-part whose sequenceNumber is 3, then the matchingRequest field should have the value 3. See Figure 8 for an illustration.

CMIP supports the notion of a partial response. The response to any CMIP request primitive involving the use of scope and filtering can be partial. A partial response uses the linkedIdentifier parameter. In CMIP, all partial responses except the last one to a request have the linkedIdentifier field value equal to the invocationIdentifier parameter value of the request primitive. The last response has the linkedIdentifier value set to 0. When a CMIP manager receives a partial response whose linkedIdentifer value is 0, it knows that it has received all the partial responses. The reason is that CMIP assumes an application association between the manager and the agent. In the model, the communication between the originating ME and the recipient ME is based on a store-and-forward communication mode. Consequently, the partial responses generated by the CMIP agent may not arrive in order at the CMIP manager.

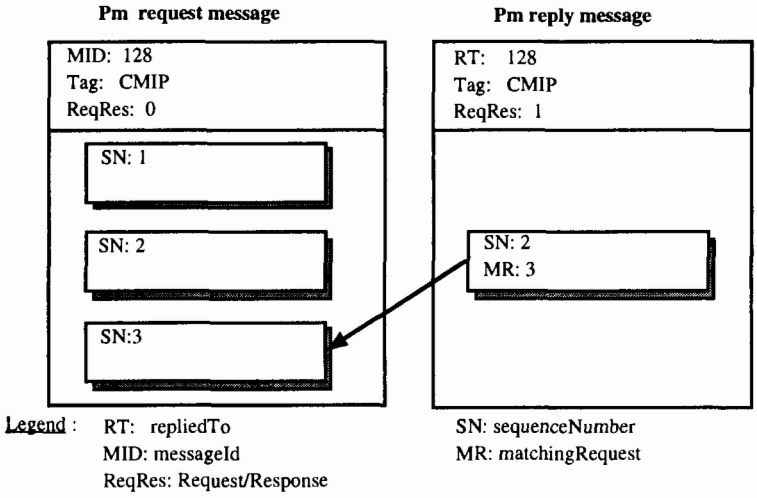

Figure 8 Use of MatchingRequest Field

Consider the situation when a total of six partial responses is generated for some management request primitive. Let us assume that three Pm messages are used to send the six partial responses. Message A contains partial responses 1 and 2, message B contains partial responses 3 and 4, and message C contains partial response 5 and 6. It may happen that the CMIP manager receives messages A and C first before message B. Since message C contains partial response 6 which has the linkedIdentifier value set to 0, the manager might think that it has received all the partial responses, not knowing that another message B is still in the wire. To fix this problem in our store-and-forward framework, we need to define the type of the linkedIdentifier field of a response to be different from that of the linkedIdentifier CMIP parameter. The linkedIdentifier field in a response has two subfields: the count subfield and a more flag. A total response (i.e., one conveying the entire response to a request primitive) has the count subfield value equal to 0 and the more flag set to 0. Partial responses have the count subfields indexed by an integer count starting from 1. Assuming that every partial response increments the integer count by one, then the final partial response should have a count field value equal to the total number of partial responses. As long as the partial response is not the

final response, the more flag value is set to 1. The more flag value of the final response is 0. Back to our example, partial response 6 in message C has the count subfield value equal to 6 and the more flag value equal to 0. After the manager has received messages A and C, it knows from the linkedIdentifier value of partial response 6 that it has yet 2 more partial responses to receive. Figure 9 illustrates our example.

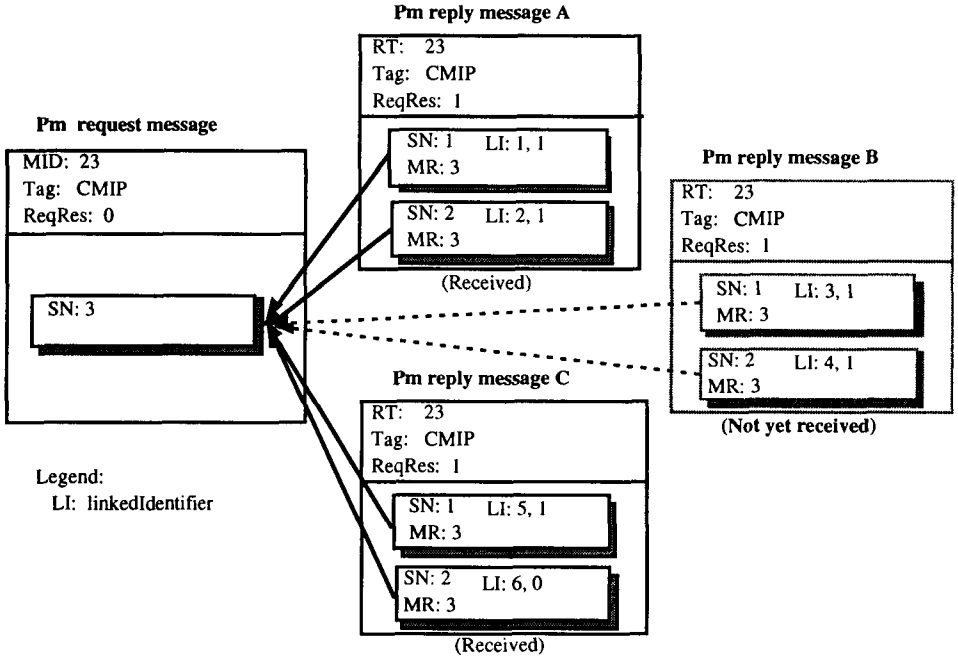

Figure 9 Use of linkedIdentifier Field

For an example to illustrate the different fields of a Pm-body-part, let's consider the case where the management primitive carried is the M-GET.response primitive. The fields of this Pm-body-part are: sequenceNumber, matchingRequest, managementPrimitive, linkedIdentifier, managedObjectClass, managedObjectInstance, currentTime and attributeList.

Next, let's take an example to illustrate that the partial responses in a Pm reply message can be "mixed". Consider the case when an originating Pm message carries two M-GET.request primitives, labelled by sequence numbers 1 and 2 respectively. Assume that five

partial responses is generated for the first M-GET.request primitive, and three partial responses is generated for the second M-GET.request primitive. These partial responses may be conveyed in a mixed Pm reply message, i.e., it carries some partial responses for the first M-GET.request primitive and some partial responses for the second M-GET.request primitive. In the mixed Pm reply message, those Pm-body-parts which are used to convey partial responses for the first M-GET.request primitive have a matchingRequest value equal to 1, while those Pm-body-parts which are used to convey partial responses for the second M-GET.request primitive have a matchingRequest value equal to 2. See Figure 10 for the illustration of a mixed Pm reply message.

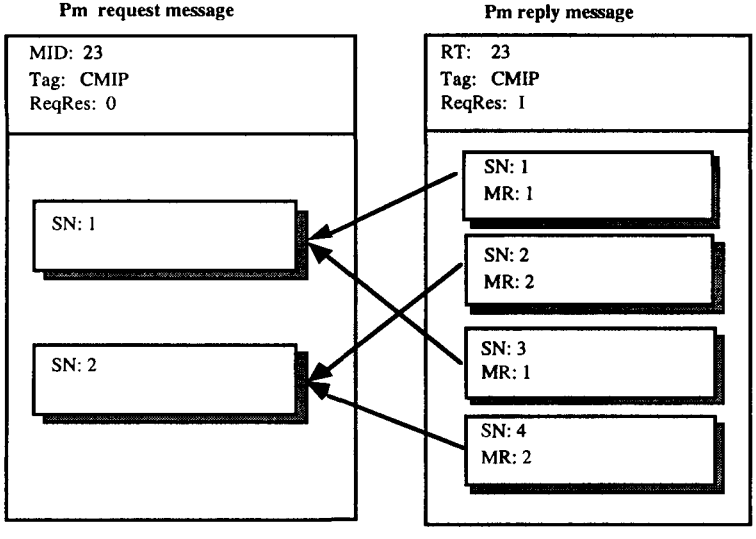

Figure 10 A "Mixed" Pm Reply Message

Next, let's consider the structure of a Pm notification which may be a Pm receipt notification or a Pm non-receipt notification. Unlike a Pm message, a Pm notification is not structured into a Pm-header and a Pm-body. This difference alone is enough to distinguish a Pm notification from a Pm message. Instead, a Pm notification has an envelope and a Pm notification content defined as a sequence of fields. The basic fields are subject-Pm,

originator, and recipient. The subject-Pm field has a value equal to the messageId value of the originating Pm message. A Pm receipt notification has a field, receipt-time, indicating the time of receipt. A Pm non-receipt notification has a field, non-receipt-reason, indicating why the recipient ME is not able to receive the management message (e.g., parsing error or extraction error).

CHAPTER 7

CONCLUSIONS

Currently used service order handling systems used by the telecommunications arena cannot be used to meet successfully the diverse business needs of service providers.

To solve this problem, a generic service order handling model was suggested. Based on the TINA-C information modeling and computation modeling concepts, necessary information object types and attributes, and also interfaces were defined.

The five information objects, the Request For Proposal, order, service, proposal and option object types were defined and the operations and input/output parameters for three interfaces were defined for the service negotiation, service ordering and order tracking for the main service provider and subcontracted service provider respectively.

To support this information and computation models, the Store-and-Forward Paradigm was proposed.

By using the service order handling model suggested, we can realize the provision of the automated and standardized service order handling between service providers. The possible positive effects are:

- By ensuring less manual interventions (e.g, re-typing) and errors, we can improve the quality of service.
- We can achieve the capability for visibility of an ordering process. This is very important for service providers who sub-contract the elements of a customer order to multiple other service providers.
- We can minimize the service provisioning time of service to the end customer.
- We can reduce the order processing cost.
- We can provide an open interface to a service provider's legacy systems and platforms.

The possible extensions to the proposed model would be the customer-service provider service order handling interface, integration with the other customer care processes, and mapping to the real service order handling systems for the improvement of the model.

APPENDIX A

ASN.1 DEFINITIONS

```
- - AcceptanceDate

    AcceptanceDate ::= GeneralizedTime

- - AdditionalInfo

    AdditionalInfo ::= GraphicString

- - AdditionalORStatusInfo

    AdditionalORStatusInfo ::= GraphicString

- - AdditionalRFPStatusInfo

    AdditionalRFPStatusInfo ::= GraphicString

- - AdditionalSRStatusInfo

    AdditionalSRStatusInfo ::= GraphicString

- - BillingInfo

    BillingInfo ::= GraphicString

- - CancelRequestedIndicator

    CancelRequestedIndicator ::= BOOLEAN

- - ChargeCode

    ChargeCode ::= CHOICE{GraphicString, INTEGER}

- - ChargeDate

    ChardgeDate ::= GeneralizedTime

- - CommittedDate

    CommittedDate ::= GeneralizedTime

- - ContractID

    ContractID ::= CHOICE{GraphicString, INTEGER}

- - CustomerID

    CustomerID ::= CHOICE{GraphicString, INTEGER}
```

```
- - Dialog

    Dialog ::= GraphicString(SIZE(0..640))

- - DueDate

    DueDate ::= GeneralizedTime

- - EndUserInfoList

    EndUserInfoList ::= SEQUENCE{
                            numberOfEndUsers   [0]    NumberOfEndUsers,
                            endUserList        [1]    EndUserList}

    - - NumberOfEndUsers

        NumberOfEndUsers ::= INTEGER

    - - EndUserList

        EndUserList ::=  SET OF CHOICE{GraphicString, INTEGER}

- - EnteredDate

    EnteredDate ::= GeneralizedTime

-- ExpectedResponsibility

    ExpectedResponsibility ::= GraphicString

- - ExpiryTime

    ExpiryTime ::= GeneralizedTime

- - FacilityTestDate

    FacilityTestDate ::= GeneralizedTime

- - FulfillmentDate

    FulfillmentDate ::= GeneralizedTime

- - GeneralInfo

    GeneralInfo ::= GraphicString

-- HagglingInfo

    HagglingInfo ::= GraphicString
```

```
- - InitialORID

    InitialORID ::= CHOICE{GraphicString, INTEGER}

- - LocationIdList

    LocationIdList ::= SET OF CHOICE{GraphicString, INTEGER}

- - NumberOfLocations

    NumberOfLocations ::= INTEGER

- - OPID

    OPID ::= CHOICE{GraphicString, INTEGER}

- - OPList

    OPList ::= SET OF CHOICE{GraphicString, ObjectIdentifier, ObjectInstance}

- - OPState

    OPState ::= ENUMERATED{
                    negotiating          (1),
                    negotiation_over     (2),
                    closed               (3)
                    }

- - ORAcceptanceDate

    ORAcceptanceDate ::= GeneralizedTime

- - ORAddList

    ORAddList ::= SET OF GraphicString

- - ORAttrIdList

    ORAttrIdList ::= SET OF GraphicString

- - ORAttrList

    ORAttrList ::= SET OF GraphicString

- - ORClearancePerson

    ORClearancePerson ::= PersonReach

    - - PersonReach

        PersonReach ::= SEQUENCE{
```

```
                number      [0]     PersonNumber DEFAULT "",
                name        [1]     PersonName DEFAULT "",
                phone       [2]     PersonPhone OPTIONAL,
                loc         [3]     PersonLocation OPTIONAL,
                email       [4]     PersonEmail OPTIONAL,
                fax         [5]     PersonFax OPTIONAL,
                respon      [6]     PersonRespon OPTIONAL
                }

- - PersonNumber

    PersonNumber ::= INTEGER

- - PersonName

    PersonName ::= GraphicString(SIZE(0..64))

- - PersonPhone

    PersonPhone ::= GraphicString(SIZE(0..64))

- - PersonLocation

    PersonLocation ::= SEQUENCE{
            address     [0]     PremiseAddress,
            name        [1]     PremiseName}

- - PremiseAddress

    PremiseAddress ::= SEQUENCE{
            civicAddress  GraphicString(SIZE(0..64)),
            city                GraphicString(SIZE(0..64)),
            state               GraphicString(SIZE(0..64)),
            zip                 GraphicString(SIZE(0..64))}

- - PremiseName

    PremiseName ::= GraphicString((SIZE(0..64))

- - PersonEmail

    PersonEmail ::= SET OF IA5String

- - PersonFax

    PersonFax ::= GraphicString(SIZE(0..64))

- - PersonRespon

    PersonRespon ::= GraphicString(SIZE(0..64))
```

```
- - ORCloseOutVerification

    ORCloseOutVerification ::= ENUMERATED{
        verified            (0),
        denied              (1),
        no_action           (2)
        }

- - ORDescription

    ORDescription ::= GraphicString

- - ORDialog

        ORDialog ::= GraphicString

- - OREngineeringPerson

    OREngineeringPerson ::= PersonReach

    - - See ORClearancePerson for the definition of PersonReach.

- - ORID

    ORID ::= CHOICE{GraphicString, INTEGER}

- - ORInstallationPerson

    ORInstallationPerson ::= PersonReach

    - - See ORClearancePerson for the definition of PersonReach.

- - ORName

    ORName   ::= GraphicString

- - ORRemoveList

    ORRemoveList ::= SET OF GraphicString

- - ORReplaceList

    ORReplaceList ::= SET OF GraphicString

- - ORClearancePerson

    ORClearancePerson ::= PersonReach

    - - See ORClearancePerson for the definition of PersonReach.

- - ORCloseOutVerification
```

```
    ORCloseOutVerification ::= ENUMERATED{
        noAction        (0),
        verified        (1),
        denied          (2)}

- - ORCommittedDate

    ORCommittedDate ::= GeneralizedTime

- - ORDescription

    ORDescription ::= GraphicString

- - ORFulfilmentDate

    ORFulfilmentDate ::= GeneralizedTime

- - ORPlacedPerson

ORPlacedPerson ::= PersonReach

    - - See ORClearancePerson for the definition of PersonReach.

- - ORPriority

    ORPriority ::= ENUMERATED{
                        undefined       (0),
                        veryLow         (1),
                        low             (2),
                        normal          (3),
                        high            (4),
                        extremelyHigh   (5)}

- - ORReceivedDate

    ORReceivedTime ::= GeneralizedTime

- - ORRepresentative

    ORRepresentative ::= PersonReach

- - ORRequestor

    ORRequestor ::= PersonReach

    - - See ORClearancePerson for the definition of PersonReach.

- - ORState

    ORState ::= SEQUENCE OF SRState
```

```
    - - SRState

        SRState ::= ENUMERATED{
                        queued          (0),
                        openActive      (1),
                        cleared         (2),
                        closed          (3)
                        }

- - ORStatus

ORStatus ::=  ENUMERATED{
                screening                                   (1),
                testing                                     (2),
                dispatchedIn                                (3),
                dispatchedOut                               (4),
                preassignedOut                              (5),
                bulkDispatchedOut                           (6),
                pendingTest                                 (7),
                pendingDispatch                             (8),
                referVendor                                 (9)
                noAccessOther                               (10),
                startNoAccess                               (11),
                stopNoAccess                                (12),
                orderEscalated                              (13),
                craftDispatched                             (14),
                backOrder                                   (15),
                completedCustNotAdvised                     (16),
                completedCustAdvised                        (17),
                completedAwaitingCustVerification           (18),
                closedOut                                   (19),
                closedOutByCustReq                          (20),
                closedOutCustVerified                       (21),
                closedOutCustDenied                         (22),
                canceledPendingWorkInProgress               (23),
                canceledPendingTestCompletion               (24),
                canceledPendingDispatchCompletion           (25)
                }

- - ORStatusTime

ORStatusTime ::= GeneralizedTime

- - ORStatusWindow

ORStatusWindow ::= TimeInterval

    - - TimeInterval

        TimeInterval ::= SEQUENCE{
```

```
        day             [0]     INTEGER (0..31) DEFAULT 0,
        hour            [1]     INTEGER (0..23) DEFAULT 0,
        minute          [2]     INTEGER (0..59) DEFAULT 0,
        second          [3]     INTEGER (0..59) DEFAULT 0,
        msec            [4]     INTEGER (0..999) DEFAULT 0
        } -- TimeInterval shall be non-zero,

- - ORType

    ORType ::= ENUMERATED{
                    initialOrder            (0),
                    subContractedOrder      (1)}

- - PRAddList

    PRAddList ::= SET OF GraphicString

- - PRAttrIDList

    PRAttrIDList ::= SET OF GraphicString

- - PRAttrList

    PRAttrList ::= SET OF GraphicString

- - PRID

    PRID ::= CHOICE{GraphicString, INTEGER}

- - ProjectCode

    ProjectCode ::= GraphicString

- - ProjectDescription

ProjectDescription ::= GraphicString

- - ProjectManager

    ProjectManager ::= PersonReach

    - - See ORClearancePerson for the definition of PersonReach.

- - ProjectRequirements

    ProjectRequirements ::= GraphicString

- - Proposer

    Proposer ::= PersonReach
```

```
    - - See ORClearancePerson for the definition of PersonReach.

- - PRRemoveList

    PRRemoveList ::= SET OF GraphicString

- - PRReplaceList

    PRReplaceList ::= SET OF GraphicString

- - PRState

    PRState ::= SEQUENCE OF OPState

    - - See the previous definition of OPState.

- - PurchaseTermsAndConditions

    PurchaseTermsAndConditions ::= GraphicString

- - ReadyForServiceDate

    ReadyForServiceDate ::= CHOICE{
        ready-for-service-date        [0]    GeneralizedTime,
        ready-for-service-date-null   [1]    NULL}

- - ReceivedTime

    ReceivedTime ::= GeneralizedTime

- - RelatedORList

    RelatedORList ::= SET OF
                      CHOICE{GraphicString, ObjectIdentifier, ObjectInstance}

- - RelatedPRList

    RelatedPRList ::= SET OF
                      CHOICE{GraphicString, ObjectIdentifier, ObjectInstance}

- - RequestedDate

    RequestedDate ::= GeneralizedTime

- - Requestor

    Requestor ::= PersonReach

    - - See ORClearancePerson for the definition of PersonReach.

- - RequestType
```

```
    RequestType ::= ENUMERATED{
                        newService          (0),
                        disconnect          (1),
                        move                (2),
                        change              (3)}

- - Requirements

    Requirements ::= GraphicString

- - RFPAddList

    RFPAddList ::= SET OF GraphicString

- - RFPAttrIDList

    RFPAttrIDList ::= SET OF GraphicString

- - RFPAttrList

    RFPAttrList ::= SET OF GraphicString

    - - This specifies the attributes of RFP.

- - RFPID

    RFPID ::=  CHOICE{GraphicString, INTEGER}

- - RFPName

    RFPName ::= GraphicString

- - RFPRemoveList

    RFPRemoveList ::= SET OF GraphicString

- - RFPReplaceList

    RFPReplaceList ::= SET OF GraphicString

- -  RFPState

    RFPState ::= ENUMERATED{
                queued          (0),
                openActive      (1),
                cleared         (2),
                closed          (3)
                }

- - RFPStatus
```

```
RFPStatus ::= ENUMERATED{
        pendingForReview                    (1),
        reviewingInProgress                 (2),
        reviewingComplete                   (3),
        proposalPreparing                   (4),
        proposalPreparationDone             (5),
        proposalDispatched                  (6),
        closeOutNoInterest                  (7),
        closeOutProposalCancelled           (8),
        closeOutRFPCancelled                (9),
        closeOutNoResponseToProposal        (10),
        closeOutProposalExpired             (11)
        }
```

- - RFPStatusTime

```
RFPStatusTime ::= GeneralizedTime
```

- - ServiceItemList

```
ServiceItemList ::= SEQUENCE{
                endUserInfoList         [0]   EndUserInfoList,
                requestType             [1]   RequestType,
                serviceType             [2]   ServiceType,
                sRDialog                [3]   SRDialog,
                sRLocationInfoList      [4]   SRLocationInfoList,
                sRName                  [5]   SRName,
                sRPriority              [6]   SRPriority,
                sRSpecificInfoList      [7]   SRSpecificInfoList,
                sRStatusWindow          [8]   SRStatusWindow}
```

- - EndUserInfoList

- - See the previous definition of InfoList.

- - RequestType

- - See the previous definition of RequestType.

- - ServiceType

```
ServiceType ::= ENUMERATED{
                pSTN      (0),
                x25       (1),
                fR        (2),
                others    (3)}
```

- - SRDialog

```
SRDialog ::= GraphicString
```

```
    - - SRLocationInfoList

        SRLocationInfoList ::= SEQUENCE{
                numberOfLocations          [0]    NumberOfLocations,
                locationIdList             [1]    LocationIdList}

    - - SRName

        SRName ::= GraphicString

    - - SRSpecificInfoList

        SRSpecificInfoList ::= GraphicString

    - - SRStatusWindow

        SRStatusWindow ::= TimeInterval

        - - See the previous definition of TimeInterval.

- - SRAddList

    SRAddList ::= SET OF GraphicString

- - SRAttrIdList

    SRAttrIdList ::= SET OF GraphicString

- - SRAttrList

    SRAttrList ::= GraphicString

- - SRClearancePerson

    SRClearancePerson ::= PersonReach

- - SRCloseOutVerification

    SRCloseOutVerification ::= ENUMERATED{
        noAction        (0),
        verified        (1),
        denied          (2)}

- - SRID

    SRID ::= CHOICE{GraphicString, INTEGER}

- - SRList

    SRList ::= SET OF GraphicString
```

```
- - SRPriority

    SRPriority ::= ENUMERATED{
                        undefined       (0),
                        veryLow         (1),
                        low             (2),
                        normal          (3),
                        high            (4),
                        extremelyHigh   (5)}

- - SRRemoveList

    SRRemoveList ::= SET OF GraphicString

- - SRReplaceList

    SRReplaceList ::= SET OF GraphicString

- - SRStatus

    SRStatus ::=  ENUMERATED{
                        installPending          (0),
                        installInProgress       (1),
                        installSuccessful       (2),
                        installFail             (3),
                        serviceTestPending      (4),
                        serviceTestInProgress   (5),
                        serviceTestSuccessful   (6),
                        serviceTestFail         (7),
                        facilityTestPending     (8),
                        facilityTestInProgress  (9),
                        facilityTestSuccessful  (10),
                        facilityTestFail        (11),
                        readyForService         (12),
                        closeOut                (13)
                    }

- - SRStatusTime

    SRStatusTime ::= GeneralizedTime

- - SystemCharacteristics

    SystemCharacteristics ::= GraphicString

- - SystemRequirements

    SystemRequirements ::= GraphicString
```

A.2 Abstract Information Objects in the Management Messaging System

```
-- The definition of the abstract information objects of Management Messaging

MMSInformationObjects {joint-iso-ccitt
        mhs-motis(6) mms(?) modules(?) information-objects(?)}
DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- Prologue
-- Exports everything.

IMPORTS
        -- MMS (Management Messaging System) upper bounds
                ub-auto-forward-comment, ub-free-form-name,
                ub-mm-identifier-suffix, ub-local-mm-identifier,
                ub-subject-field, ub-telephone-number
                --
                FROM MMSUpperBounds {joint-iso-ccitt
                        mhs-motis(6) mms(?) modules(?), upper-bounds(?)}

        -- DTAM
                ProtocolElement
                ---
                FROM dTAM

        -- MTS abstract service
                EncodedInformationTypes, G3FacsimileNonBasicParameters,
                MessageDeliveryTime, ORAddress, ORName,
                OtherMessageDeliveryFields, SupplementaryInformation,
                TeletexNonBasicParameters,
                ----
                FROM MTSAbstractService {joint-iso-ccitt
                        mhs-motis(6) mts(3) modules(0) mts-abstract-service(1)};

        -- Directory Service definitions
                DistinguishedName, RDNSequence
                FROM InformationFramework {joint-iso-ccitt
                        ds(5) modules(1) informationFramework(1)};

        -- CMIP
                FROM CMIP-1 {joint-iso-ccitt
                        ms(9) cmip(1) modules(0) protocol(3)};

        -- SNMPv1 and SNMPv2
                ObjectName, ObjectSyntax, NetworkAddress, IpAddress, TimeTicks
                FROM RFC1155-SMI;
        -- DMI1
```

```
        FROM "Desktop Management Interface Specification - Version 1.0
    April 29, 1994";

Time ::= UTCTime


-- Information object
InformationObject ::=       CHOICE {
                            mm      [0]     MM
                            mn      [1]     MN }

-- MM (Management Message)
MM ::=      SEQUENCE{
            heading         Heading,
            body            Body}

-- Heading
Heading ::= SET{
            this-MM                 [0]   ThisMMField,
            moriginator             [1]   MOriginatorField OPTIONAL,
            authorizing-users       [2]   AuthorizingUsertsField OPTIONAL,
            primary-recipients      [3]   PrimaryRecipientsField OPTIONAL,
            copy-recipients         [4]   CopyRecipientsField OPTIONAL,
            blind-copy-recipients   [5]   BlindCopyRecipientsField OPTIONAL,
            replied-to-MM           [6]   RepliedToMMField OPTIONAL,
            obsoleted-MMs           [7]   ObsoletedMMsField DEFAULT {},
            related-MMs             [8]   RelatedMMsField DEFAULT {},
            subject                 [9]   EXPLICIT SubjectField OPTIONAL,
            expiry-time             [10]  ExpiryTimeField OPTIONAL,
            reply-time              [11]  ReplyTimeField OPTIONAL,
            reply-recipients        [12]  ReplyREcipientsField OPTIONAL,
            importance              [13]  ReplyRecipientsField OPTIONAL,
            sensitivity             [14]  SensitivityField OPTIONAL,
            auto-forwarded          [15]  AutoForwardedField DEFAULT FALSE,
            extensions              [16]  ExtensionsField DEFAULT {},
            tag                     [17]  TagField,
            control                 [18]  ControlField
            }

-- Heading component types
MMIdentifier ::= [APPLICATION 11] SET {
            user                        ORAddress OPTIONAL,
            user-relative-identifier    LocalMMIdentifier }

LocalMMIdentifier ::= PrintableString
            (SIZE (1..ub-local-mm-identifier))

RecipientSpecifier ::= SET {
            ae-title                    [0]   AETitle,
            recipient                   [1]   ORDescriptor,
            reply-flag                  [2]   Flag DEFAULT 0,
```

```
        pm-receipt-notification-flag        [3]    Flag DEFAULT 0,
        pm-non-receipt-notification-flag    [4]    Flag DEFAULT 1 }

Flag ::= CHOICE { INTEGER {
        noset (0),
        set (1) } }

NotificationRequests ::= BIT STRING {
        rn          (0),
        nrn         (1),
        mm-return   (2)}

ORDescriptor ::= SET {
        formal-name         ORName OPTIONAL,
        free-form-name      [0]    FreeFormName OPTIONAL,
        telephone-number    [1]    TelephoneNumber OPTIONAL }

FreeFormName ::= TeletexString (SIZE (0..ub-free-form-name))

TelephoneNumber ::= PrintableString (SIZE (0..ub-telephone-number))


-- This MM heading field
ThisMMField ::= MMIdentifier

-- M Originator heading field
MOriginatorField ::= SEQUENCE {
        ae-title        AETitle,
        ordescriptor    ORDescriptor
        }

AETitle ::= IA5String

--  Authorizing users heading field
AuthorizingUsersField ::= SEQUENCE OF AuthorizingUsersSubfield
AuthorizingUsersSubfield ::= ORDescriptor

-- Primary recipients heading field
PrimaryRecipientsField ::= SEQUENCE OF CopyRecipientsSubfield
CopyRecipientsSubfield ::= RecipientSpecifier

-- Copy recipients heading field
CopyRecipientsField ::= SEQUENCE OF CopyRecipientsSubfield
CopyRecipientsSubfield ::= RecipientSpecifier

-- Blind copy recipients heading field
BlindCopyRecipientsField ::= SEQUENCE OF BlindCopyRecipientsSubfield
BlindCopyRecipientsSubfield ::= RecipientSpecifier

-- Replied-to MM heading field
RepliedToMMField ::= MMIdentifier
```

```
-- Obsoleted MMs heading field
ObsoletedMMsField ::= SEQUENCE OF ObsoletedMMsSubfield
ObsoletedMMsSubfield ::= MMIdentifier

-- Related MMs heading field
FieldRelatedMMsField ::= SEQUENCE OF RelatedMMsField
RelatedMMsSubfield ::= MMIdentifier


-- Subfield heading field
SubjectField ::= TeletexString (SIZE (0..ub-subject-field))

-- Expiry time heading field
ExpiryTimeField ::= Time

-- Reply time heading field
ReplyTimeField ::= Time

-- Reply recipient heading field
ReplyRecipientField ::= SEQUENCE OF ReplyRecipientsSubfield
ReplyRecipientSubfield ::= ORDescriptor

-- Importance heading field
ImportanceField ::= ENUMERATED {
        low             (0),
        normal          (1),
        high            (2)}

-- Sensitivity heading field
SensitivityField ::= ENUMERATED {
        person                  (1),
        private                 (2),
        company-confidential    (3)}

-- Auto-forwarded heading field
AutoForwardedField ::= BOOLEAN

-- Extensions heading field
ExtensionsField ::= SET OF HeadingExtension
HeadingExtension ::= SEQUENCE {
        type    OBJECT IDENTIFIER,
        value   ANY DEFINED BY type DEFAULT NULL NULL}

HEADING-EXTENSION MACRO ::=
BEGIN
                TYPE NOTATION ::= "VALUE" type | empty
                VALUE NOTATION ::= value (VALUE OBJECT IDENTIFIER)
END
```

```
-- Tag heading field
TagField ::= ENUMERATED {
        cmip            "CMIP",
        snmp            "SNMP",
        dmi             "DMI"
        }

-- Control heading field
ControlField ::= CHOICE {
        bit             BitField,
        error           "ERROR"
        }

-- CMIP, SNMP and DMI Operations
OperationField ::=  CHOICE {
        m-get                                   "M-GET",
        m-cancel-get                            "M-CANCEL-GET",
        m-set                                   "M-SET",
        m-action                                "M-ACTION",
        m-create                                "M-CREATE",
        m-delete                                "M-DELETE",
        m-event-report                          "M-EVENT-REPORT",
        get                                     "Get",
        get-next                                "Get-Next",
        get-bulk                                "Get-Bulk",
        set                                     "Set",
        trap                                    "Trap",
        inform                                  "Inform",
        dmi-register-mgmt-cmd                   "Dmi-Register-Mgmt-Cmd",
        dmi-unregister-mgmt-cmd                 "Dmi-Unregister-Mgmt-Cmd",
        dmi-cancel-cmd                          "Dmi-Cancel-Cmd",
        dmi-list-component-cmd                  "Dmi-List-Component-Cmd",
        dmi-list-first-component-cmd            "Dmi-List-First-Component-Cmd",
        dmi-list-next-component-cmd             "Dmi-List-Next-Component-Cmd",
        dmi-list-component-desc-cmd             "Dmi-List-Component-Desc-Cmd",
        dmi-list-group-cmd                      "Dmi-List-Group-Cmd",
        dmi-list-first-group-cmd                "Dmi-List-First-Group-Cmd",
        dmi-list-next-group-cmd                 "Dmi-List-Next-Group-Cmd",
        dmi-list-group-desc-cmd                 "Dmi-List-Group-Desc-Cmd",
        dmi-list-attribute-cmd                  "Dmi-List-Attribute-Cmd",
        dmi-list-first-attribute-cmd            "Dmi-List-First-Attribute-Cmd",
        dmi-list-next-attribute-cmd             "Dmi-List-Next-Attribute-Cmd",
        dmi-list-attribute-desc-cmd             "Dmi-List-Attribute-Desc-Cmd",
        dmi-get-attribute-cmd                   "Dmi-Get-Attribute-Cmd",
        dmi-set-attribute-cmd                   "Dmi-Set-Attribute-Cmd",
        dmi-set-reserve-attribute-cmd           "Dmi-Set-Reserve-Attribute-Cmd",
        dmi-set-release-attribute-cmd           "Dmi-Set-Release-Attribute-Cmd",
        dmi-get-row-cmd                         "Dmi-Get-Row-Cmd",
        dmi-get-first-row-cmd                   "Dmi-Get-First-Row-Cmd",
        dmi-get-next-row-cmd                    "Dmi-Get-Next-Row-Cmd"
        }
```

```
BitField ::= ENUMERATED {
        request         (0),
        response        (1)}

-- Body

Body ::= BodyPart

-- CMIP, SNMP and DMI
BodyPart ::= CHOICE {
                m-action                        [0]     M-Action,
                m-action-response               [1]     M-Action-Response,
                m-cancel-get                    [2]     M-Cancel-Get,
                m-cancel-get-response           [3]     M-Cancel-Get-Response,
                m-create                        [4]     M-Create,
                m-create-response               [5]     M-Create-Response,
                m-delete                        [6]     M-Delete,
                m-delete-response               [7]     M-Delete-Response,
                m-event-report                  [8]     M-Event-Report,
                m-event-report-response         [9]     M-Event-Report-Response,
                m-get                           [10]    M-Get-Type,
                m-get-response                  [11]    M-Get-Response,
                m-set                           [12]    M-Set,
                m-set-response                  [13]    M-Set-Response,
                get-request                     [14]    GetRequest,
                get-next-request                [15]    GetNext,
                get-bulk-request                [16]    GetBulkRequest,
                set-request                     [17]    SetRequest,
                trap                            [18]    Trap,
                inform-request                  [19]    InformRequest,
                response                        [20]    Response,
                dmi-register-mgmt-cmd           [21]    Dmi-Register-Mgmt-Cmd,
                dmi-register-cnf                [22]    Dmi-Register-Cnf,
                dmi-unregister-mgmt-cmd         [23]    Dmi-Unregister-Mgmt-Cmd,
                dmi-cancel-cmd                  [24]    Dmi-Cancel-Cmd,
                dmi-list-component-cmd          [25]    Dmi-List-Component-Cmd,
                dmi-list-first-component-cmd            [26]    Dmi-List-First-Component-Cmd,
                dmi-list-next-component-cmd             [27]    Dmi-List-Next-Component-Cmd,
                dmi-list-component-desc-cmd             [28]    Dmi-List-Component-Desc-Cmd,
                dmi-list-component-cnf          [29]    Dmi-List-Component-Cnf,
                dmi-list-group-cmd              [30]    Dmi-List-Group-Cmd,
                dmi-list-first-group-cmd        [31]    Dmi-List-First-Group-Cmd,
                dmi-list-next-group-cmd         [32]    Dmi-List-Next-Group-Cmd,
                dmi-list-group-cnf              [33]    Dmi-List-Group-Cnf,
                dmi-list-group-desc-cmd         [34]    Dmi-List-Group-Desc-Cmd,
                dmi-list-attribute-cmd          [35]    Dmi-List-Attribute-Cmd,
                dmi-list-first-attribute-cmd    [36]    Dmi-List-First-Attribute-Cmd,
                dmi-list-next-attribute-cmd     [37]    Dmi-List-Next-Attribute-Cmd,
                dmi-list-attribute-cnf          [38]    Dmi-List-Attribute-Cnf,
                dmi-list-attribute-desc-cmd     [39]    Dmi-List-Attribute-Desc-Cmd,
```

```
                dmi-get-attribute-cmd           [40]    Dmi-Get-Attribute-Cmd,
                dmi-get-attribute-cnf           [41]    Dmi-Get-Attribute-Cnf,
                dmi-set-attribute-cmd           [42]    Dmi-Set-Attribute-Cmd,
                dmi-set-reserve-attribute-cmd   [43]    Dmi-Set-Reserve-Attribute-Cmd,
                dmi-set-release-attribute-cmd   [44]    Dmi-Set-Release-Attribute-Cmd,
                dmi-get-row-cmd                 [45]    Dmi-Get-Row-Cmd,
                dmi-get-first-row-cmd           [46]    Dmi-Get-First-Row-Cmd,
                dmi-get-next-row-cmd            [47]    Dmi-Get-Next-Row-Cmd,
                dmi-get-row-cnf                 [48]    Dmi-Get-Row-Cnf
                }
-- CMIP Operations

-- M-Get
M-Get-Type ::= M-Get | M-Get-Type

M-Get ::= SEQUENCE {
        sequenceNumber      [0]     SequenceNumber,
        operation           [1]     OperationField,
        purpose             [2]     Purpose OPTIONAL,
        baseObjectClass     [3]     BaseObjectClass,
        baseObjectInstance  [4]     BaseObjectInstance,
        accessControl       [5]     AccessControl OPTIONAL,
        synchronization     [6]     IMPLICIT CMISSync DEFAULT bestEffort,
        scope               [7]     Scope DEFAULT baseObject,
        filter              [8]     CMISFilter DEFAULT and {},
        attributeIdList     [9]     IMPLICIT SET OF AttributeId OPTIONAL
        }

Purpose ::= IA5String

BeseObjectClass ::= ObjectClass

BaseObjectInstance ::= ObjectInstance

SequenceNumber ::= INTEGER

MatchingRequest ::= SequenceNumber

linkedIdentifier ::= SEQUENCE {
        count       CountField
        moreFlag    MoreFlag}

CountField ::= INTEGER

MoreFlag ::= CHOICE {INTEGER {
        more        (1),
        noMore      (0)} }

M-Get-Response ::= SEQUENCE{
        sequenceNumber          [0]     SequenceNumber,
        operation               [1]     OperationField,
```

```
        matchingRequest             [2]     MatchingRequest,
        linkedIdentifier            [3]     LinkedIdentifier,
        managedObjectClass          [4]     ObjectClass OPTIONAL,
        managedObjectInstance       [5]     ObjectInstance OPTIONAL,
        currentTime                 [6]     IMPLICIT GeneralizedTime OPTIONAL,
        attributeList               [7]     IMPLICIT SET OF Attribute OPTIONAL,
        errors                      [8]     M-Get-Errors
        }

M-Get-Errors ::= INTEGER {
        accessDenied                (2),
        classInstanceConflict       (19),
        complexityLimitation        (20),
        getListError                (7),
        invalidFilter               (4),
        invalidScope                (16),
        noSuchObjectClass           (0),
        noSuchObjectInstance        (1),
        operationCancelled          (23),
        processingFailure           (10),
        syncNotSupported            (3)
        }

AccessControl ::= EXTERNAL

CMISSync ::= ENUMERATED {
        bestEffort      (0),
        atomic          (1)
        }

Scope ::= CHOICE {INTEGER {
        baseObject              (0),
        firstLevelOnly          (1),
        wholeSubtree            (2) },
        individualLevels        [1] IMPLICIT INTEGER,    -- POSITIVE integer
                                indicates the level to be selected
        baseToNthLevel          [2] IMPLICIT INTEGER }   -- POSITIVE integer N
                                indicates that the range of levels
                                                         -- (0 - N) is to be selected

CMISFilter ::= CHOICE {
        item    [8] FilterItem,
        and     [9] IMPLICIT SET OF CMISFilter,
        or      [10] IMPLICIT SET OF CMISFilter,
        not     [11] CMISFilter
        }

FilterItem ::= CHOICE {
        equality                0] IMPLICIT Attribute,
        substrings              [1] IMPLICIT SEQUENCE OF CHOICE {
        initialString   [0] IMPLICIT SEQUENCE {
```

```
                        attributeId     AttributeId,
                        string  ANY DEFINED BY attributeId},
        anyString       [1]  IMPLICIT SEQUENCE {
                        attributeId     AttributeId,
                        string  ANY DEFINED BY attributeId},
        finalString     [2] IMPLICIT SEQUENCE {
                        attributeId     AttributeId,
                        string  ANY DEFINED BY attributeId} },
        graeterOrEqual          [2] IMPLICIT Attribute,      -- asserted value
                                                    >=attribute value
        lessOrEqual             [3] IMPLICIT Attribute,      -- asserted value
                                                    <=attribute value
        present                 [4] AttributeId,
        subsetOf                [5] IMPLICIT Attribute,      -- asserted value is a
                                    subset of attribute value
        supersetOf              [6] IMPLICIT Attribute,      -- asserted value is a
                                    superset of attribute value
        nonNullSetIntersection[7] IMPLICIT Attribute
        }

Attribute ::= SEQUENCE {
        attributeId             AttributeId,
        attributeValue  ANY DEFINED BY attributeId
        }

AttributeId ::= CHOICE {
        globalForm              [0] IMPLICIT OBJECT IDENTIFIER,
        localForm               [1] IMPLICIT INTEGER
        }

ObjectClass ::= CHOICE {
        globalForm              [0] IMPLICIT OBJECT IDENTIFIER,
        localForm               [1] IMPLICIT INTEGER
        }

ObjectInstance ::= CHOICE {
        distinguishedName               [2] IMPLICIT DistinguishedName,
        nonSpecificForm                 [3] IMPLICIT OCTET STRING,
        localDistinguishedName          [4] IMPLICIT RDNSequence
        }

--M-Event-Report
M-Eevent-Report ::=  SEQUENCE{
        sequenceNumber                  [0] SequenceNumber,
        operation                       [1] OperationField,
        mode                            [2] Mode,
        managedObjectClass              ObjectClass,
        managedObjectInstance           ObjectInstance,
        eventTime                       [3] IMPLICIT GeneralizedTime OPTIONAL,
        eventType                       [4] EventTypeId,
        eventInfo                       [5] ANY DEFINED BY eventType OPTIONAL
```

```
        }

EventTypeID ::= CHOICE {
        globalForm                  [0] IMPLICIT OBJECT IDENTIFIER,
        localForm                   [1] IMPLICIT INTEGER
        }

M-Event-Report-Response ::= SEQUENCE{
        sequenceNumber              [0] SequenceNumber,
        operation                   [1] OperationField,
        matchingRequest             [2] MatchingRequest,
        managedObjectClass          ObjectClass OPTIONAL,
        managedObjectInstance       ObjectInstance OPTIONAL,
        currentTime                 [3] IMPLICIT GeneralizedTime OPTIONAL,
        eventReply                  EventReply OPTIONAL,
        errors                      [4] M-Event-Report-Errors
        }

M-Event-Report-Errors ::= INTEGER {
        invalidArgumentValue        (15),
        noSuchArgument              (14),
        noSuchEventType             (13),
        noSuchObjectClass           (0),
        noSuchObjectInstance        (1),
        processingFailure           (10),
        }

EventReply ::= SEQUENCE {
        eventType                   EventTypeId,
        eventReplyInfo              [0] ANY DEFINED BY eventType OPTIONAL
        }

-- M-Action
M-Action ::= SEQUENCE {
        sequenceNumber              [0] SequenceNumber,
        operation                   [1] OperationField,
        purpose                     [2] Purpose OPTIONAL,
        mode                        Mode,
        COMPONENTS OF               BaseManagedObjectId,
        accessControl               [3] AccessControl OPTIONAL,
        synchronization             [4] IMPLICIT CMISSync DEFAULT bestEffort,
        scope                       [5] Scope DEFAULT baseObject,
        filter                      CMISFilter DEFAULT and {},
        actionInfo                  [6] IMPLICIT ActionInfo
        }

M-Action-Response ::= SEQUENCE {
        sequenceNumber              [0] SequenceNumber,
        operation                   [1] OperationField,
        matchingRequest             [2] MatchingRequest,
        linkedIdentifier            [3] LinkedIdentifier,
```

```
        managedObjectClass          ObjectClass OPTIONAL,
        managedObjectInstance       ObjectInstance OPTIONAL,
        currentTime                 [4] IMPLICIT GeneralizedTime OPTIONAL,
        actionReply                 ActionReply OPTIONAL,
        errors                      [5] M-Action-Errors
        }

M-Action-Errors ::= INTEGER {
        accessDenied                (2),
        classInstanceConflict       (19),
        complexityLimitation        (20),
        invalidScope                (16),
        invalidArgumentValue        (15),
        invalidFilter               (4),
        noSuchAction                (9),
        noSuchArgument              (14),
        noSuchObjectClass           (0),
        noSuchObjectInstance        (1),
        processingFailure           (10),
        syncNotSupported            (3)
        }

-- M-Cancel-Get
M-Cancel-Get ::= SEQUENCE {
        sequenceNumber              [0] SequenceNumber,
        operation                   [1] OperationField,
        purpose                     [2] Purpose OPTIONAL
        }

M-Cancel-Get-Response ::= SEQUENCE {
        sequenceNumber              [0] SequenceNumber,
        operation                   [1] OperationField,
        matchingRequest             [2] MatchingRequest,
        errors                      [3] M-Cancel-Get-Errors
        }

M-Cancel-Get-Errors ::= INTEGER {
        mistypedOperation   (21),
        processingFailure   (10),
        }

-- M-Create
M-Create ::= SEQUENCE {
        sequenceNumber                  [0] SequenceNumber,
        operation                       [1] OperationField,
        purpose                         [2] Purpose OPTIONAL,
        managedObjectClass              ObjectClass,
        CHOICE {
                managedObjectInstance   ObjectInstance,
                superiorObjectInstance  [3] ObjectInstance } OPTIONAL,
                accessControl           [4] AccessControl OPTIONAL,
```

```
        referenceObjectInstance     [5] ObjectInstance OPTIONAL,
        attributeList               [6] IMPLICIT SET OF Attribute OPTIONAL
    }

M-Create-Response ::= SEQUENCE {
    sequenceNumber              [0] SequenceNumber,
    operation                   [1] OperationField,
    matchingRequest             [2] MatchingRequest,
    managedObjectClass          ObjectClass OPTIONAL,
    managedObjectInstance       ObjectInstance OPTIONAL,
                                -- shall be returned if omitted from M-
                        Create
    currentTime                 [3] IMPLICIT GeneralizedTime OPTIONAL,
    attributeList               [4] IMPLICIT SET OF Attribute OPTIONAL,
    errors                      [5] M-Create-Errors
    }

M-Create-Errors ::= INTEGER {
    accessDenied                        (2),
    classInstanceConflict               (19),
    duplicateManagedObjectInstance      (11),
    invalidAttributeValue               (6),
    invalidObjectInstance               (17),
    missingAttributeValue               (18),
    noSuchAttribute                     (5),
    noSuchObjectClass                   (0),
    noSuchObjectInstance                (1),
    noSuchReferenceObject               (12),
    processingFailure                   (10),
    }


-- M-Delete
M-Delete ::= SEQUENCE {
    sequenceNumber              [0] SequenceNumber,
    operation                   [1] OperationField,
    purpose                     [2] Purpose OPTIONAL,
    COMPONENTS OF               BaseManagedObjectId,
    accessControl               [3] AccessControl OPTIONAL,
    synchronization             [4] IMPLICIT CMISSync DEFAULT bestEffort,
    scope                       [5] Scope DEFAULT baseObject,
    filter                      CMISFilter DEFAULT and {}
    }

BaseManagedObjectID ::= SEQUENCE {
    baseManagedObjectClass      ObjectClass,
    baseManagedObjectInstance   ObjectInstance
    }

M-Delete-Response ::= SEQUENCE {
    sequenceNumber              [0] SequenceNumber,
```

```
        operation                   [1] OperationField,
        matchingRequest             [2] MatchingRequest,
        linkedIdentifier            [3] LinkedIdentifier,
        managedObjectClass          ObjectClass OPTIONAL,
        managedObjectInstance       ObjectInstance OPTIONAL,
        currentTime                 [4] IMPLICIT GeneralizedTime OPTIONAL,
        errors                      [5] M-Delete-Errors
        }

M-Delete-Errors ::= INTEGER {
        accessDenied                (2),
        classInstanceConflict       (19),
        complexityLimitation        (20),
        invalidFilter               (4),
        invalidScope                (16),
        noSuchObjectClass           (0),
        noSuchObjectInstance        (1),
        processingFailure           (10),
        syncNotSupported            (3)
        }

-- M-Set
M-Set ::= SEQUENCE {
        sequenceNumber              [0] SequenceIdentifier,
        operation                   [1] OperationField,
        purpose                     [2] Purpose OPTIONAL,
        COMPONENTS OF               BaseManagedObjectId,
        accessControl               [3] AccessControl OPTIONAL,
        synchronization             [4] IMPLICIT CMISSync DEFAULT bestEffort,
        scope                       [5] Scope DEFAULT baseObject,
        filter                      CMISFilter DEFAULT and {},
        modificationList            [6] IMPLICIT SET OF SEQUENCE {
        modifyOperator              [7] IMPLICIT ModifyOperator DEFAULT replace,
        attributeId                 AttributeId,
        attributeValue              ANY DEFINED BY attributeId OPTIONAL
                                    -- absent for setToDefault
        } }
ModifyOperator ::= INTEGER {
        replace                     (0),
        addValues                   (1),
        removeValues                (2),
        setToDefault                (3)
        }

M-Set-Response ::= SEQUENCE {
        sequenceNumber              [0] SequenceNumber,
        operation                   [1] OperationField,
        matchingRequest             [2] MatchingRequest,
        linkedIdentifier            [3] LinkedIdentifier,
        managedObjectClass          ObjectClass OPTIONAL,
        managedObjectInstance       ObjectInstance OPTIONAL,
```

```
        currentTime                 [4] IMPLICIT GeneralizedTime OPTIONAL,
        attributeList               [5] IMPLICIT SET OF Attribute OPTIONAL,
        errors                      [6] M-Set-Errors
        }

M-Set-Errors ::= INTEGER {
        accessDenied                (2),
        classInstanceConflict       (19),
        complexityLimitation        (20),
        invalidFilter               (4),
        invalidScope                (16),
        noSuchObjectClass           (0),
        noSuchObjectInstance        (1),
        processingFailure           (10),
        setListError                (8),
        syncNotSupported            (3)
        }


-- SNMP

-- Get (SNMPv1 and SNMPv2)
Get-Request ::= SEQUENCE {
        sequenceNumber      [0]     SequenceNumber,
        operation           [1]     OperationField,
        purpose             [2]     Purpose OPTIONAL,
        version             [3]     Version,
        community           [4]     Community,
        request-id          [5]     Request-id,
        error-status        [6]     "0",
        error-index         [7]     "0",
        variable-bindings   [8]     Variable-bindings
        }

-- GetNext (SNMPv1 and SNMPv2)
Get-Next-Request ::= SEQUENCE {
        sequenceNumber      [0]     SequenceNumber,
        operation           [1]     OperationField,
        purpose             [2]     Purpose OPTIONAL,
        version             [3]     Version,
        community           [4]     Community,
        request-id          [5]     Request-id,
        error-status        [6]     "0",
        error-index         [7]     "0",
        variable-bindings   [8]     Variable-bindings
        }

-- GetBulk (SNMPv2)
Get-Bulk-Request ::=SEQUENCE {
        sequenceNumber      [0]     SequenceNumber,
        operation           [1]     OperationField,
```

```
        purpose             [2]     Purpose OPTIONAL,
        version             [3]     Version,
        community           [4]     Community,
        request-id          [5]     Request-id,
        non-repeaters       [6]     Non-repeaters,
        max-repetitions     [7]     Max-repetitions,
        variable-bindings   [8]     Variable-bindings
        }

-- Set (SNMPv1 and SNMPv2)
Set-Request ::=SEQUENCE {
        sequenceNumber      [0]     SequenceNumber,
        operation           [1]     OperationField,
        purpose             [2]     Purpose OPTIONAL,
        version             [3]     Version,
        community           [4]     Community,
        request-id          [5]     Request-id,
        error-status        [6]     "0",
        error-index         [7]     "0",
        variable-bindings   [8]     Variable-bindings
        }

Trap ::= CHOICE {
                snmpv1-trap   SNMPv1-Trap,
                snmpv2-trap   SNMPv2-Trap},

--Trap (SNMPv1)
SNMPv1-Trap ::=SEQUENCE {
        sequenceNumber      [0]     SequenceNumber,
        operation           [1]     OperationField,
        purpose             [2]     Purpose OPTIONAL,
        version             [3]     Version,
        community           [4]     Community,
        enterprise          [5]     Enterprise,
        agent-addr          [6]     Agent-addr,
        generic-trap        [7]     Generic-trap,
        specific-trap       [8]     Specific-trap,
        time-stamp          [9]     Time-stamp,
        variable-bindings   [10]    Variable-bindings
        }

--Trap (SNMPv2)
SNMPv2-Trap ::=SEQUENCE {
        sequenceNumber      [0]     SequenceNumber,
        operation           [1]     OperationField,
        purpose             [2]     Purpose OPTIONAL,
        version             [3]     Version,
        community           [4]     Community,
        request-id          [5]     Request-id,
        error-status        [6]     "0",
        error-index         [7]     "0",
```

```
        variable-bindings       [8]     Variable-bindings
        }

-- Inform (SNMPv2)
InformRequest ::=SEQUENCE {
        sequenceNumber          [0]     SequenceNumber,
        operation               [1]     OperationField,
        purpose                 [2]     Purpose OPTIONAL,
        version                 [3]     Version,
        community               [4]     Community,
        request-id              [5]     Request-id,
        error-status            [6]     "0",
        error-index             [7]     "0",
        variable-bindings       [8]     Variable-bindings
        }

-- Response (SNMPv1 and SNMPv2)
Response ::= SEQUENCE {
        sequenceNumber          [0]     SequenceNumber,
        operation               [1]     OperationField,
        matchingRequest         [2]     MatchingRequest,
        linkedIdentifier        [3]     LinkedIdentifier,
        version                 [4]     Version,
        community               [5]     Community,
        request-id              [6]     Request-id,
        error-status            [7]     Error-status,
        error-index             [8]     Error-index,
        variable-bindings       [9]     Variable-bindings
        }

Error-Status ::= Error-Status-1 | Error-Status-2

Version ::= INTEGER {version-1 (0))} -- version-1 for this RFC

Community ::= OCTET STRING -- community name

Request-id ::= INTEGER

-- values are sometimes ignored
Variable-bindings ::= VarBindList

-- variable-binding list
VarBindList ::= VarBindList-1 | VarBindList-2

VarBindList-1 ::= SEQUENCE OF VarBind-1

VarBindList-2 ::= SEQUENCE (SIZE (0..max-bindings)) OF VarBind-2

-- variable binding
VarBind-1 ::= SEQUENCE { name ObjectName,
                         value ObjectSyntax }
```

```
VarBind-2 ::=
        SEQUENCE { name ObjectName,
                   CHOICE { value ObjectSyntax,
                            unSpecified NULL,
                            noSuchObject        [0] IMPLICIT NULL,
                   noSuchInstance       [1] IMPLICIT NULL,
                            endOfMibView        [2] IMPLICIT NULL
                            }
                   }

Non-repeaters ::= INTEGER (0..max-bindings)

Max-repetitions ::= INTEGER (0..max-bindings)

Enterprise ::= OBJECT IDENTIFIER        -- type of object generating trap

Agent-addr ::= NetworkAddress           -- address of object generating trap

Generic-trap ::= INTEGER {coldStart         (0),
                   warmStart                (1),
                   linkDown                 (2),
                   linkUp                   (3),
                   authenticationFailure    (4),
                   egpNeighborLoss          (5),
                   enterpriseSpecific       (6)
                   }

Specific-Trap ::= INTEGER   -- specific code, present even if
                            -- generic-trap is not enterpriseSpecific

Time-stamp ::= TimeTicks    -- time elapsed between the last
                            -- (re)initialization of the network

-- sometimes ignored
Error-status ::= Error-status-1 | Error-status-2

Error-status-1 ::= INTEGER {
                noError         (0),
                tooBig          (1),
                noSuchName      (2), -- for proxy compatibility,
                badValue        (3), -- for proxy compatibility
                readOnly        (4), -- for proxy compatibility
                genError        (5)
                }

Error-status-2 ::= INTEGER {
                noError                 (0),
                tooBig                  (1),
                noSuchName              (2), -- for proxy compatibility,
                badValue                (3), -- for proxy compatibility
                readOnly                (4), -- for proxy compatibility
```

```
                genError            (5),
                noAccess            (6),
                wrongType           (7),
                wrongLength         (8),
                wrongEncoding       (9),
                wrongValue          (10),
                noCreation          (11),
                inconsistentValue   (12),
                resourceUnavailable (13),
                commitFailed        (14),
                undoFailed          (15),
                authorizationError  (16),
                notWritable         (17),
                inconsistentName    (18)
                }

-- sometimes ignored
Error-index ::= Error-index-1 | Error-index-2

Error-index-1 ::= INTEGER

Error-index-2 ::= INTEGER (0..max-bindings)

max-bindings INTEGER ::= 2147483647


-- DMI

-- Dmi-Register-Mgmt-Cmd
Dmi-Register-Mgmt-Cmd ::= SEQUENCE {
        sequenceNumber          [0]     SequenceNumber,
        operation               [1]     OperationField,
        purpose                 [2]     Purpose OPTIONAL,
        dmiMgmtCommand          [3]     DmiMgmtCommand,
        pConfirmFunc            [4]     Ptr,
        pIndicationFunc         [5]     Ptr
        }

DmiMgmtCommand ::= SEQUENCE {
        iLevelCheckversions     [0]     INTEGER,
        iCommand                [1]     INTEGER,
        iCmdLen                 [2]     INTEGER,
        iMgmtHandle             [3]     INTEGER,
        iCmdHandle              [4]     INTEGER,
        osLanguage              [5]     Offset,
        oSecurity               [6]     Offset,
        iCnfBufLen              [7]     INTEGER,
        pCnfBuf                 [8]     Ptr,
        iRequestCount           [9]     INTEGER,
        iCnfCount               [10]    INTEGER,
        iStatus                 [11]    INTEGER,
```

```
        dmiCiCommand                [12]    DmiCiCommand
        }

Ptr ::= INTEGER

Offset ::= INTEGER

DmiCiCommand ::= SEQUENCE {
        oCmdListEntry               [0]     Offset,
        iCnfBufLen                  [1]     INTEGER,
        pCnfBuf                     [2]     Ptr,
        pConfirmFunc                [2]     Ptr
        }

Dmi-Register-Cnf ::= SEQUENCE {
        sequenceNumber              [0]     SequenceNumber,
        operation                   [1]     OperationField,
        matchingRequest             [2]     MatchingRequest,
        linkedIdentifier            [3]     LinkedIdentifier,
        dmiVersion                  [4]     DmiVersion,
        iDmiHandle                  [5]     INTEGER,
        dmiConfirm                  [6]     DmiConfirm
        }

DmiConfirm ::= SEQUENCE {
        iLevelCheck                 [0]     INTEGER,
        pDmiMgmtCommand             [1]     Ptr,
        iStatus                     [2]     StatusCode
        }

DmiVersion ::= SEQUENCE {
        osDmiSpecLevel              [0]     Offset,
        osImplDesc                  [1]     Offset
        }

-- Dmi-Unregister-Mgmt-Cmd
Dmi-Unregister-Mgmt-Cmd ::= SEQUENCE {
        sequenceNumber              [0]     SequenceNumber,
        operation                   [1]     OperationField,
        purpose                     [2]     Purpose OPTIONAL,
        iLevelCheckversions         [3]     INTEGER,
        iCommand                    [4]     "0x101",
        iCmdLen                     [5]     INTEGER,
        iMgmtHandle                 [6]     INTEGER,
        iCmdHandle                  [7]     INTEGER,
        osLanguage                  [8]     Offset,
        oSecurity                   [9]     Offset,
        iCnfBufLen                  [10]    INTEGER,
        pCnfBuf                     [11]    Ptr,
        iRequestCount               [12]    INTEGER,
        iCnfCount                   [13]    INTEGER,
```

```
        iStatus                 [14]    INTEGER,
        dmiCiCommand            [15]    DmiCiCommand
        }

-- Dmi-Cancel-Cmd
Dmi-Cancel-Cmd ::= SEQUENCE {
        sequenceNumber          [0]     SequenceNumber,
        operation               [1]     OperationField,
        purpose                 [2]     Purpose OPTIONAL,
        iLevelCheckversions     [3]     INTEGER,
        iCommand                [4]     "0x102",
        iCmdLen                 [5]     INTEGER,
        iMgmtHandle             [6]     INTEGER,
        iCmdHandle              [7]     INTEGER,
        osLanguage              [8]     Offset,
        oSecurity               [9]     Offset,
        iCnfBufLen              [10]    INTEGER,
        pCnfBuf                 [11]    Ptr,
        iRequestCount           [12]    INTEGER,
        iCnfCount               [13]    INTEGER,
        iStatus                 [14]    INTEGER,
        dmiCiCommand            [15]    DmiCiCommand
        }

-- Dmi-List-Component-Cmd
Dmi-List-Component-Cmd ::= SEQUENCE {
        sequenceNumber          [0]     SequenceNumber,
        operation               [1]     OperationField,
        purpose                 [2]     Purpose OPTIONAL,
        dmiMgmtCommand          [3]     DmiMgmtCommand,
        iComponentId            [4]     INTEGER,
        osClassString           [5]     Offset,
        iGroupKeyCount          [6]     INTEGER,
        oGroupKeyList           [7]     Offset
        }

-- Dmi-List-First-Component-Cmd
Dmi-List-First-Component-Cmd ::= SEQUENCE {
        sequenceNumber          [0]     SequenceNumber,
        operation               [1]     OperationField,
        purpose                 [2]     Purpose OPTIONAL,
        dmiMgmtCommand          [3]     DmiMgmtCommand,
        iComponentId            [4]     INTEGER,
        osClassString           [5]     Offset,
        iGroupKeyCount          [6]     INTEGER,
        oGroupKeyList           [7]     Offset
        }

-- Dmi-List-Next-Component-Cmd
Dmi-List-Next-Component-Cmd ::= SEQUENCE {
        sequenceNumber          [0]     SequenceNumber,
```

```
        operation               [1]     OperationField,
        purpose                 [2]     Purpose OPTIONAL,
        dmiMgmtCommand          [3]     DmiMgmtCommand,
        iComponentId            [4]     INTEGER,
        osClassString           [5]     Offset,
        iGroupKeyCount          [6]     INTEGER,
        oGroupKeyList           [7]     Offset
        }

-- Dmi-List-Component-Cnf
Dmi-List-Component-Cnf ::= SEQUENCE {
        sequenceNumber          [0]     SequenceNumber,
        operation               [1]     OperationField,
        matchingRequest         [2]     MatchingRequest,
        linkedIdentifier        [3]     LinkedIdentifier,
        iComponentId            [4]     INTEGER,
        iMatchType              [5]     INTEGER,
        osComponentName         [6]     Offset,
        iClassListCount         [7]     INTEGER,
        oClassNameList          [8]     Offset,
        iFileCount              [9]     INTEGER,
        dmiFileData             [10]    DmiFileData,
        dmiConfirm              [11]    DmiConfirm
        }

DmiFileData ::= SEQUENCE {
        iFileType               [0]     INTEGER,
        oFileData               [1]     Offset
        }

-- Dmi-List-Component-Desc-Cmd
Dmi-List-Component-Desc-Cmd ::= SEQUENCE {
        sequenceNumber          [0]     SequenceNumber,
        operation               [1]     OperationField,
        purpose                 [2]     Purpose OPTIONAL,
        dmiMgmtCommand          [3]     DmiMgmtCommand,
        iComponentId            [4]     INTEGER,
        iGroupId                [5]     INTEGER,
        iAttributeId            [6]     INTEGER,
        iOffset                 [7]     INTEGER
        }

-- Dmi-List-Group-Cmd
Dmi-List-Group-Cmd ::= SEQUENCE {
        sequenceNumber          [0]     SequenceNumber,
        operation               [1]     OperationField,
        purpose                 [2]     Purpose OPTIONAL,
        dmiMgmtCommand          [3]     DmiMgmtCommand,
        iComponentId            [4]     INTEGER,
        iGroupId                [5]     INTEGER
        }
```

```
-- Dmi-List-First-Group-Cmd
Dmi-List-First-Group-Cmd ::= SEQUENCE {
        sequenceNumber          [0]     SequenceNumber,
        operation               [1]     OperationField,
        purpose                 [2]     Purpose OPTIONAL,
        dmiMgmtCommand          [3]     DmiMgmtCommand,
        iComponentId            [4]     INTEGER,
        iGroupId                [5]     INTEGER
        }

-- Dmi-List-Next-Group-Cmd
Dmi-List-Next-Group-Cmd ::= SEQUENCE {
        sequenceNumber          [0]     SequenceNumber,
        operation               [1]     OperationField,
        purpose                 [2]     Purpose OPTIONAL,
        dmiMgmtCommand          [3]     DmiMgmtCommand,
        iComponentId            [4]     INTEGER,
        iGroupId                [5]     INTEGER
        }

-- Dmi-List-Group-Cnf ::= SEQUENCE {
        sequenceNumber          [0]     SequenceNumber,
        operation               [1]     OperationField,
        matchingRequest         [2]     MatchingRequest,
        linkedIdentifier        [3]     LinkedIdentifier,
        iGroupId                [4]     INTEGER,
        osGroupName             [5]     Offset,
        osClassString           [6]     Offset,
        iGroupKeyCount          [7]     INTEGER,
        oGroupKeyList           [8]     Offset,
        dmiConfirm              [9]     DmiConfirm
        }
-- Dmi-List-Group-Desc-Cmd
Dmi-List-Group-Desc-Cmd ::= SEQUENCE {
        sequenceNumber          [0]     SequenceNumber,
        operation               [1]     OperationField,
        purpose                 [2]     Purpose OPTIONAL,
        dmiMgmtCommand          [3]     DmiMgmtCommand,
        iComponentId            [4]     INTEGER,
        iGroupId                [5]     INTEGER,
        iAttributeId            [6]     INTEGER,
        iOffset                 [7]     INTEGER
        }

-- Dmi-List-Attribute-Cmd
Dmi-List-Attribute-Cmd ::= SEQUENCE {
        sequenceNumber          [0]     SequenceNumber,
        operation               [1]     OperationField,
        purpose                 [2]     Purpose OPTIONAL,
        dmiMgmtCommand          [3]     DmiMgmtCommand,
```

```
        iComponentId                [4]     INTEGER,
        iGroupId                    [5]     INTEGER,
        iAttributeId                [6]     INTEGER
        }

-- Dmi-List-First-Attribute-Cmd
Dmi-List-First-Attribute-Cmd ::= SEQUENCE {
        sequenceNumber              [0]     SequenceNumber,
        operation                   [1]     OperationField,
        purpose                     [2]     Purpose OPTIONAL,
        iComponentId                [3]     INTEGER,
        iGroupId                    [4]     INTEGER,
        iAttributeId                [5]     INTEGER
        }

-- Dmi-List-Next-Attribute-Cmd
Dmi-List-Next-Attribute-Cmd ::= SEQUENCE {
        sequenceNumber              [0]     SequenceNumber,
        operation                   [1]     OperationField,
        purpose                     [2]     Purpose OPTIONAL,
        iComponentId                [3]     INTEGER,
        iGroupId                    [4]     INTEGER,
        iAttributeId                [5]     INTEGER
        }

-- Dmi-List-Attribute-Cnf
Dmi-List-Attribute-Cnf ::= SEQUENCE {
        sequenceNumber              [0]     SequenceNumber,
        operation                   [1]     OperationField,
        matchingRequest             [2]     MatchingRequest,
        linkedIdentifier            [3]     LinkedIdentifier,
        iAttributeId                [4]     INTEGER,
        osAttributeName             [5]     Offset,
        iAttributeAccess            [6]     INTEGER,
        iAttributeType              [7]     INTEGER,
        iAttributeMaxSize           [8]     INTEGER,
        iEnumListCount              [9]     INTEGER,
        oEnumList                   [10]    Offset,
        dmiConfirm                  [11]    DmiConfirm
        }

-- Dmi-List-Attribute-Desc-Cmd
Dmi-List-Attribute-Desc-Cmd ::= SEQUENCE {
        sequenceNumber              [0]     SequenceNumber,
        operation                   [1]     OperationField,
        purpose                     [2]     Purpose OPTIONAL,
        dmiMgmtCommand              [3]     DmiMgmtCommand,
        iComponentId                [4]     INTEGER,
        iGroupId                    [5]     INTEGER,
        iAttributeId                [6]     INTEGER,
        iOffset                     [7]     INTEGER
```

```
        }

-- Dmi-Get-Attribute-Cmd
Dmi-Get-Attribute-Cmd ::= SEQUENCE {
        sequenceNumber          [0]     SequenceNumber,
        operation               [1]     OperationField,
        purpose                 [2]     Purpose OPTIONAL,
        dmiMgmtCommand          [3]     DmiMgmtCommand,
        iComponentId            [4]     INTEGER,
        dmiGetAttributeData     [5]     DmiGetAttributeData
        }

DmiGetAttributeData ::= SEQUENCE {
        iGroupId                [0]     INTEGER,
        iGroupKeyCount          [1]     INTEGER,
        oGroupKeyList           [2]     Offset,
        iAttributeId            [3]     INTEGER
        }

-- Dmi-Get-Attribute-Cnf
Dmi-Get-Attribute-Cnf ::= SEQUENCE {
        sequenceNumber          [0]     SequenceNumber,
        operation               [1]     OperationField,
        matchingRequest         [2]     MatchingRequest,
        linkedIdentifier        [3]     LinkedIdentifier,
        iGroupId                [4]     INTEGER,
        iAttributeId            [5]     INTEGER,
        iAttributeType          [6}     INTEGER,
        oAttributeValue         [7]     Offset,
        dmiConfirm              [8]     DmiConfirm
        }

-- Dmi-Set-Attribute-Cmd
Dmi-Set-Attribute-Cmd ::= SEQUENCE {
        sequenceNumber          [0]     SequenceNumber,
        operation               [1]     OperationField,
        purpose                 [2]     Purpose OPTIONAL,
        dmiMgmtCommand          [3]     DmiMgmtCommand,
        iComponentId            [4]     INTEGER,
        dmiSetAttributeData     [5]     DmiSetAttributeData
        }

DmiSetAttributeData ::= SEQUENCE {
        iGroupId                [0]     INTEGER,
        iGroupKeyCount          [1]     INTEGER,
        oGroupKeyList           [2]     Offset,
        iAttributeId            [3]     INTEGER,
        oAttributeValue         [4]     Offset
        }

-- Dmi-Set-Reserve-Attribute-Cmd
```

```
Dmi-Set-Reserve-Attribute-Cmd ::= SEQUENCE {
        sequenceNumber          [0]     SequenceNumber,
        operation               [1]     OperationField,
        purpose                 [2]     Purpose OPTIONAL,
        dmiMgmtCommand          [3]     DmiMgmtCommand,
        iComponentId            [4]     INTEGER,
        dmiSetAttributeData     [5]     DmiSetAttributeData
        }

-- Dmi-Set-Release-Attribute-Cmd
Dmi-Set-Release-Attribute-Cmd ::= SEQUENCE {
        sequenceNumber          [0]     SequenceNumber,
        operation               [1]     OperationField,
        purpose                 [2]     Purpose OPTIONAL,
        dmiMgmtCommand          [3]     DmiMgmtCommand,
        iComponentId            [4]     INTEGER,
        dmiSetAttributeData     [5]     DmiSetAttributeData
        }

-- Dmi-Get-Row-Cmd
Dmi-Get-Row-Cmd ::= SEQUENCE {
        sequenceNumber          [0]     SequenceNumber,
        operation               [1]     OperationField,
        purpose                 [2]     Purpose OPTIONAL,
        dmiMgmtCommand          [3]     DmiMgmtCommand,
        iComponentId            [4]     INTEGER,
        iGroupId                [5]     INTEGER,
        iGroupKeyCount          [6]     INTEGER,
        oGroupKeyList           [7]     Offset,
        iAttributeId            [8]     INTEGER
        }

-- Dmi-Get-First-Row-Cmd
Dmi-Get-First-Row-Cmd ::= SEQUENCE {
        sequenceNumber          [0]     SequenceNumber,
        operation               [1]     OperationField,
        purpose                 [2]     Purpose OPTIONAL,
        dmiMgmtCommand          [3]     DmiMgmtCommand,
        iComponentId            [4]     INTEGER,
        iGroupId                [5]     INTEGER,
        iGroupKeyCount          [6]     INTEGER,
        oGroupKeyList           [7]     Offset,
        iAttributeId            [8]     INTEGER
        }

-- Dmi-Get-Next-Row-Cmd
Dmi-Get-Next-Row-Cmd ::= SEQUENCE {
        sequenceNumber          [0]     SequenceNumber,
        operation               [1]     OperationField,
        purpose                 [2]     Purpose OPTIONAL,
        dmiMgmtCommand          [3]     DmiMgmtCommand,
```

```
        iComponentId                [4]    INTEGER,
        iGroupId                    [5]    INTEGER,
        iGroupKeyCount              [6]    INTEGER,
        oGroupKeyList               [7]    Offset,
        iAttributeId                [8]    INTEGER
        }

-- Dmi-Get-Row-Cnf
Dmi-Get-Row-Cnf ::= SEQUENCE {
        sequenceNumber              [0]    SequenceNumber,
        operation                   [1]    OperationField,
        matchingRequest             [2]    MatchingRequest,
        linkedIdentifier            [3]    LinkedIdentifier,
        iGroupId                    [4]    INTEGER,
        iGroupKeyCount              [5]    INTEGER,
        oGroupKeyList               [6]    Offset,
        iAttributeCount             [7]    INTEGER,
        dmiGetAttributeCnf          [8]    DmiGetAttributeCnf,
        dmiConfirm                  [9]    DmiConfirm
        }

-- StatusCode
StatusCode ::=INTEGER {
        suucess                                 (0x00000),
        moreDataIsAvailable                     (0x00001),
        attributeNotFound                       (0x00100),
        valueExceedsMaximumSize                 (0x00101),
        componentInstrumentationNotFound        (0x00102),
        enumerationError                        (0x00103),
        groupNotFound                           (0x00104),
        illegalKeys                             (0x00105),
        illegalToSet                            (0x00106),
        cannotResolveAttributeFunctionname      (0x00107),
        illegalToGet                            (0x00108),
        noDescription                           (0x00109),
        rowNotFound                             (0x0010a),
        directInterfaceNotRegistered            (0x0010b),
        mIFDatabaseIsCorrupt                    (0x0010c),
        attributeIsNotSupported                 (0x0010d),
        bufferFull                              (0x00200)
        ill-FormedCommand                       (0x00201)
        illegalCommand                          (0x00202)
        illegalhandle                           (0x00203)
        outOfMemory                             (0x00204)
        noConfirmFunction                       (0x00205)
        noResponseBuffer                        (0x00206)
        commandHandleIsAlreadyInUse             (0x00207)
        dMIVersionMismatch                      (0x00208)
        unknownCiRegistry                       (0x00209)
        commandHasBeenCancelled                 (0x0020a)
        insufficientPrivileges                  (0x0020b)
```

```
        noAccessFunctionProvided                (0x0020c)
        oSFileIOError                           (0x0020d)
        couldNotSpawnANewTask                   (0x0020e)
        ill-formedMIF                           (0x0020f)
        invalidFileType                         (0x00210)
        serviceLayerIsInactive                  (0x00211)
        uNICODENotSupported                     (0x00212)
        startOfDOS-specificStatusCodes          (0x01000)
        startOfWindows-specificStatusCodes      (0x02000)
        startOfOS2-specificStatusCodes          (0x03000)
        startOfUNIX-specificStatusCodes         (0x04000)
        startOfMacOS-specificStatusCodes        (0x05000)
        startOfWindowsNT-specificStatusCodes    (0x06000)
        startOfComponent-specificStatusCodes    (0x10000)
        }


-- MN (Management Notification)

MN ::= CHOICE {
        mrn             [0]     MRN,
        mnrn    [1]     MNRN }

MRN ::= MNM -- with receipt-fields chosen

MNRN ::= MNM -- with non-receipt-fields chosen

MNM ::= SET {
        -- common-fields -- COMPONENTS OF CommonFields,
        choice [0] CHOICE {
                        non-receipt-fields      [0] Non-ReceiptFields,
                        receipt-fields          [1] ReceiptFields}}

CommonFields ::= SET {
        subject-mn                              SubjectMNFields,
        mn-originator                   [1]     MNOriginatorField OPTIONAL,
        mn-preferred-recipient          [2]     MNPreferredRecipientField OPTIONAL,
        conversion-eits                         ConversionEITsField OPTIONAL}

NonReceiptFields ::= SET {
        non-receipt-reason              [0]     NonReceiptReasonField,
        discard-reason                  [1]     DiscardReasonField OPTIONAL,
        auto-forward-comment            [2]     AutoForwardCommentField OPTIONAL,
        returned-mn                     [3]     ReturnedMNField OPTIONAL }

ReceiptFields ::= SET {
        receipt-time                    [0]     RecipientTimeField,
        acknowledgement-mode            [1]     AcknowledgementModeField DEFAULT
manual,
        suppl-receipt-info              [2]     SupplReceiptInfoField DEFAULT ""}
```

```
-- Common fields
SubjectMNField ::= MMIdentifier

-- Originator heading field
MNOriginatorField ::= SEQUENCE {
        ae-title        AETitle,
        ordescriptor    ORDescriptor
        }

MNPreferredRecipientField ::= ORDescriptor

ConversionEITsField ::= EncodedInformationTypes

-- Non-receipt fields
NonReceiptReasonField ::= EUNMERATED {
        mm-discarded        (0),
        mm-auto-forwarded   (1)}

DiscardReasonField ::= EUMERATED {
        mm-expired                  (0),
        mm-obsoleted                (1),
        user-subscription-terminated (2)}

AutoForwardCommentField ::= AutoForwardComment

AutoForwardComment ::= PrintableString
        (SIZE (0..ub-auto-forward-comment))

ReturnedMMField ::= MM

-- Receipt fields
ReceiptTimeField ::= Time

AcknowledgementModeField ::= ENUMERATED {
        manual      (0),
        automatic   (1)}

SupplReceiptField ::= SupplementaryInformation

-- Message store realization
ForwardedInfo ::= SET {
        auto-forwarding-comment             [0] AutoForwardComment OPTIONAL,
        cover-note                          [1] IA5TextBodyPart OPTIONAL,
        this-mm-prefix          [2] PrintableString (SIZE (1..ub-mm-identifier-suffix))
                        OPTIONAL}

END -- of MMInformationObjects


-- Definition of upper bounds for the MMS (Management Messaging System)
```

```
MMSUpperBounds {joint-iso-ccitt
                    mhs-motis(6) mms(?) modules(?) upper-bounds(?)}

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- Prologue
-- Exports everything.

IMPORTS -- nothing --;

-- Upper bounds

ub-auto-forward-comment     INTEGER ::= 256

ub-free-form-name           INTEGER ::= 64

ub-mm-identifier-suffix             INTEGER ::= 2

ub-local-mm-identifier              INTEGER ::= 64

ub-subject-field                    INTEGER ::= 128

ub-telephone-number         INTEGER ::= 32

END -- of MMUpperBounds
```

APPENDIX B

PARAMETER DEFINITIONS

B.1 Service Order Handling Inteerfaces: Negotiation, Ordering and Tracking

This appendix gives the definition of the main and supporting types for the interface operation parameters. A number of supporting "opt" types are defined here. An "opt" type is one containing a special value, "pass." It is primarily used in the specification of a structure type in which there are one or more optional fields. For example, consider the definition of PersonReach below:

```
enum  Pass{
          pass
          };

...
union  PersonPhoneOpt  switch (boolean){
    case   TRUE:              Pass          pass;
    case   FALSE:             PersonPhone   personPhone;

};

typedef    GraphicString64     PersonPhone;
...
struct  PersonReach{
    Personname                 name;
    PersonNumber               number;
    PersonPhoneOpt             phone;
    PersonLocationOpt          loc;
    PersonEmailOpt             email;
    PersonFaxOpt               fax;
    PersonResponOpt            respon;
};
```

When the PersonReach type is instantiated, only the name and number fields are required to be specified and the rest are optional. If the phone number of the person is unknown, the TRUE option in the PersonPhoneOpt type is taken and the special value "pass" is assigned to the phone field.

```
/* Supporting Type Definitions */

enum  Pass{
      pass
};

enum  Day{
    1, 2, ..., 31
};

enum  Hour{
    0, 1, ..., 23
};

enum  Minute{
```

```
    0, 1, ..., 59
};

enum  Month{
    jan, mar, apr, may, jun, jul, aug, sep, oct, nov, dec
};

enum  Msec{
    0, 1, ..., 999
};

enum  Second{
    0, 1, ..., 59
};

enum  Year{
    1980, 1991, ..., 2100
};

struct  GeneralizedTime{
    Year        year;
    Month       month;
    Day         day;
    Hour        hour;
    Minute      minute;
    Second      second;
    Msec        msec;
    };

typedef    string <>          GraphicString;

typedef    string <32>        GraphicString32;

typedef    string <64>        GraphicString64;

typedef    string <256>       GraphicString256;

typedef    string <640>       GraphicString640;

typedef    GeneralizedTime          AcceptanceDate;

typedef    GraphicString256         AdditionalInfo;

typedef    GraphicString256         AdditionalORStatusInfo;

typedef    GraphicString256         AdditionalRFPStatusInfo;

typedef    GraphicString256         AdditionalSRStatusInfo;

typedef    GraphicString256         BillingInfo;
```

```
typedef     boolean                 CancelRequestedindicator;

typedef     GraphicString256        ChargeCode;

typedef     GeneralizedTime         ChargeDate;

typedef     GeneralizedTime         CommittedDate;

typedef     GraphicString256        ContractID;

typedef     GraphicString256        CustomerID;

typedef     GraphicString640        Dialog;

typedef     GeneralizedTime         DueDate;

typedef     sequence  <GraphicString256>  EndUserList;

typedef     unsigned  long    NumberOfEndUsers;

struct  EndUserInfo{
        NumberOfEndUsers        numberOfEndUsers;
        EndUserList             endUserList;
};

typedef     sequence  <EndUserInfo>       EndUserInfoList;

typedef     GeneralizedTime         EnteredDate;

typedef     sequence  <GraphicString640>  ExpectedResponsibility;

typedef     GeneralizedTime         ExpiryTime;

typedef     GeneralizedTime         FulfillmentDate;

typedef     sequence  <GraphicString640>  GeneralInfo;

typedef     GraphicString640 HagglingInfo;

typedef     GraphicString64         InitialORID;

typedef     sequence  <GraphicString256>  LocationIdList;

typedef     short                         NumberOfLocations;

enum  OPAttributeID{
    hagglingInfo    (0),
    oPID            (1),
    oPState         (2),
    pRID            (3),
    serviceItemList (4)
```

```
};

typedef     GraphicString256        OPID;

typedef     GraphicString256        PRID;

typedef     OPState{
            negotiating             (0),
            negotiation_over        (1),
            closed                  (2)
            };

union   OPAttributeValue    switch(OPAttributeID){
    case    0:  HagglingInfo        hagglingInfo;
    case    1:  OPID                oPID;
    case    2:  OPState             oPState;
    case    3:  PRID                pRID;
    case    4:  ServiceItemList     serviceItemList;
};

struct      OPAttribute{
    OPAttributeID           oPAttributeID;
    ORAttrValue             oPAttrValue;
};

struct      OPAttributeComp{
    OPAttributeID           oPAttributeID;
    OPAttrCompValue         oPAttrCompValue;
};

typedef     sequence    <GRaphicString640> OPList;

typedef     GeneralizedTime         ORAcceptanceDate;

typedef     GeneralizedTime         FacilityTestDate;

enum        ORAttributeID{
    acceptanceDate              (0),
    additionalORStatusInfo      (1),
    billingInfo                 (2),
    cancelRequestedIndicator    (3),
    chargeDate                  (4),
    chargeCode                  (5),
    committedDate               (6),
    contractID                  (7),
    facilityTestDate            (8),
    fulfillmemtDate             (9),
    initialORID                 (10),
    oPID                        (11),
    oRClearancePerson           (12),
    oRCloseOutVerification      (13),
```

```
    oRDescription                   (14),
    oRDialog                        (15),
    oREngineeringPerson             (16),
    oRID                            (17),
    oRInstallationPerson            (18),
    oRName                          (19),
    oRPriority                      (20),
    oRReceivedTime                  (21),
    oRRepresentative                (22),
    oRRequestor                     (23),
    oRState                         (24),
    oRStatus                        (25),
    oRStatusTime                    (26),
    oRStatusWindow                  (27),
    oRType                          (28),
    projectCode                     (29),
    readyForServiceDate             (30),
    requestedDate                   (31),
    serviceItemList                 (32),
    sRList                          (33)
};

enum    ORStatus{
        screening                               (1),
        testing                                 (2),
        dispatchedIn                            (3),
        dispatchedOut                           (4),
        preassignedOut                          (5),
        bulkDispatchedOut                       (6),
        pendingTest                             (7),
        pendingDispatch                         (8),
        referVendor                             (9),
        noAccessOther                           (10),
        startNoAccess                           (11),
        stopNoAccess                            (12),
        orderEscalated                          (13),
        craftDispatched                         (14),
        backOrder                               (15),
        completedCustNotAdvised                 (16),
        completedCustAdvised                    (17),
        completedAwaitingCustVerification       (18),
        closedOut                               (19),
        closedOutByCustReq                      (20),
        closedOutCustVerified                   (21),
        closedOutCustDenied                     (22),
        canceledPendingWorkInProgress           (23),
        canceledPendingTestCompletion           (24),
        canceledPendingDispatchCompl            (25)
};

typedef     GeneralizedTime   OrderStatusTime;
```

```
typedef     GraphicString256          ProjectCode;

union       ORAttributeValue  switch(ORAttributeID){
    case    0:      AcceptanceDate              acceptanceDate;
    case    1:      AdditionalORStatusInfo      additionalORStatusInfo;
    case    2:      BillingInfo                 BillingInfo;
    case    3:      CancelRequestedIndicator    cancelRequestedIndicator;
    case    4:      ChargeDate                  chargeDate;
    case    5:      ChargeCode                  chargeCode;
    case    6:      CommittedDate               committedDate;
    case    7:      ContractID                  contractID;
    case    8:      FacilityTestDate            facilityTestDate;
    case    9:      FulfillmemtDate             fulfillmemtDate;
    case    10:     InitialORID                 initialORID;
    case    11:     OPID                        oPID;
    case    12:     ORClearancePerson           oRClerancePerson;,
    case    13:     ORCloseOutVerification      oRCloseOutVerification;
    case    14:     ORDescription               oRDescription;
    case    15:     ORDialog                    oRDialog;
    case    16:     OREngineeringPerson         oREngineeringPerson;
    case    17:     ORID                        oRID;
    case    18:     ORInstallationPerson        oRInstallationPerson;
    case    19:     ORName                      oRName;
    case    20:     ORPriority                  oRPriority;
    case    21:     ORReceivedDate              oRReceivedDate;
    case    22:     ORRepresentative            oRRepresentative;
    case    23:     ORRequester                 oRRequester;
    case    24:     ORState                     oRState;
    case    25:     ORStatus                    oRStatus;
    case    26:     ORStatusTime                oRStatusTime;
    case    27:     ORStatusWindow              oRStatusWindow;
    case    28:     ORType                      oRType;
    case    29:     ProjectCode                 projectCode;
    case    30:     ReadyForServiceDate         readyForServiceDate;
    case    31:     RequestedDate               requestedDate;
    case    32:     ServiceItemList             serviceItemList;
    case    33:     SRList                      sRList;
    case    34:     SRName                      sRname;
};

union   ORAttributeCompValue        switch(ORAttributeID){
    case    2:      BillingInfo                 billingInfo;
    case    32:     ServiceItemList     serviceItemList;
};

struct      ORAttribute{
    ORAttributeID       oRAttributeID;
    ORAttrValue         oRAttrValue;
};
```

```
struct      ORAttributeComp{
    ORAttributeID           oRAttributeID;
    ORAttrCompValue         oRAttrCompValue;
};

typedef     sequence    <ORAttributeComp>   ORAttributeCompList;

typedef     sequence    <GraphicString256>  ORAddList;

typedef     sequence    <GraphicString256>  ORAttrList;

typedef     GraphicString64             PersonEail;

union       PersonEmailOpt swithch (boolean) {
    case TRUE:    Pass              pass;
    case FALSE:   PersonEmail       personEmail;
};

typedef     GraphicString64             PersonFax;

union       PersonFaxOpt swithch (boolean) {
    case TRUE:    Pass              pass;
    case FALSE:   PersonFax         personFax;
};

typedef     LocationAddress             PersonLocation;

union       PersonLocationOpt swithch (boolean) {
    case TRUE:    Pass              pass;
    case FALSE:   PersonLocation    personLocation;
};

typedef     GraphicString64             PersonName;

typedef     unsigned Long               PersonNumber;

typedef     GraphicString64             PersonPhone;

union       PersonPhoneOpt swithch (boolean) {
    case TRUE:    Pass              pass;
    case FALSE:   PersonPhone       personPhone;
};

typedef     GraphicString64             PersonRespon;

union   PersonResponOpt swithch (boolean) {
    case TRUE:    Pass              pass;
    case FALSE:   PersonRespon      personRespon;
};

struct   PersonReach{
```

```
    PersonName              name;
    PersonNumber            number;
    PersonPhoneOpt          phoneOpt;
    PersonLocation          locOpt;
    PersonEmailOpt          emailOpt;
    PersonFaxOpt            faxOpt;
    PersonResponOpt         responOpt;
};

union  PersonReachOpt       switch (boolean){
    case TRUE:   Pass               pass;
    case FALSE:  PersonReach        personReach;
};

typedef     PersonReach             ORClearancePerson;

enum        ORCloseOutVerification {
            noAction                (0),
            verified                (1),
            denied          (2)
};

typedef     GraphicString640 ORDescription;

typedef     GraphicString640 ORDialog;

typedef     PerasonReach            OREngineeringPerson;

typedef     GraphicString256 ORID;

typedef     PersonReach             ORInstallationPerson;

typedef     GraphicString64         ORName;

typedef     sequence    <GraphicString256>  ORRemoveList;

typedef     sequence    <GraphicString256>  ORReplaceList;

typedef     GeneralizedTime         ORCommittedDate;

typedef     GeneralizedTime         ORFulfillmentDate;

typedef     PerasonReach            ORPlacedPerson;

enum        ORPriority{
    undefined       (0),
    veryLow         (1),
    low             (2),
    normal          (3),
    high            (4),
    extremelyHigh   (5)
```

```
};

typedef     GeneralizedTime         ORReceivedDate;

typedef     PersonReach             ORRepresentative;

typedef     PersonReach             ORRequestor;

enum  SRState{
    queued          (0),
    openActive      (1),
    cleared         (2),
    closed          (3)
};

typedef     sequence  <SRState>     ORState;

typedef         GeneralizedTime     ORStatusTime;

typedef         long                TimeInterval;

typedef         TimeInterval        ORStatusWindow;

enum        ORType{
            initialOrder            (0),
            subContractedOrder      (1)
};

enum  PRAttributeID{
    expiryTime      (0),
    oPList          (1),
    pRID            (2),
    pRState         (3),
    proposer        (4),
    rFPID           (5)
};

typedef     sequence  <PRID>            RelatedPRList;

struct      ServiceItem{
    EndUserInfoList         endUserInfoList,
    RequestType             requestType,
    ServiceType             serviceType,
    SRDialog                sRDialog,
    SRLocationInfoList      sRLocationInfoList,
    SRName                  sRName,
    SRPriority              sRPriority,
    SRSpecificInfoList      sRSpecificInfoList,
    SRStatusWindow          sRStatusWindow}

typedef     PersonReach     Proposer;
```

```
typedef    GraphicString256 RFPID;

typedef    sequence  <PRAttributeID>      PRAttrIDList;

union  PRAttributeValue      switch(PRAttributeID){
    case    0:  ExpiryTime          expiryTime;
    case    1:  OPList              oPList;
    case    2:  PRID                pRID;
    case    3:  PRState             pRState;
    case    4:  Proposer            proposer;
    case    5:  RFPID               rFPID;
};

union  PRAttributeCompValue      switch(PRAttributeID){
    case    1:  OPList              oPList;
};

struct     PRAttribute{
    PRAttributeID     pRAttributeID;
    PRAttrValue       pRAttrValue;
};

struct     PRAttributeComp{
    PRAttributeID                   pRAttributeID;
    PRAttributeCompValue            pRAttributeCompValue;
};

typedef    GraphicString256 PRAttribute;

typedef    sequence  <PRAttribute>              PRAttrList;

typedef    sequence  <PRAttributeComp>  PRAttributeCompList;

typedef    sequence  <GraphicString640>  ProjectDescription;

typedef    PersonReach              ProjectManager;

typedef    sequence  <GraphicString640>  ProjectRequirements;

enum       sequence  <OPState>      PRState;

typedef    sequence  <GraphicString640>  PurchaseTermsAndConditions;

typedef    GeneralizedTime          ReadyForServiceDate;

typedef    GeneralizedTime          ReceivedTime;

typedef    GraphicString            RelatedOR;

typedef    sequence  <RelatedOR>  RelatedORList;
```

```
typedef     GeneralizedTime          RequestedDate;

typedef     PersonReach              Requestor;

enum        RequestType{
                newService           (0),
                disconnect           (1),
                move                 (2),
                change               (3)
        };

typedef     sequence  <GraphicString640>  Requirements;

enum  RFPAttributeID{
    additionalInfo                   (0),
    cancelRequestedIndicator         (1),
    dueDate                          (2),
    enteredDate                      (3),
    expectedResponsibility           (4),
    generalInfo                      (5),
    projectDescription               (6),
    projectRequirements              (7),
    purchaseTermsAndConditions       (8),
    requestor                        (9),
    rFPID                            (10),
    rFPName                          (11),
    rFPState                         (12),
    rFPStatus                        (13),
    rFPStatusTime                    (14),
};

enum        RFPState{
                queued          (0),
                openActive      (1),
                cleared         (2),
                closed          (3)
};

union  RFPAttributeValue     switch(RFPAttributeID){
    case    0:      AdditionalInfo                  additionalInfo;
    case    1:      CancelRequestedIndicator        cancelRequestedIndicator;
    case    2:      DueDate                         dueDate;
    case    3:      EnteredDate                     enteredDate;
    case    4:      ExpectedResponsibility          expectedResponsibility;
    case    5:      GeneralInfo                     generalInfo;
    case    6:      ProjectDescription              projectDescription;
    case    7:      ProjectRequirements             projectRequirements;
    case    8:      PurchaseTermsAndConditions      purchaseTermsAndConditions;
    case    9:      Requestor                       requestor;
    case    10:     RFPID                           rFPID;
```

```
    case    11:     RFPName                     rFPName;
    case    12:     RFPState                    rFPState;
    case    13:     RFPStatus                   rFPStatus;
    case    14:     RFPStatusTime               rFPStatusTime;
};

struct      RFPAttribute{
    RFPAttributeID              rFPAttributeID;
    RFPAttributeValue           rFPAttributeValue;
};

struct      RFPAttributeComp{
    RFPAttributeID          rFPAttributeID;
    RFPAttrCompValue        rFPAttrCompValue;
};

typedef     sequence  <RFPAttributeComp>        RFPAttributeCompList;

typedef     RFPAttributeCompList    RFPAddList;

typedef     GraphicString64         RFPName;

typedef     RFPAttributeCompList    RFPRemoveList;

typedef     RFPAttrList             RFPReplaceList;

enum            RFPStatus{
            pendingForReview                    (1),
            reviewingInProgress                 (2),
            reviewingComplete                   (3),
            proposalPreparing                   (4),
            proposalPreparationDone             (5),
            proposalDispatched                  (6),
            closeOutNoInterest                  (7),
            closeOutProposalCancelled           (8),
            closeOutRFPCancelled                (9),
            closeOutNoResponseToProposal        (10),
            closeOutProposalExpired             (11)
};

typedef     GeneralizedTime         RFPStatusTime;

struct  ServiceItemList{
                EndUserInfoList             endUserInfoList;
                RequestType                 requestType;
                ServiceType                 serviceType;
                SRDialog                    sRDialog;
                SRLocationInfoList          sRLocationInfoList;
                SRName                      sRName;
                SRPriority                  sRPriority;
                SRSpecificInfoList          SRSpecificInfoList;
```

```
                SRStatusWindow          SRStatusWindow;
};

enum    ServiceType{
            pSTN            (0),
            x25             (1),
            fR              (2),
            others          (3)
};

enum  SRAttributeID{
    additionalSRStatusInfo          (0),
    cancelRequestedIndicator        (1),
    endUserInfoList                 (2),
    oRID                            (3),
    requestType                     (4),
    serviceType                     (5),
    sRDialog                        (6)
    sRID                            (7),
    sRLocationInfoList              (8),
    sRName                          (9),
    sRPriority                      (10),
    sRSpecificInfoList              (11),
    sRState                         (12),
    sRStatus                        (13),
    sRStatusTime                    (14),
    sRStatusWindow                  (15)
};

union  SRAttributeValue     switch(SRAttributeID){
    case  0:    AdditionalSRStatusInfo  additionalSRStatusInfo;
    case  1:    CancelRequestedIndicator cancelRequestedIndicator;
    case  2:    EndUserInfoList         endUserInfoList;
    case  3:    ORID                    oRID;
    case  4:    RequestType             requestType;
    case  5:    ServiceType             serviceType;
    case  6:    SRDialog                sRDialog;
    case  7:    SRID                    sRID;
    case  8:    SRLocationInfoList      sRLocationInfoList;
    case  9:    SRName                  sRName;
    case  10:   SRPriority              sRPriority;
    case  11:   SRSpecificInfoList      sRSpecificInfoList;
    case  12:   SRState                 sRState;
    case  13:   SRStatus                sRStatus;
    case  14:   SRStatusTime            sRStatusTime;
    case  15:   SRStatusWindow          sRStatusWindow;
};

struct      SRAttribute{
    SRAttributeID           SRAttributeID;
    SRAttributeValue        SRAttributeValue;
```

```
};

union  SRAttributeCompValue      switch(SRAttributeID){
    case   0:     AdditionalSRStatusInfo   additionalSRStatusInfo;
    case   2:     EndUserInfoList          endUserInfoList;
    case   6:     SRDialog                 sRDialog;
    case   8:     SRLocationInfoList       sRLocationInfoList;
    case   11:    SRSpecificInfoList       sRSpecificInfoList;
};

struct     SRAttributeComp{
    SRAttributeID     SRAttributeID;
    SRAttrCompValue          SRAttrCompValue;
};

typedef    sequence  <SRAttributeComp>  SRAttributeCompList;

typedef    GraphicString256 SRAttributeID;

typedef    PersonReach              SRClearancePerson;

enum  SRCloseOutVerification{
        noAction          (0),
        verified          (1),
        denied            (2)
};

typedef    GraphicString640 SRDialog;

typedef    GraphicString256         SRID;

typedef          sequence  <GraphicString256>  SRList;

struct     SRLocationInfoList{
           NumberOfLocations        numberOfLocations;
           LocationIdList           locationIdList;
};

typedef    GraphicString64          SRName;

enum       SRPriority{
    undefined         (0),
    veryLow           (1),
    low               (2),
    normal            (3),
    high              (4),
    extremelyHigh     (5)
};

typedef    sequence  <GraphicString256>  SRSpecificInfoList;
```

```
enum  SRStatus{
        installPending              (0),
        installInProgress           (1),
        installSuccessful           (2),
        installFail                 (3),
        serviceTestPending          (4),
        serviceTestInProgress       (5),
        serviceTestSuccessful       (6),
        serviceTestFail             (7),
        facilityTestPending         (8),
        facilityTestInProgress      (9),
        facilityTestSuccessful      (10),
        facilityTestFail            (11),
        readyForService             (12),
        closeOut                    (13)
};

typedef     GeneralizedTime         SRStatusTime;

typedef     TimeInterval            SRStatusWindow;

typedef     GeneralizedTime         StatusTime;

typedef     sequence  <GraphicString640>  SystemCharacteristics;

typedef     sequence  <GraphicString640>  SystemRequirements;

/* Main type declaration */

typedef     ORAttributeCompList     ORAddList;

typedef     sequence  <ORAttributeID>     ORAttrIDList;

typedef     sequence  <ORAttribute>       ORAttrList;

typedef     ORAttributeCompList     ORRemoveList;

typedef     ORAttributeList   ORReplaceList;

typedef     OPAttributeCompList     OPAddlist;

typedef     sequence  <OPAttributeID>     OPAttrIDList;

typedef     sequence  <OPAttribute>       OPAttrList;

typedef     OPAttributeCompList     OPRemoveList;

typedef     OPAttributeCompList     OPReplaceList;

typedef     PRAttributeCompList     PRAddList;
```

```
typedef    sequence   <PRAttributeID>      PRAttrIDList;

typedef    sequence   <PRAttribute>        PRAttrList;

typedef    PRAttributeCompList    PRRemoveList;

typedef    PRAttrList             PRReplaceList;

typedef    RFPAttributeCompList   RFPAddList;

typedef    sequence   <RFPAttributeID>    RFPAttrIDList;

typedef    sequence   <RFPAttribute>      RFPAttrList;

typedef    RFPAttributeCompList   RFPRemoveList;

typedef    RFPAttrList            RFPReplaceList;

typedef    SRAttributeCompList    SRAddList;

typedef    sequence   <SRAttributeID>     SRAttrIDList;

typedef    sequence   <SRAttribute>              SRAttrList;

typedef    SRAttributeCompList    SRRemoveList;

typedef    SRAttrList             SRReplaceList;

exception  AddDenied                    {};
exception  CancellationDenied           {};
exception  ChangeDenied                 {};
exception  NegotiationRejected          {};
exception  RemoveDenied                 {};
exception  ReplaceDenied                {};
exception  UnknownOPID                  {};
exception  UnknownORID                  {};
exception  UnknownPRID                  {};
exception  UnknownRFPID                 {};
exception  UnknownSRID                  {};

/* End of type declaration */
```

REFERENCES

1. A Service Management Business Process Model - Issue 1.0, NMF.

2. CCITT X.790 Information Technology - Open Systems Interconnection - Trouble Management Function, Draft Recommendation 2.0, April 1995.

3. Choi, D., Choi, T., Choi, Y. B., and Tang, A., A Store-and-forward Management Paradigm, The 6th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM '95), Ottawa, Canada, October 16 - 18, 1995.

4. Customer to Service Provider Trouble Management Interface Specification, The SMART TT Meeting, London, OSE Laboratory, Computer Networking Department, Computer Science Telecommunications, University of Missouri-Kansas City, October 30-31, 1995.

5. Desktop Management Interface Specification, Version 1.0, Desktop Management Task Force.

6. Draft new Recommendation X.161 - Definition of Customer Network Management Services for Public Data Networks, December 1994.

7. Draft new Recommendation X.162 - Definition of Management Information for Customer Network Management Service for Public Data Networks to be Used with the CNMc Interface, December 1994.

8. Draft new Recommendation X.163 - Definition of Management Information for Customer Network Management Service for Public Data Networks to be Used with the CNMe Interface, December 1994.

9. ITU Draft Recommendation X.901, Information Technology - Open Systems Interconnection - Basic Reference Model of Open Distributed Processing, Part 1: Overview.

10. ITU Recommendation M.3010, Principles for a Telecommunications Management Network, 1992.

11. ITU Recommendation X.411 (1988) | ISO/IEC 10021-4:1988, Information Processing Systems - Text Communication - Message Oriented Text Interchange System: Message Transfer System: Abstract Service Definition and Procedure.

12. ITU Recommendation X.460 (1995) | ISO/IEC 11588-1:1995, Information Technology - Open Systems Interconnection - MHS Management: Model and Architecture.

13. ITU Recommendation X.462 (1995) | ISO/IEC 11588-3:1995, Information Technology - Open Systems Interconnection - MHS Management: Logging Information.

14. ITU Recommendation X.464 (1993) | ISO/IEC 11588-5:1993, Information Technology - Open Systems Interconnection - MHS Management: Configuration Management Functions.

15. ITU Recommendation X.467 (1995) | ISO/IEC 11588-8:1995, Information Technology - Open Systems Interconnection - MHS Management: MTA Management.

16. ITU Recommendation X.468 (1993) | ISO/IEC 11588-9:1993, Information Technology - Open Systems Interconnection - MHS Management: User Agent Entity.

17. ITU Recommendation X.469 (1993) | ISO/IEC 11588-10:1993, Information Technology - Open Systems Interconnection - MHS Management: Message Store Entity.

18. ITU Recommendation X.500-X.525 - The Directory, Geneva, 1991.

19. ITU Recommendation X.710 (1991) | ISO/IEC 9595:1991, Information Technology - Open Systems Interconnection - Common Management Information Service Definition.

20. ITU Recommendation X.711 (1991) | ISO/IEC 9596-1:1991, Information Technology - Open Systems Interconnection - Common Management Information Protocol.

21. ITU Recommendation X.722 - Information Technology - Open Systems Interconnection - Structure of Management Information: Guidelines for the Definition of Managed Object, Geneva, 1992.

22. ITU Recommendation X.725 - Information Technology - Open Systems Interconnection - Structure of Management Information: General Relationship Model (GRM) Model, Part 7, 1991.

23. ITU-T Recommendation Q.822, Stage 1, Stage 2 and Stage 3 Description for the Q3 Interface - Performance Management, April, 1994.

24. ITU-T Recommendation X.730, Information Technology - Open Systems Interconnection - Systems Management: Object Management Function, 1993.

25. ITU-T Recommendation X.731, Information Technology - Open Systems Interconnection - Systems Management: State Management Function, 1993.

26. ITU-T Recommendation X.733, Information Technology - Open Systems Interconnection - Systems Management: Alarm Reporting Function, 1992.

27. ITU-T Recommendation X.734, Information Technology - Open Systems Interconnection - Systems Management: Event Report Management Function, 1993.

28. ITU-T Recommendation X.735, Information Technology - Open Systems Interconnection - Systems Management: Log Control Function, 1993.

29. ITU-T Recommendation X.738, Information Technology - Open Systems Interconnection - Systems Management - Part 13: Summarization Function.

30. ITU-T Recommendation X.739, Information Technology - Open Systems Interconnection - Systems Management - Part 11: Metric Objects and Attributes.

31. ITU Recommendation X.790 (1995), Information Technology - Open Systems Interconnection - Trouble Management Function.

32. Malhotra, M., and Veeraraghavan, M., Performability-based Quality of Service Metrics for Communication Services, The Fourth International Conference on Computer Communications and Networks (ICCCN '95), pp. 512-528, Las Vegas, Nevada, USA, September 20-23, 1995.

33. Minutes of the Rome Meeting, NM Forum SMART Ordering Team Issue 1, July 10, 1995.

34. Network Management Forum - Customer to Service Provider Trouble Administration Interface: Requirements Specification.

35. Network Management Forum Requirements Capture Team. A Service Management Business Process Model - Issue 1.0. Network Management Forum, October, 1994.

36. Network Management Forum - NM024, Network Management Forum Trouble Management, 1992.

37. Network Performance Survey Results Report, Jeff Paschke, International Network Services, June 1995.

38. Notes from the New Jersey Meeting, NM Forum SMART Ordering Team, Draft Version 0.1, February 2, 1995.

39. Open Networking with OSI, Adrian Tang and Sophia Scoggins, Prentice Hall, 1992.

40. Quality of Service (QOS) in Distributed Hypermedia-Systems, The Fourth International Conference on Computer Communications and Networks (ICCCN '95), pp. 529-534, Las Vegas, Nevada, USA, September 20-23, 1995.

41. Revised Text of Draft Recommendation X.160 (CNMA). - "Architecture for Customer Network Management for Public Data Networks.", December 1994.

42. Service Management in Computing and Telecommunications, Richard Hallws, Artech House, Inc., 1995.

43. Simplifying Quality of Service for Network Providers and for Users, Lightsey Wallace, Hekimian Laboratories, Inc., 1993.

44. SMART Ordering White Paper, NM Forum, Version 1.0, October 1995.

45. SMART Performance Reporting, Issue 0.01, February 27, 1995.

46. SMART Performance Reporting White Paper, Draft 0.04, August 17, 1995.

47. SNMP, SNMPv2, and CMIP - The Practical Guide to Network-Management Standards, William Stallings, Addison Wesley, 1993.

48. Statement of User Requirements for Management of Networked Information Systems, Network Management Forum User Advisory Council, October 1992.

49. Telecommunications Network Management into the 21st Century - Techniques, Standards, Technologies, and Applications, Edited by Salah Aidarous and Thomas Plevyak, IEEE Press, 1995.

50. Terplan, K., Communication Networks Management, PTR Prentice-Hall, 1992.

51. The Generic Electronic Communications Interface Implementation Requirements Specifications for PIC/CARE (Working Draft), May 1995.

52. TINA-C Computational Modelling Concepts, Version 2.0, Document No. TB_A2.HC.012_1.2_94, TINA-C, February 1995.

53. TINA-C Information Modelling Concepts, Version 2.0, Document No. TB_EAC.001_1.2_94, TINA-C, April 3, 1995.

54. TINA-C Object Definition Language (TINA-ODL) MANUAL, Version 1.3, TINA-C, June 20, 1995.

55. TINA-C Overall Concepts and Principles of TINA, Version 1.0, 1995.

56. X.407 Abstract Service Definition Conventions.

At the author's request, the Curriculum Vitae (CV) on pages 187-189 has been redacted from this digitized copy.