MASTER EN SCIENCES INFORMATIQUES

Contribution à l'aide à la conception d'applications pour systèmes ouverts

Les éléments de service d'applications et leur développement dans ISODE 4.0; réalisation d'une application les utilisant

Cayphas, Philippe

Award date:
1989

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix, Namur

Institut d'Informatique

Année académique 1988 - 1989

Contribution à l'aide à la conception d'applications pour systèmes ouverts:

Les éléments de service d'applications et leur développement dans ISODE 4.0;
réalisation d'une application les utilisant.

Ph. Cayphas

Mémoire présenté en vue de de l'obtention du titre de licencié et maître en informatique

Promoteur : Ph. van Bastelaer

ABSTRACT

This final thesis attempts to describe and explain the Application Service Elements of the OSI Model.

Using a program which implements them we have created an application based on remote operations to show their large use in various situations and not only in standardized applications.

RESUME

Ce mémoire a comme objectif de décrire et d'expliquer les éléments de service de la couche application du modèle ISO.

Nous avons utilisé un programme qui implémente ces éléments de service pour le développement d'une application basée sur les opérations à distance. Nous voulons ainsi montrer leur utilité pour un grand nombre d'applications et pas uniquement pour celles qui sont standardisées.

Nous tenons à exprimer notre profonde reconnaissance au Professeur van Bastelaer, qui a accepté d'assurer la direction de ce mémoire, pour le temps et l'attention qu'il nous a consacrés.

Nous adressons également nos plus vifs remerciements à tous les membres de la firme BIM, pour l'accueil chaleureux et l'attention qu'ils nous ont témoignés durant le stage. Que Monsieur Geurts se trouve, ici, tout particulièrement assuré de notre profonde reconnaissance pour ses précieux conseils et son entière disponibilité.

Enfin, nous adressons également nos sentiments de profonde reconnaissance à tous ceux qui ont aidé à mener à bien ce mémoire.

INTRODUCTION

C'est une lapalissade de dire que l'informatique prend une place considérable dans notre société.

La diversité et l'incompatibilité des équipements installés sont cependant hautement regrettables et de gros efforts sont donc accomplis pour tenter d'uniformiser ce monde du traitement de l'information.

Une des motivations est la possibilité d'établissement de communications entre les différents systèmes informatiques et, par là même, la coopération à la réalisation de diverses activités.

Des organismes de standardisation internationaux établissent, à cette fin, des standards de communication.

Parmi ceux-ci, le plus connu et celui qui rassemble la plus grande unanimité est, sans conteste, le modèle de référence de ISO [ISO7498] et [CCITTX200], caractérisé par un très large consensus pour ses innombrables possibilités.

Ce modèle est divisé en couches, chacune prenant à son compte une partie du processus de communication.

Un des objectifs de ce mémoire est de faire l'état de l'art en ce qui concerne le niveau supérieur de ce modèle: la couche application.

D'autre part, il est intéressant de se demander si ces différents protocoles, et en particulier celui qui définit la couche application, sont effectivement utilisables.

Pour répondre à cette question, nous analyserons, à titre d'exemple, le programme ISODE qui réalise un tel développement. Nous analyserons en détail la réalisation de la couche application.

Finalement, nous tenterons de prouver, par la réalisation d'une application, que ISODE, et donc le modèle de référence ISO, facilite grandement le développement de programmes utilisant la communication à distance entre ordinateurs différents. Nous pourrons ainsi affirmer que l'architecture ISO n'est pas seulement prévue pour des applications standardisées telles que le courrier électronique, le transfert de fichier...

Le chapitre 2 de ce mémoire présentera brièvement le modèle de référence, tandis que le chapitre 3 expliquera la couche application.

Ensuite, nous expliquerons comment celle-ci est réalisée dans le programme ISODE (chapitre 4) et nous utiliserons une méthodologie proposée par les concepteurs de ISODE pour le développement d'une application (chapitre 5).

CHAPITRE 2 : LE MODELE ISO

2.1 Introduction au modèle de référence ISO

Le modèle de référence ISO (Interconnexion de Systèmes Ouverts) a pour objectif de permettre à des processus tournant sur des ordinateurs éloignés de communiquer entre eux.

Il est développé par l'Organisation de Standardisation Internationale (OSI). Cet organisme a comme but, dans le domaine de la télécommunication, de définir un ensemble de services basés sur les communications entre ordinateurs de constructeurs différents et se trouvant sur des sites différents; ceux-ci sont définis dans des normes.

A la base de celles-ci se trouve le célèbre modèle de référence. Il est divisé en sept couches (cfr Figure 1), chacune d'entre elles assurant une partie spécifique du traitement de la communication.

2.2 Principe du modèle de référence ISO : quelques définitions

Lorsque nous considérons la seule couche application qui fournit des services de télécommunication, il ne faut pas oublier que six autres couches se trouvent sous celle-ci. Toutes ensembles, elles forment le modèle de référence qui sera utilisé pour la communication des processus par des systèmes finaux.

Un système final est un système d'ordinateurs qui a comme objectif la communication en utilisant le modèle de référence. C'est pourquoi il est souvent appelé système ouvert.

Une entité de couche est une implémentation d'une couche de ISO dans un système final. Sa fonction peut être accomplie par l'échange de message (Protocol Data Unit) avec un autre système final (l'entité paire).

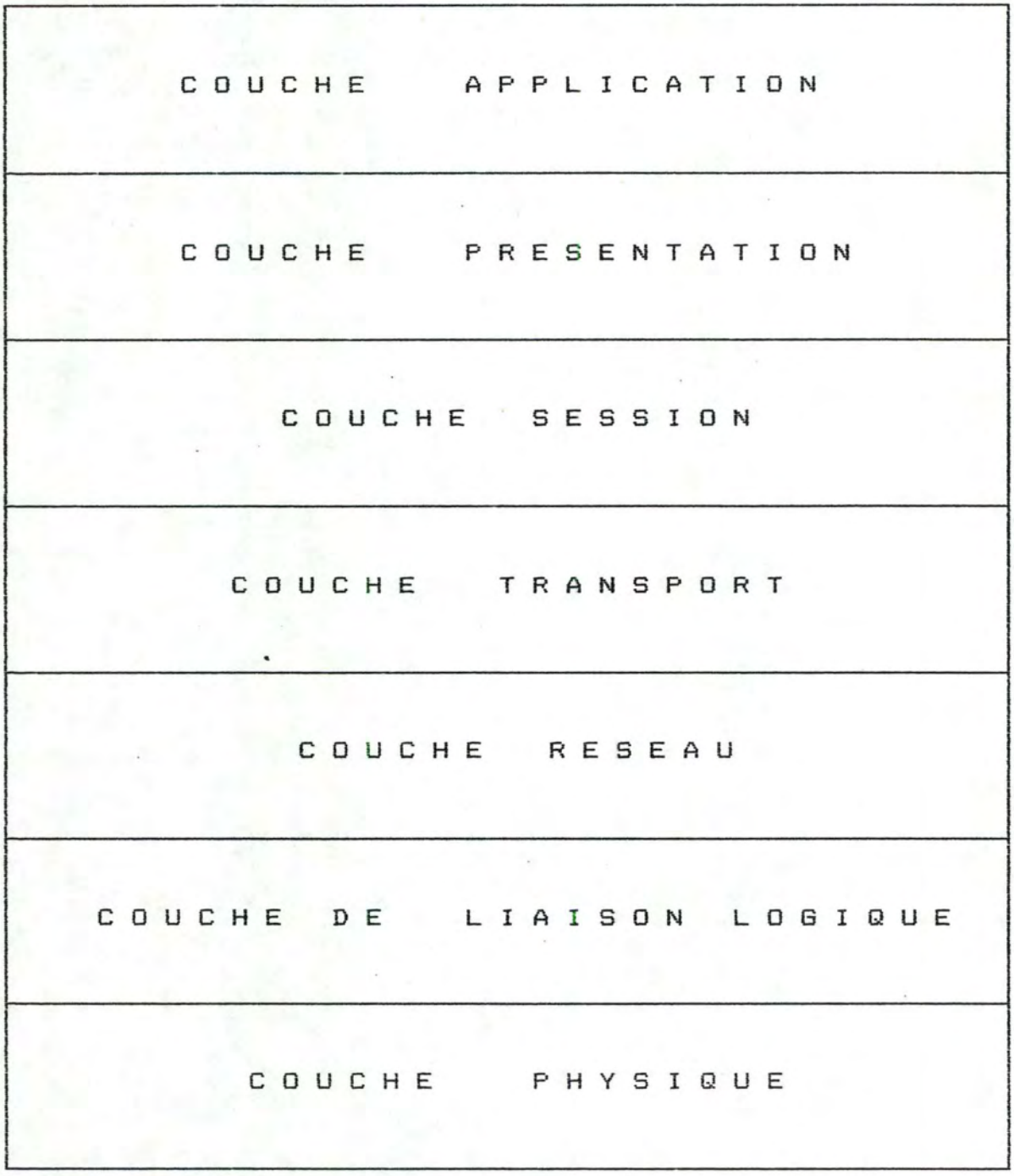

Figure 1 : Le modèle de référence ISO

Une entité offre ses services à ses utilisateurs.

En toute généralité, nous pouvons dire qu'un utilisateur de la couche n (1 <= n <= 7) demande un service à la couche n; excepté au niveau supérieur (n = 7), ce demandeur sera la couche n+1.

La réalisation de ce service sera effectuée par la coopération entre les couches de niveau n des deux systèmes ouverts. Elles l'exécuteront en utilisant le protocole choisi d'un commun accord.

La figure 2 synthétise ces différentes notions.

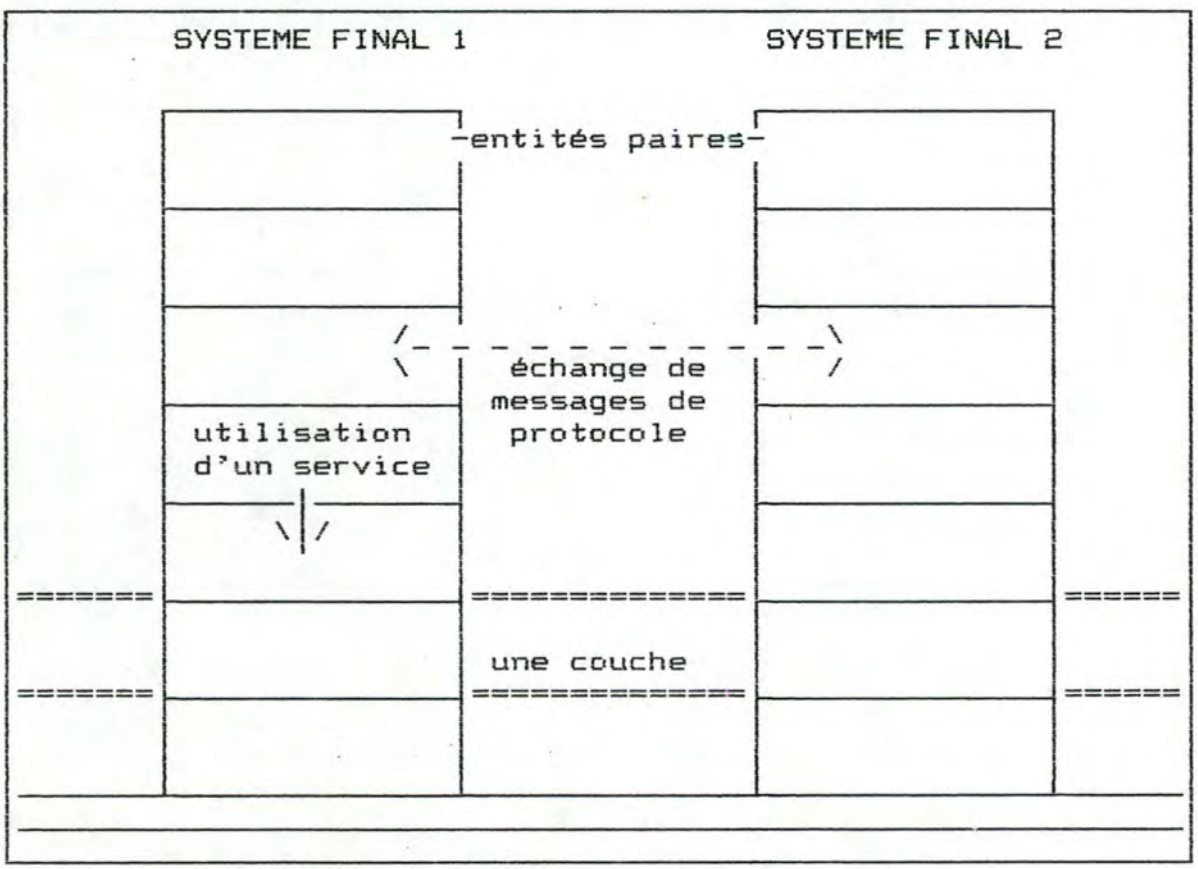

Figure 2 : coopération entre couches

2.3 Les fonctions des différentes couches

2.3.1 La couche application

La couche application est la couche supérieure du modèle de référence. Son utilisateur est soit un processus, soit un être humain.

C'est à ce niveau que se trouvent des programmes utilisant la télécommunication, indépendamment du type de l'ordinateur sur lequel ils se trouvent.

De tels programmes sont, soit des clients locaux du système, soit des serveurs qui permettent l'exécution d'opérations à distance.

Remarquons que cette couche diffère totalement de toutes les autres du modèle ISO puisque ses services sont directement disponibles pour les utilisateurs du système.

Nous précisons les possibilités de ce niveau application au chapitre suivant.

2.3.2 La couche présentation

Deux ordinateurs coopérants à l'exécution d'une tâche commune emploieront plus que probablement des modes de représentation différents pour les données qu'ils échangent. Voilà pourquoi il faut définir une syntaxe d'échange de données commune aux deux systèmes.

Par exemple, considérons, d'une part, un ordinateur utilisant la représentation ASCII pour le codage des caractères et d'autre part, un ordinateur utilisant la représentation EBCDIC. Il faudra, inévitablement, définir un mode de représentation des données compréhensible par les deux systèmes.

La couche présentation veille à résoudre ce genre de problème, en permettant une négociation sur la syntaxe d'échange à utiliser.

2.3.3 La couche session

Nous n'avons, jusqu'à présent, pas encore parlé de communication. C'est la couche session qui est chargée de celle-ci.

Son objectif est de permettre la communication entre deux applications qui veulent effectuer une tâche en commun.

La couche session gère des services tels que : établir une connexion avec une autre entité session, décider qui peut envoyer des données, fermer une connexion, fournir un système de gestion d'activités...

2.3.4 La couche transport

La couche transport fournit une liaison fiable pour la transmission de données à la couche session.

Elle peut combler les faiblesses des couches inférieures grâce à des mécanismes tels que le contrôle de flux et la gestion d'erreurs.

2.3.5 La couche réseau

Le service de base de la couche réseau fournit à la couche transport un transfert de données transparent entre deux entités transport.

Cette couche est notamment responsable du routage entre les différents noeuds du réseau; elle fournit un mécanisme de transfert de paquets.

2.3.6 La couche de liaison logique

La couche de liaison logique tente de rendre fiable la liaison physique sous-jacente. Elle essaye de repérer les erreurs commises et de les corriger. Elle est également responsable du contrôle de flux sur cette liaison physique.

2.3.7 La couche physique

La couche physique transforme les données en signaux qui sont transmis par un support physique (câble coaxial, fibre optique ...).

Le lecteur intéressé par des renseignements plus précis sur le modèle de référence pourra consulter [HESHSHOW] et [STALLINGS].

CHAPITRE 3 : LA COUCHE APPLICATION

3.1 Introduction

La couche application a pour but de supporter les exigences de communication d'applications utilisant plusieurs systèmes ouverts.

Elle repose sur les couches inférieures du modèle ISO; en particulier sur la couche présentation pour la représentation des données à échanger et sur la couche session pour la gestion de la connexion.

Cette couche est totalement différente des autres puisqu'elle est accessible à des utilisateurs finaux directs et non plus par une autre couche.

3.2 Composition

Les entités de la couche application (EA) sont composées d'un ensemble d'éléments de services (ASE). Chacun d'eux est défini dans les standards émis par l'OSI.

Un ASE, est un ensemble de fonctions d'aide à la communication pour l'interconnexion d'applications.

L'ensemble des ASE utilisé définit l'entité application (ensemble spécifique des fonctions d'une application).

Dans les normes antérieures (1984), apparaissait une division supplémentaire que nous voulons signaler : les CASE (3.5) et les SASE (3.4).

Chacun de ces ASE est défini par un ensemble de deux standards.

Le premier spécifie les services offerts, avec notamment, les primitives d'accès. Le second définit le protocole d'échange avec la définition rigoureuse des unités de données de protocole d'application (APDU) utilisées.

3.3 Organisation

Une association se définit comme une liaison de communication entre deux entités applications dans le but de coopérer à une tâche commune en échangeant des informations de contrôle de protocole application grâce à l'utilisation des services de la couche présentation.

Une fonction de contrôle de l'association se trouvera dans l'application. Celle-ci a pour rôle la coordination entre les différents ASE.

Si une seule association est suffisante pour l'application, la fonction de contrôle de simple association (SACF) suffira.

Si plusieurs associations s'avèrent nécessaires, on emploiera la fonction de contrôle de plusieurs associations (MACF) qui gère le tout.

La figure 3 représente une entité association.

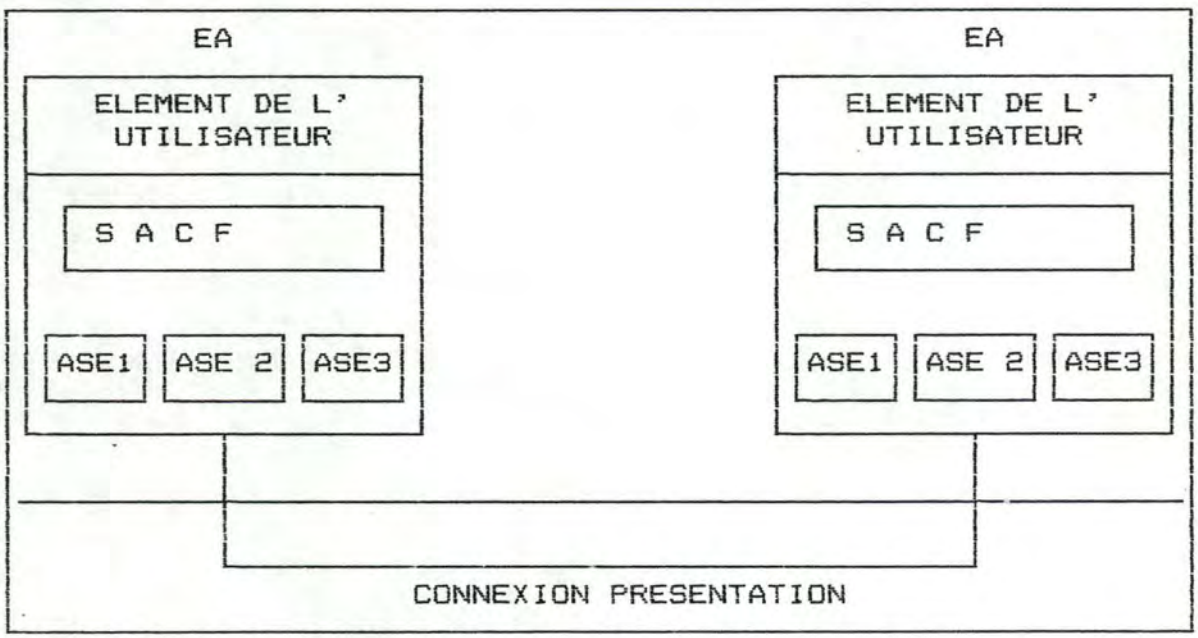

Figure 3 : Exemple d'une entité association

3.4 Les SASE

Les éléments de services spécifiques à une application sont compris dans l'ensemble des SASE; par exemple, le service de gestion de fichier FTAM ou le service de messagerie de MOTIS.

Nous ne parlerons plus de ces SASE, ce terme étant inusité.

3.5 Les CASE

Dans la version de définition de la couche application de 1984, les CASE formaient un ensemble de service commun pour les différentes applications. Cette abréviation a également aujourd'hui totalement disparu.

3.6 Les ASE

Les ASE sont, on peut l'imaginer, des éléments très intéressants lors du développement de nouvelles applications.

En effet, leur utilisation simplifie grandement le travail des programmeurs.

Actuellement, quatre éléments de service sont standardisés: l'ACSE, le RTSE, le ROSE et le CCRSE.

- Le plus ancien est l'élément de service de contrôle d'association (ACSE).

En effet, il est évident que toute application utilisera une connexion. Le choix du contexte de cette application sera négocié lors de l'établissement de l'association.

- Le service RTSE a vu le jour grâce au développement du courrier électronique X400 . Il a pour mission de transférer fiablement des données entre deux applications.

- X400 a également souligné l'utilité d'un service d'exécution d'opération à distance : le ROSE.

Cette découverte semble d'autant plus importante qu'elle pourrait bien devenir le passage obligé de toutes les applications à venir.

- Finalement, on a également défini un service de gestion de transaction pour les systèmes ouverts : le CCRSE.

Nous allons maintenant développer en détail la composition des différents éléments de service brièvement exposés ci-dessus.

Pour chacun d'eux, nous rappellerons l'objectif et définirons les services et le protocole d'échange qu'ils proposent.

3.7 L'ACSE (Association Control Service Element)

3.7.1 Objectif

Lorsqu'un processus application souhaite établir une relation avec un autre processus, il utilise essentiellement une connexion session. Toutefois, il n'est pas possible d'utiliser un ensemble spécifique d'activités propres à une application. Le choix d'un contexte d'application n'est, par exemple, plus possible.

Voilà pourquoi l'association entre entités application a été fournie, pour pouvoir laisser aux processus application un choix quant aux différentes options d'une même application.

exemple

Deux applications peuvent supporter plusieurs syntaxes abstraites. Quelle sera celle qui sera choisie?
Ceci sera défini lors de la création de l'association, de la connexion de niveau 7.

3.7.2 Définition du service

Le service ACSE se répartit en trois fonctions :

- l'établissement d'une association
- la fermeture (négociée) d'une association
- l'abandon (non-négocie) d'une association

Il est clair que ce service est utile à toutes les applications. Son emploi est donc obligatoire. Il faut inévitablement passer par lui pour ouvrir une connexion session.

Ce service est défini dans [ISO8649].

3.7.2.1 Etablissement d'une association

La primitive associée est le A-ASSOCIATE. C'est une primitive de service confirmé, c'est-à-dire en quatre temps (Request, Indication, Response, Confirmation). Elle est représentée à la figure 4.

Dans beaucoup d'applications standardisées, on laisse notamment le choix des différents éléments de services que l'on souhaite employer. Ils seront précisés dans le paramètre contexte application.

Ce contexte est le paramètre essentiel de la primitive A-ASSOCIATE. Il peut être négocié lors de la création de l'association et à ce moment uniquement. C'est, en quelque sorte, le choix des options proposées dans le protocole application que l'on utilisera.

Nous pouvons, dès lors, distinguer deux groupes dans les paramètres de cette primitive.

Le premier, celui qui nous intéresse ici plus directement, reprend les paramètres qui ont une influence directe sur la nature de l'association au niveau 7.

Le second reprend l'ensemble des paramètres qui seront transmis tels quels aux couches présentation et session.

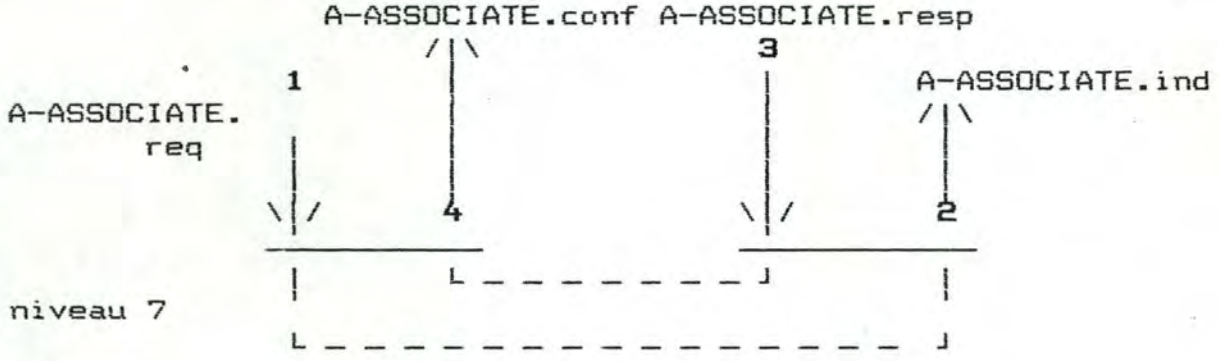

Figure 4 : La primitive A-ASSOCIATE

<u>les paramètres et leur signification</u>

Nous n'envisagerons que les paramètres du premier groupe, liés à la notion d'association.

. Le mode

- identifie la version du protocole en vigueur : normal ou X410 de 1984.
- est un paramètre optionnel pour l'utilisateur, mais obligatoire pour le fournisseur du service.
- apparaît dans le Request (optionnel) et dans le Indication (obligatoire).

. Le qualificatif de l'entité application appelée

- identifie l'entité avec laquelle l'association doit être établie.
- est un paramètre optionnel. S'il n'est pas fourni, c'est l'adresse présentation qui est considérée comme identificateur.
- apparaît dans les primitives request et indication.

. Le qualificatif de l'entité application appelante

- identifie l'entité demandant l'association.
- est un paramètre optionnel.
- apparaît dans les primitives request et indication.

. Le qualificatif de l'entité application répondante

- identifie l'entité répondant à la demande de l'association.
- est un paramètre optionnel.
- apparaît dans les primitives response et confirmation.

. Le titre du processus application (PA) appelé

- identifie le processus avec lequel l'association doit être établie.
- est un paramètre optionnel. S'il n'est pas fourni, c'est l'adresse présentation qui est considérée comme identificateur.
- apparaît dans les primitives request et indication.

. Le titre du processus application appelant

- identifie le processus demandant l'association.
- est un paramètre optionnel.
- apparaît dans les primitives request et indication.

. Le titre du processus application répondant

- identifie le processus répondant à la demande de l'association.
- est un paramètre optionnel.
- apparaît dans les primitives response et confirmation.

. L'identificateur d'invocation de l'EA appelée

- identifie l'invocation de EA avec laquelle

l'association doit être établie.
- est un paramètre optionnel. S'il n'est pas fourni, c'est l'adresse présentation qui est considérée comme identificateur.
- apparaît dans les primitives request et indication.

. L'identificateur d'invocation de l'EA appelante

- identifie l'invocation de EA demandant l'association.
- est un paramètre optionnel.
- apparaît dans les primitives request et indication.

. L'identificateur d'invocation de l'EA répondante

- identifie l'invocation de l'EA répondant à la demande de l'association.
- est un paramètre optionnel.
- apparaît dans les primitives response et confirmation.

. L'identificateur d'invocation du PA appelé

- identifie l'invocation du processus avec lequel l'association doit être établie.
- est un paramètre optionnel. S'il n'est pas fourni, c'est l'adresse présentation qui est considérée comme identificateur.
- apparaît dans les primitives request et indication.

. L'identificateur d'invocation du PA appelant

- identifie l'invocation du processus demandant l'association.
- est un paramètre optionnel.
- apparaît dans les primitives request et indication.

. L'identificateur d'invocation du PA répondant

- identifie le processus répondant à la demande de l'association.
- est un paramètre optionnel
- apparaît dans les primitives response et confirmation.

. Le nom du contexte application

- identifie le nom contexte application choisi par le processus application.
- est un paramètre optionnel. Un paramètre par

défaut est défini dans le protocole en vigueur.
- apparaît dans les primitives request et indication et peut réapparaître (modifié ou non) dans les primitives response et confirmation. C'est le mécanisme de négociation de contexte application.

. Les données de l'utilisateur

- est un paramètre laissé libre pour l'application. Elle peut y mettre ce qu'elle veut selon la convention du protocole utilisé.
- est un paramètre optionnel.
- apparaît dans les quatre primitives.

. Le diagnostic

- indique l'acceptation ou le refus de l'association de l'entité appelée.
- est un paramètre obligatoire.
- apparaît dans les primitives response et confirmation avec une des significations suivantes :
  + accepté
  + contexte application non acceptable
  + titre de l'entité application appelée inconnu
  + titre de l'entité application appelante inconnu
  + temporairement indisponible
  + indisponible
  + données de l'utilisateur illisibles

  Si le répondeur accepte une association en modifiant le nom de contexte de l'application avec une valeur non acceptable pour le demandeur, ce dernier peut interrompre l'association en utilisant la primitive A-ABORT (cfr Figure 5).

. La source du résultat

- indique de quel service provient le résultat.
- paramètre facultatif.

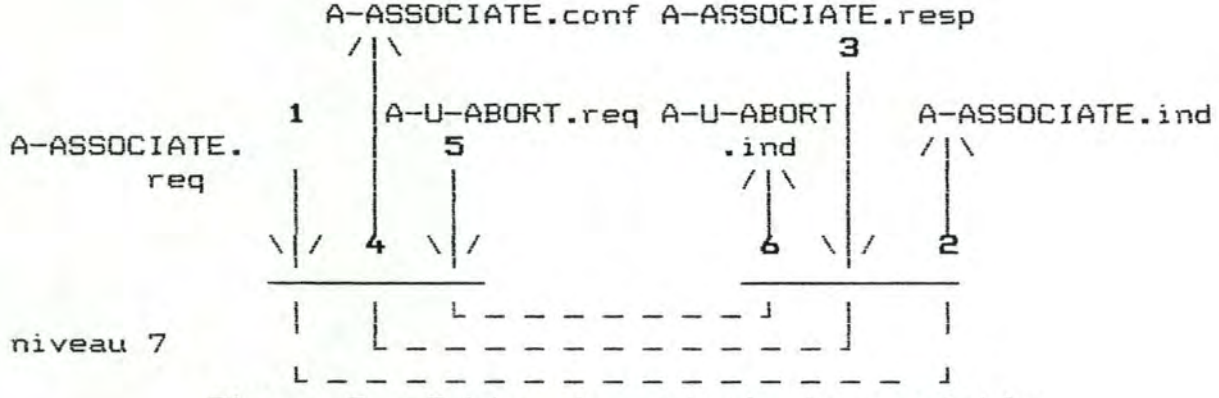

Figure 5 : Schéma de contexte inacceptable

3.7.2.2 Fermeture (négociée) d'une association

La primitive associée, représentée à la figure 6, est le A-RELEASE. C'est une primitive de service confirmé, ce qui veut dire, en l'occurrence, que la fermeture est négociable.

Lorsqu'une entité a terminé son travail, elle signale la fin de l'association en utilisant le A-RELEASE.Request. L'entité paire peut soit accepter, soit refuser cette fermeture en renvoyant un A-RELEASE.Response.

Cette possibilité de refus n'est envisageable que si le jeton d'abandon négocié est utilisé dans la connexion session sous-jacente.

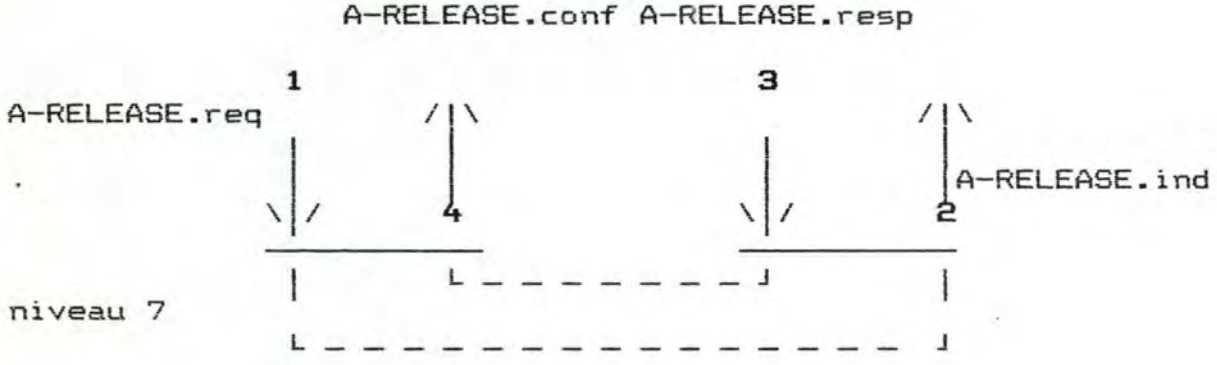

Figure 6 : La primitive A-RELEASE

les paramètres et leur signification

. La raison

- identifie la raison de la demande de fermeture.
- est un paramètre optionnel.
- apparaît dans les quatre phases.

. Les données de l'utilisateur

- sont les données finales définies dans le protocole.
- est un paramètre optionnel.
- apparaît dans les quatre phases.

. Le résultat

- de valeur affirmative ou négative, ce paramètre donne le résultat de fin d'association.

- est un paramètre obligatoire.
- apparaît dans les primitives response et confirmation.

3.7.2.3 Abandon (non négocié) de l'association

La primitive de ce service est le A-U-ABORT. C'est un service non confirmé ; il stoppe immédiatement la connexion entre les deux entités paires; toutes les données en cours sont perdues.

Cette primitive est représentée à la figure 7.

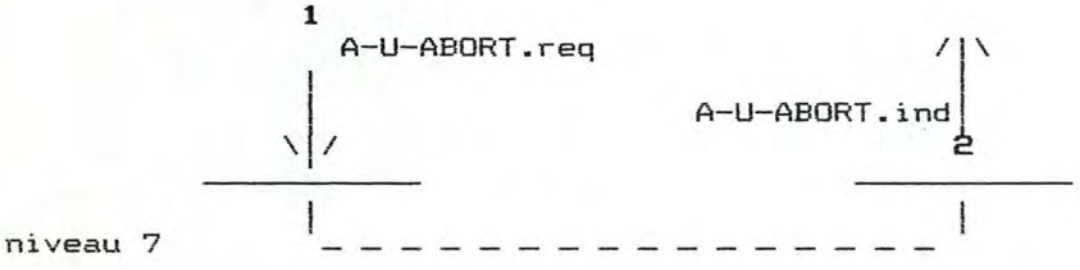

Figure 7 : la primitive A-U-ABORT

Le paramètre associé et sa signification

. Les données de l'utilisateur

- paramètre réservé à l'application utilisatrice. Son contenu est spécifié dans le protocole utilisé.
- paramètre optionnel
- apparaît dans les deux temps.

. La source

- indique la source de l'abandon : le fournisseur ACSE ou le demandeur du service.
- paramètre obligatoire
- apparaît dans le indication

3.7.2.4 Fermeture du fournisseur de l'application

La primitive A-P-ABORT (représentée à la figure 8) est utilisée par le service ACSE qui a un problème. Son résultat

est une déconnexion immédiate et la perte de toutes les données en cours de route.

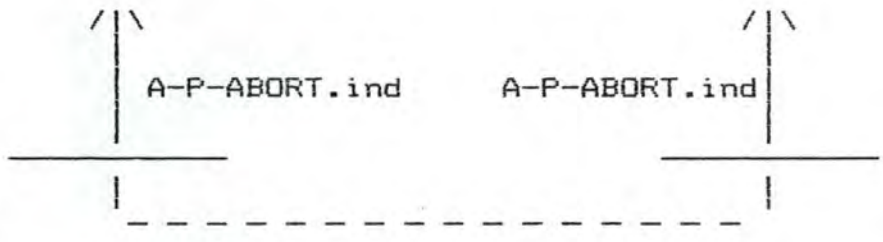

Figure 8 : La primitive A-P-ABORT

3.7.3 Définition du protocole

Le service offert par l'ACSE est l'établissement d'association. Il existe une correspondance directe entre l'association d'une application et la connexion de la couche présentation.

Le protocole realisant ce service est spécifié dans [ISO8650].

Les procédures de connexion et de déconnexion entre applications sont les suivantes :

- L'exécution de A-ASSOCIATE.Req crée un élément de donnée de protocole (APDU) AARQ. Celui-ci est intégré dans le paramètre de données de l'utilisateur de la primitive P-CONNECT.Req qui est activée. L'application paire recevra cet APDU dans le paramètre correspondant du P-CONNECT.Ind.

- L'entité répondante exécute alors un A-ASSOCIATE.Resp et crée ainsi un APDU AARE. Celui-ci sera également transmis via le paramètre de données de l'utilisateur de la primitive P-CONNECT.Resp.

- Les APDU correspondant aux A-RELEASE.req et A-RELEASE.Resp sont respectivement les RLRQ et RLRE. Ils sont transmis dans le paramètre de l'utilisateur des P-RELEASE correspondants.

- La déconnexion non négociée se fait, quant à elle, par l'envoi d'un APDU ABRT.

3.7.3.1 L'AARQ (A-ASSOCIATE ReQuest)

Les paramètres sont dans l'ordre :

- la version du protocole ACSE employé
- le nom de contexte application
- le titre du processus application appelé
- le titre de l'entité application appelée
- l'identificateur d'invocation du processus application appelé
- l'identificateur d'invocation de l'entité application appelée
- le titre du processus application appelant
- le titre de l'entité application appelante
- l'identificateur d'invocation du processus application appelante
- l'identificateur d'invocation de l'entité application appelante
- une information sur l'implémentation
- les données de l'utilisateur

3.7.3.2 L'AARE (A-ASSOCIATE REsponse)

Les paramètres sont dans l'ordre :

- la version du protocole employé
- le nom de contexte application employé
- le résultat
- la raison
- le titre du processus application répondant
- le titre de l'entité application répondante
- l'identificateur d'invocation du processus application appelant
- l'identificateur d'invocation de l'entité application appelante
- une information sur l'implémentation
- les données de l'utilisateur

3.7.3.3 Le RLRQ (ReLease ReQuest)

Les paramètres sont dans l'ordre :

- la raison
- les données de l'utilisateur

3.7.3.4 Le RLRE (ReLease REsponse)

Les paramètres sont dans l'ordre :

- la raison
- les données de l'utilisateur

3.7.3.5 Le ABRT (ABoRT)

Les paramètres sont dans l'ordre :

- la source
- les données de l'utilisateur

3.8 Le RTSE (Reliable Transfer Service Element)

3.8.1 Objectif

Pour une application, il peut être très important de savoir que le service de transfert qu'elle utilise pour ses APDU est d'une sécurité maximale.

L'objectif du RTSE est précisément de fournir un tel service, celui du transfert fiable de données en général et d'APDU en particulier.

exemple

Lors de la transmission d'une image, une erreur n'est pas grave et il ne faut pas perdre de temps à redemander une transmission.
Mais pour un fichier des comptes d'une banque...

3.8.2 Définition du service

Les services du RTSE se répartissent en 6 fonctions :

- l'ouverture d'une liaison de transfert fiable
- la fermeture d'une liaison de transfert fiable
- l'abandon d'une liaison de transfert fiable
- le transfert de données par une entité de façon fiable
- la demande de permission de transmettre
- l'accord de permission de transmettre

Ce service est évidemment optionnel; c'est-à-dire que son utilisation est laissée totalement au choix de l'application. Cet ASE n'est donc pas obligatoire, comme l'était l'ACSE.

Il est défini dans [CCITTX218] et [ISO9066-1].

3.8.2.1 Ouverture d'une liaison de transfert fiable

La primitive associée est le RT-OPEN. C'est une primitive de service confirmé (en quatre temps).

La principale action de cette primitive de service est de signaler le mode de dialogue de la liaison.

En effet, une liaison de transfert fiable pourra être soit sous forme de monologue (une seule entité transmet à l'autre des données), soit sous forme de dialogue (chacune des deux entités peut transmettre des données, mais chacune à son tour); c'est cette forme qui est à signaler lors de l'ouverture de la liaison.

Cette primitive, représentée à la figure 9, contient un certain nombre de paramètres que nous pouvons répartir en deux groupes. D'une part, nous trouvons les paramètres servant à l'ouverture de la liaison fiable et d'autre part il y ceux qui sont directement transmis à la primitive d'ouverture du service ACSE.

Remarquons que dans ce deuxième groupe se trouvent des paramètres qui seront utiles au service présentation et pas au service ACSE. Celui-ci se chargera de simplement les transmettre.

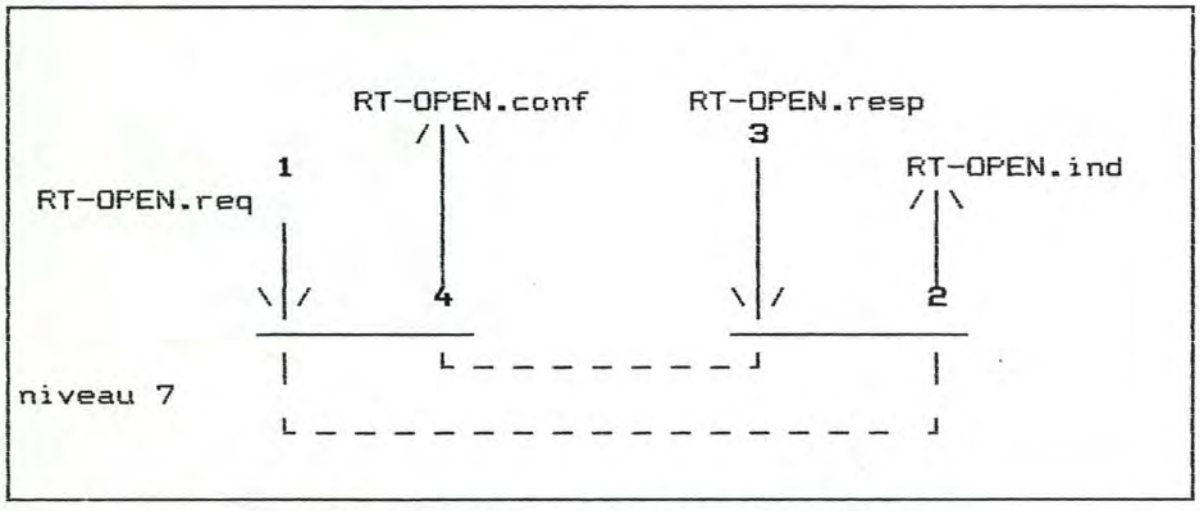

Figure 9 : La primitive RT-OPEN

<u>les paramètres et leur signification</u>

Nous n'envisagerons que les paramètres du premier groupe, utiles directement au service RTSE.

. Le mode du dialogue

- est le paramètre de demande d'ouverture pour un monologue ou pour un dialogue.
- est un paramètre obligatoire.
- apparaît dans les primitives request et indication.

. Le tour initial

- représente quand le mode dialogue a été choisi, celui qui commencera à transmettre (initiateur ou répondeur).
- est un paramètre obligatoire.
- apparaît dans les primitives request et indication.

. Le protocole application

- désigne le protocole application qui gouverne l'association.
- est un paramètre optionnel.
- apparaît dans les primitives request et indication, uniquement dans le cas du mode d'association X410 -1984.

. Les données de l'utilisateur

- est le paramètre laissé libre pour l'application utilisatrice. Celle-ci peut y mettre ce quelle veut selon la convention du protocole utilisé.
- paramètre optionnel.
- apparaît dans les quatre primitives.

3.8.2.2 Fermeture d'une liaison fiable

La primitive associée, représentée à la figure 10, est le RT-CLOSE. C'est une primitive de service confirmé. Elle ne peut être utilisée que si l'entité possède la permission de clôturer la liaison.

Lorsqu'une entité a terminé son travail, elle signale la fin de la liaison en utilisant le RT-CLOSE.Request.

Cette phase ne nécessite aucun paramètre propre au service RTSE.

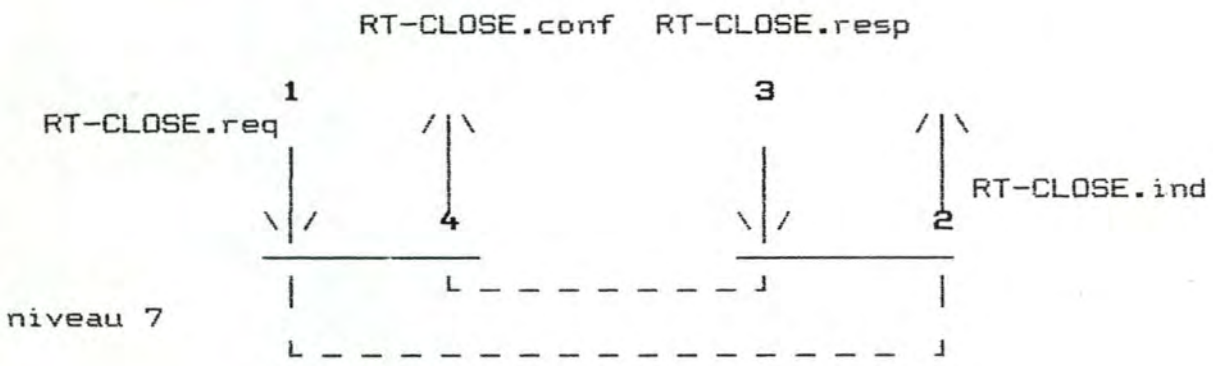

Figure 10 : La primitive RT-CLOSE

3.8.2.3 Abandon d'une liaison fiable

La primitive de ce service est le RT-U-ABORT. C'est un service non confirmé ; il stoppe immédiatement la liaison entre les deux entités paires; toutes les données en cours sont perdues.

Cette primitive, représentée à la figure 11, peut être employée par les deux utilisateurs du RTSE pour signaler que la liaison en cours ne peut être maintenue.

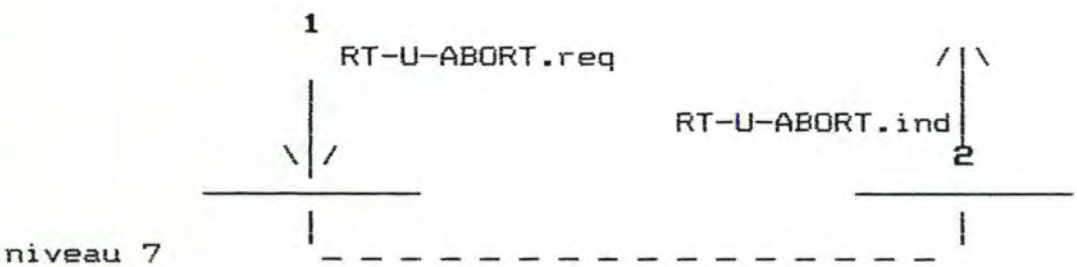

Figure 11 : la primitive RT-U-ABORT

Le paramètre associé et sa signification

. Les données de l'utilisateur

- paramètre réservé a l'application utilisatrice, son contenu est spécifié dans le protocole utilisé.
- est un paramètre optionnel.
- apparaît dans les 4 primitives.

3.8.2.4 Fermeture du fournisseur de l'application

La primitive utilisée est le RT-P-ABORT. Son résultat est une déconnexion immédiate et la perte de toutes les données en cours de route.

Elle n'a aucun paramètre et est représentée à la figure 12.

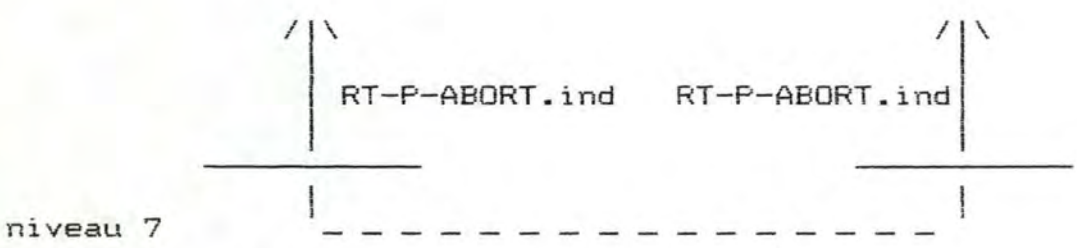

Figure 12 : La primitive RT-P-ABORT

3.8.2.5 Transfert de données de façon fiable

La primitive de transfert d'APDU est le RT-TRANSFER (cfr figure 13). Un appel à RT-TRANSFER.request équivaut à une demande de transfert d'APDU. Lorsque celui-ci est terminé, le fournisseur du service RTSE émet un RT-TRANSFER.confirmation.

Cette primitive se déroule donc en **trois temps :**

- demande de transmission (1)
- signal de transmission à l'entité répondante (2)
- signal de bonne fin de transmission (3)

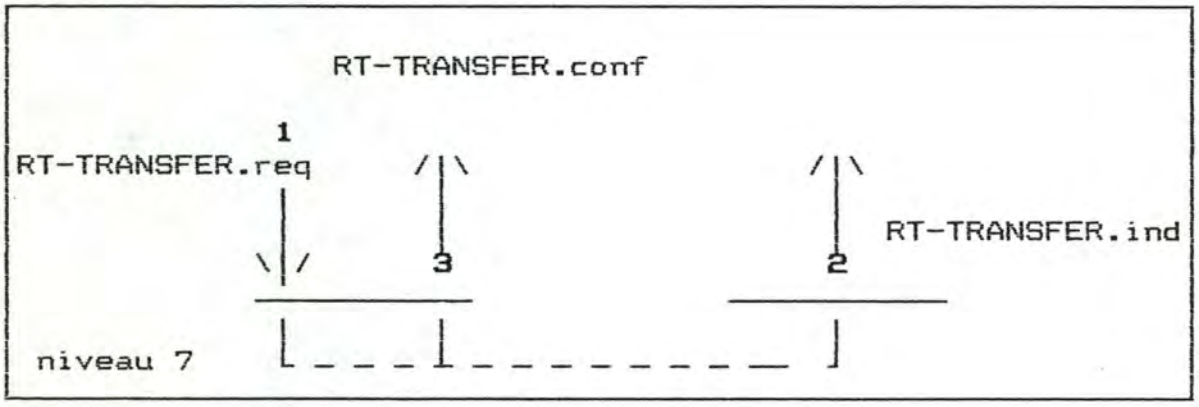

Figure 13 : La primitive RT-TRANSFER

les paramètres et leur signification

Trois paramètres sont utilisés par cette primitive.

. L'APDU

- est l'unité de donnée de protocole application à transférer de façon fiable.
- est un paramètre obligatoire.
- apparaît dans les primitives request et indication.

. Le temps de transfert

- est le temps de transfert maximum endéans lequel l'APDU doit être transmis à l'entité paire.
- est un paramètre obligatoire.
- apparaît dans la primitive request.

. Le résultat

- est le signal de bonne fin de transmission.
- est un paramètre obligatoire.
- apparaît dans la primitive confirmation.

3.8.2.6 Demande de permission de transmettre

Pour pouvoir émettre des APDU, en utilisant le RT-TRANSFER, une entité doit posséder la permission de le faire, c'est-à-dire que ce soit à son tour de pouvoir transférer des données.

Si ce n'est pas le cas, elle peut le demander en utilisant la primitive de service non-confirmé RT-TURN-PLEASE.request (cfr figure 14).

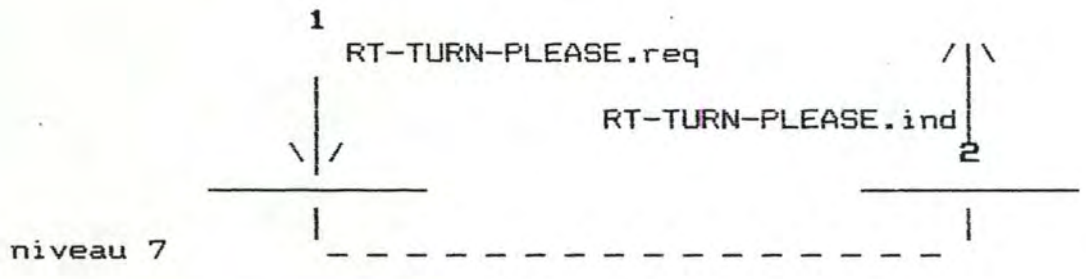

Figure 14 : La primitive RT-TURN-PLEASE

le paramètre associé et sa signification

. La priorité

- à chaque action du RTSE est défini un niveau de priorité. C'est celui-ci que représente le paramètre à exécuter.
- est un paramètre optionnel. S'il n'est pas présent, la valeur par défaut est 0, c'est-à-dire la priorité la plus haute.
- apparaît dans les deux temps de la primitive.

3.8.2.7 Accord de permission de transmettre

Quand une entité a terminé de transmettre ses APDU, elle peut céder le tour à son entité paire.

Pour ce faire, elle utilise la primitive RT-TURN-GIVE.req . Cette primitive de service non confirmé n'utilise aucun paramètre (cfr figure 15).

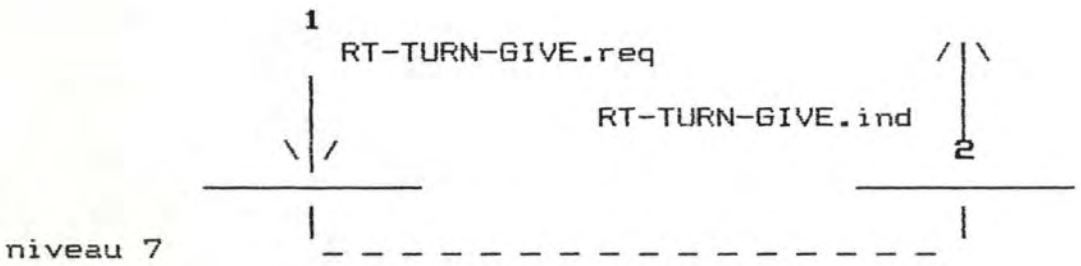

Figure 15 : La primitive RT-TURN-GIVE

3.8.3 Définition du protocole

Le protocole RTSE, [CCITTX228] et [ISO9066-2], définit les procédures à suivre pour l'échange de données entre deux entités.

Nous allons analyser, en détail, comment chacune des actions possibles sera réalisée : l'établissement d'association, la fermeture d'association, le transfert de données, la demande du tour, la cession du tour, le rapport d'erreurs, le traitement d'erreurs, la correction d'erreurs et l'abandon d'association.

Une représentation d'une entité application employant le RTSE est donnée à la figure 16.

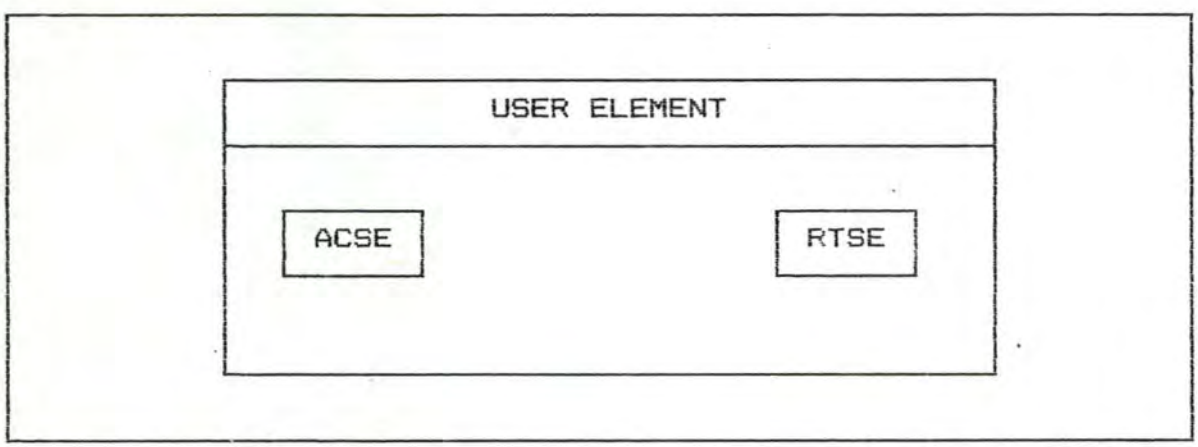

Figure 16 Une entité application employant le RTSE

3.8.3.1 Etablissement d'association

. Procédure

Pour établir une association, l'utilisateur du service RTSE utilise la primitive RT-OPEN.request.

Ensuite, le fournisseur du service construit un APDU RTORQ qui sera transmis dans le champ de données de l'utilisateur du A-ASSOCIATE (remarquons que l'APDU RTORQ se trouvera donc dans un autre APDU,le AARQ).

Si l'association a été refusée par l'ACSE sous-jacent, la procédure d'établissement se termine.

Sinon, le RTSE de l'entité répondante reçoit le RTORQ et l'analyse.

Si cet APDU n'est pas acceptable, le RTSE répondant fabrique un APDU RTORJ qu'il transmet via le paramètre de données de l'utilisateur du A-ASSOCIATE.response et du RT-OPEN.confirmation.

Si cet APDU est acceptable, elle indique son arrivée par un RT-OPEN.indication à son utilisateur.

Celui-ci peut alors répondre à la demande en utilisant la primitive RT-OPEN.response. Il le fait en construisant soit un APDU RTOAC s'il accepte l'association, soit un APDU RTORJ s'il la refuse.

Cet APDU est transféré dans le champ données de l'utilisateur de la primitive A-ASSOCIATE.response.

Le RTSE demandeur recevra le A-ASSOCIATE.confirmation et exécutera le RT-OPEN.confirmation en y ajoutant l'APDU arrivé et en donnant la valeur adéquate au paramètre résultat.

Cette procédure est schématisée à la figure 17.

. Le RTORQ

Les champs sont dans l'ordre :

- la **taille du point de contrôle**, c'est-à-dire le nombre maximum de Kilo Byte qui peuvent être envoyé entre deux points de synchronisation mineurs.
- la **taille de la fenêtre**, c'est-à-dire le nombre maximum de confirmation de pose de points de synchronisation mineurs à attendre avant de continuer à transmettre (3 par défaut).
- le **mode du dialogue:** monologue ou dialogue.
- les **données de l'utilisateur.**
- l'**identificateur de connexion session.**
- le **protocole application** (si mode X410-1984)

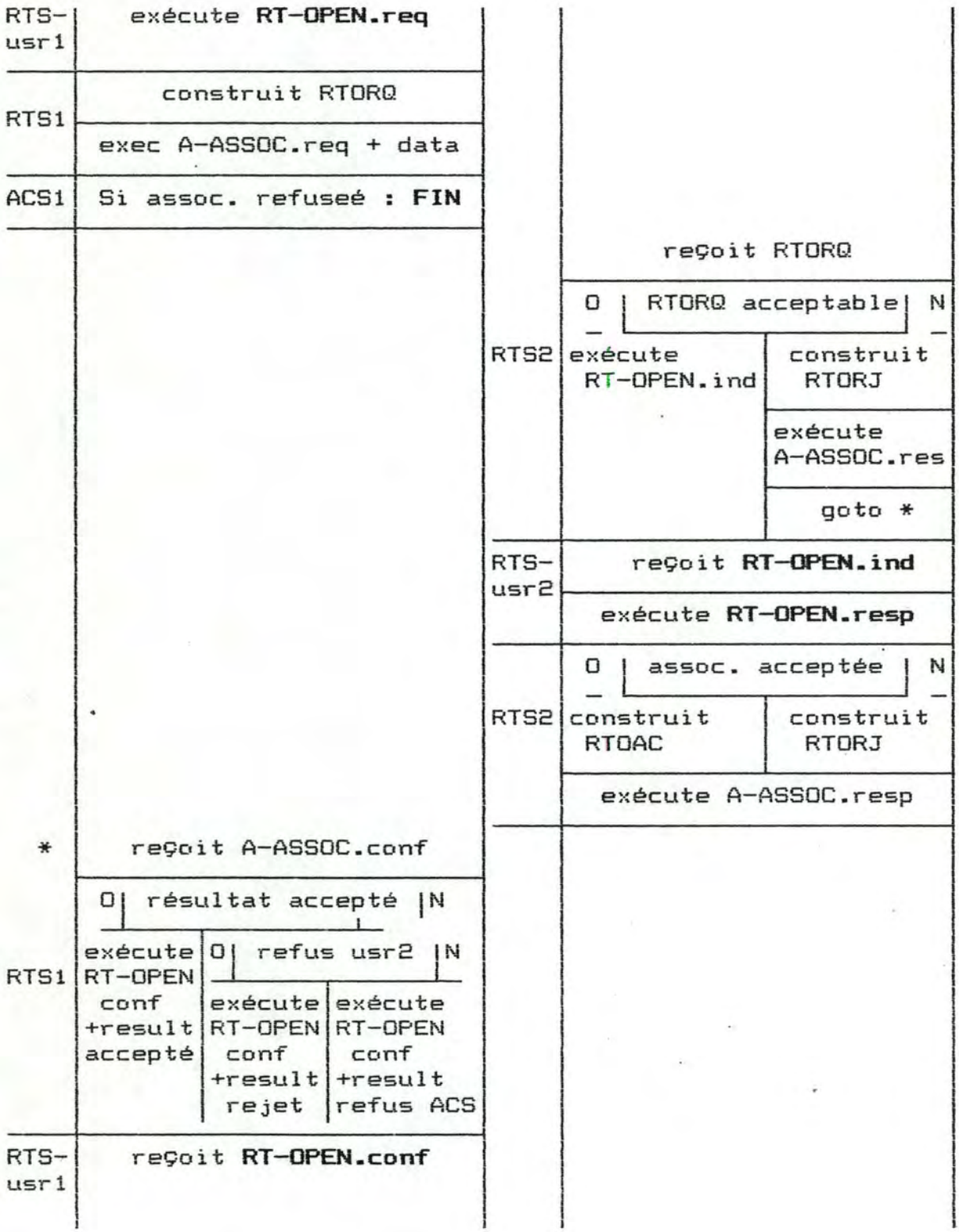

Figure 17: la procédure d'établissement d'association du RTSE

. Le RTOAC

Les champs sont dans l'ordre :

- la **taille du point de contrôle.**
- la **taille de la fenêtre.**
- les **données de l'utilisateur.**
- l'**identificateur de connexion session.**

. LeRTORJ

Les champs sont dans l'ordre :

- la **raison du refus.**
- les **données de l'utilisateur.**

Tous ces APDU sont décrits en ASN.1 à la figure 18.

```
RTORQapdu ::= SET {
    checkpointSize[0] IMPLICIT INTEGER DEFAULT 0,
    windowSize[1] IMPLICIT INTEGER DEFAULT 3,
    dialogueMode[2] IMPLICIT INTEGER {
        monologue(0), twa(1)} DEFAULT monologue,
    connectionDataRQ[3] ConnectionData,
    applicationProtocol[4] IMPLICIT INTEGER OPTIONAL
        -- uniquement en mode X410-1984
    }

RTOAC ::= SET {
    checkpointSize[0] IMPLICIT INTEGER DEFAULT 0,
    windowSize[1] IMPLICIT INTEGER DEFAULT 3,
    connectionDataAC[2] ConnectionData
    }

RTORJapdu ::= SET {
    refuseReason[0] IMPLICIT OACS.refuseReason OPTIONAL,
        -- uniquement en mode X410-1984
    userDataRJ[1] ANY OPTIONAL
        -- uniquement en mode normal
    }


ConnectionData ::= CHOICE {
    open[0] ANY,
        -- données de l'utilisateur RTSE
    recover[1] IMPLICIT SessionConnectionIdentifier
    }
SessionConnectionIdentifier ::= SEQUENCE {
    CallingSSuserReference,
    CommonReference,
    [0] IMPLICIT AdditionalReferenceInformation OPTIONAL
    }
CallingSSuserReference ::= CHOICE {
    T61String, -- uniquement en mode X410-1984
    OCTET STRING -- uniquement en mode normal
    }
CommonReference ::= UTCTime
AdditionalReferenceInformation ::= T61String
```

Figure 18 : la description ASN1 des APDU RTORQ RTOAC RTORJ

3.8.3.2 Fermeture d'association

. Procédure

La fermeture ne peut être entamée que si l'entité RTSE possède le jeton de fermeture.

L'utilisateur du service RTSE demandant la fermeture exécute la primitive RT-CLOSE.req.

Le RTS fournisseur exécute alors un A-RELEASE.req.

L'autre RTS reçoit alors un A-RELEASE.ind et exécute un RT-CLOSE.ind pour prévenir l'entité paire.

L'utilisateur de RTSE est ainsi prévenu de la fermeture de l'association; il exécute un RT-CLOSE.resp.

Son service RTSE exécute alors un A-RELEASE.resp.

Le service RTSE demandeur de la fermeture sera prévenu dès la réception du A-RELEASE.conf et exécutera un RT-CLOSE.conf à destination de son utilisateur.

Cette procédure est schématisée à la figure 19.

. Aucun APDU n'est associé à cette phase de fermeture.

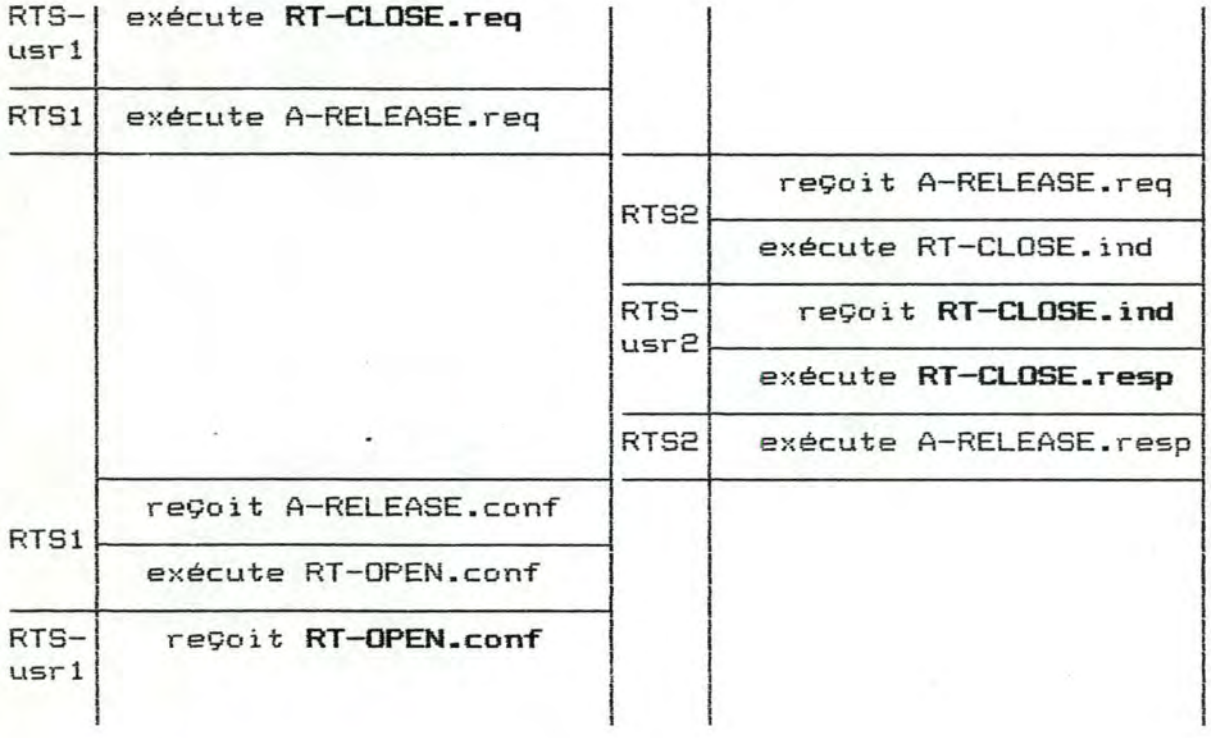

Figure 19 : La procédure de fermeture d'association

3.8.3.3 Transfert de données

. Objectif

L'objectif du RTSE est de permettre un transfert fiable d'APDU entre deux entités associations. C'est donc lors du transfert des données qu'il faut s'attendre à trouver un mécanisme permettant d'assurer que l'APDU a bien été transmis.

Pour réaliser cet objectif, le RTSE utilisera certaines des unités fonctionnelles de la couche session.

Remarquons aussi que l'ACSE n'a plus aucun rôle dans cette phase de la communication puisqu'il ne s'agit ni d'une connexion ni d'une déconnexion.

. Procédure

Un transfert d'APDU ne peut se faire que si l'entité demandeuse possède le tour pour émettre.

Si c'est le cas, un transfert se fait par l'appel de la primitive RT-TRANSFER.req.
Le fournisseur RTSE encode alors l'APDU a transférer. Ensuite il ouvre une activité (P-ACTIVITY-START.req) (un transfert d'APDU = une activité) et exécute un P-DATA.req contenant dans le paramètre de données de l'utilisateur l'APDU RTTR.
Il pose alors un point de synchronisation mineur (P-MINOR-SYNC.req) avant de continuer a envoyer les autres parties de son APDU grâce au P-DATA. Il peut faire cela tant qu'il reste des RTTR à transmettre.
Quand il a terminé, il exécute un P-ACTIVITY-END.req.

De l'autre coté, après avoir reçu le signal de début d'activité, le service RTSE répondant met en sécurité, au fur et à mesure qu'ils arrivent les APDU;il exécute alors les P-MINOR-SYNC.resp correspondants.

Quand il reçoit un P-ACTIVITY-END.ind, il sait que le transfert est terminé et renvoie un P-ACTIVITY.resp.

Les deux services RTSE signalent alors à leurs utilisateurs que le transfert est terminé en utilisant d'une part, les primitives RT-TRANSFER.conf pour l'émetteur et d'autre part, le RT-TRANSFER.ind pour le récepteur.

Cette procédure est schématisée à la figure 20.

. Le RTTR

Le seul champ est **l'APDU** à transférer. Cet APDU est décrit en ASN.1 à la figure 21.

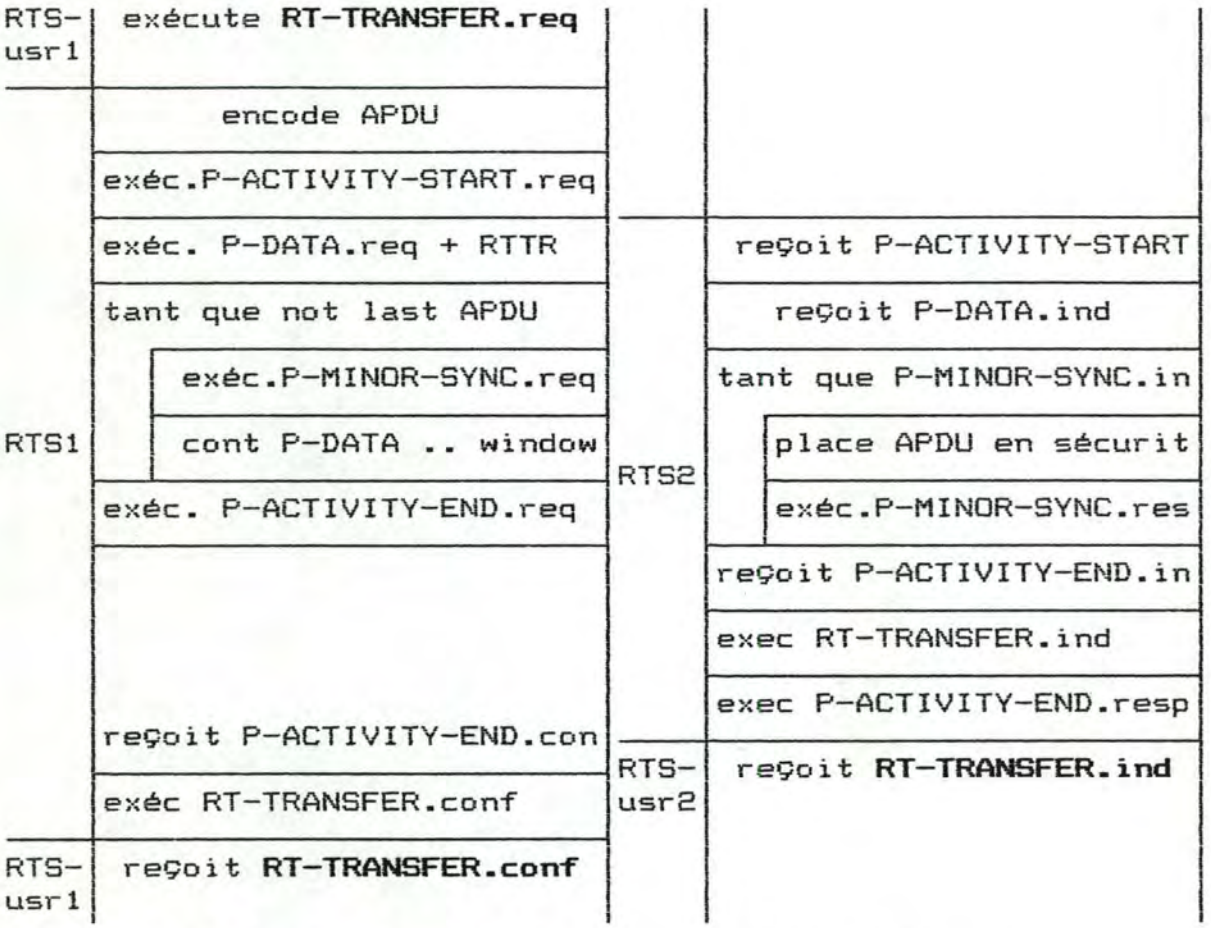

Figure 20 : la procédure de transfert d'APDU

```
RTTRapdu ::= OCTET STRING
```

Figure 21 : la description ASN1 de l'APDU RTTR

3.8.3.4 Demande du tour

. Procédure

L'utilisateur du service RTSE qui souhaite pouvoir envoyer des données doit posséder la permission de le faire; ce doit être son tour d'émettre.

Pour posséder celui-ci, il faut le demander en utilisant la primitive RT-TURN-PLEASE.req.

Le fournisseur du service regarde s'il possède le tour. Si oui, il n'a rien à faire.
Sinon, il construit éventuellement un APDU RTTP et l'envoie à l'entité RTSE paire via le paramètre données de l'utilisateur de la primitive P-TOKEN-PLEASE.req.

Le service paire reçoit alors le signal d'indication et le transmet à son utilisateur avec l'exécution de la primitive RT-TURN-PLEASE.ind.

Cette procédure est schématisée à la figure 22.

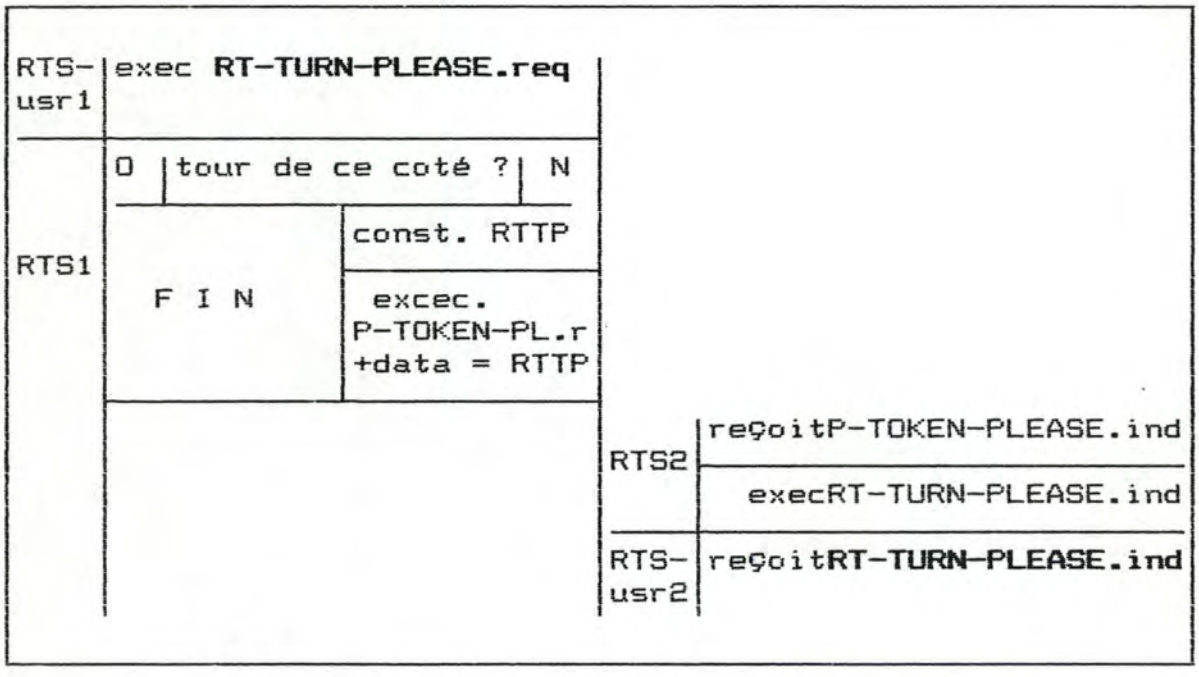

Figure 22 : La procédure de demande de tour

. Le RTTP

Cette unité de donnée de protocole peut contenir un paramètre donnant la **priorité** de l'opération qu'il a à effectuer.

Ce paramètre est facultatif et l'APDU en lui-même est donc également facultatif. Celui-ci est représenté en ASN.1 à la figure 23.

```
RTTPapdu ::= INTEGER  -- priorité
```

Figure 23 : la description ASN1 de l'APDU RTTP

3.8.3.5 Cession du tour

. Procédure

Quand l'entité qui possède le tour veut donner à son entité paire la possibilité de transférer des données, elle exécute la primitive RT-TURN-GIVE.req.

Le RTSE serveur exécute alors un P-CONTROL-GIVE.req

L'autre RTSE recevant cette indication, signale que le message est bien reçu en émettant un P-CONTROL-GIVE.resp à destination du RTSE paire et informe son utilisateur qu'elle possède le tour (RT-TURN-GIVE.ind).

Cette procédure est schématisée à la figure 24.

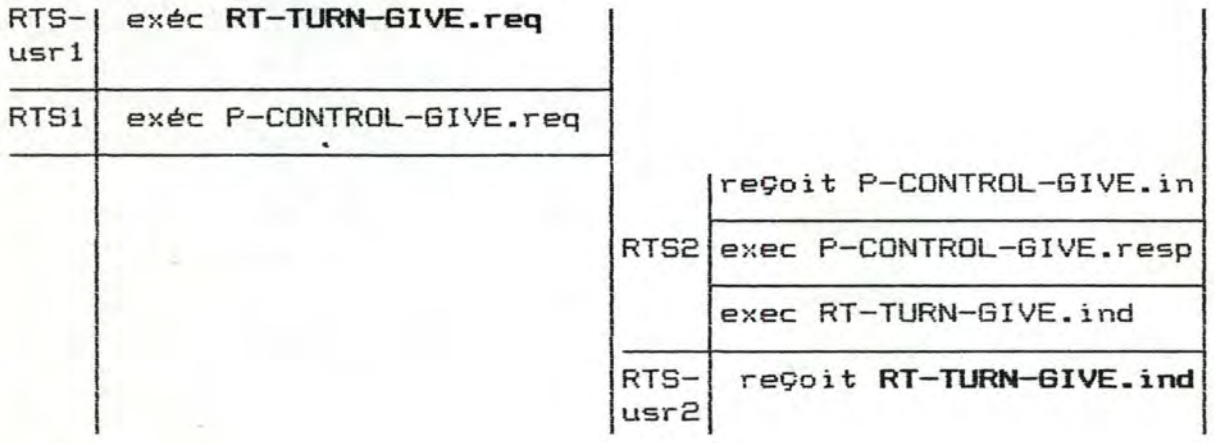

Figure 24 : La procédure de cession de tour

. Cette primitive n'utilise pas d'APDU.

3.8.3.6 Rapport d'erreurs

. Procédure

L'utilisateur RTSE de l'association qui se rencontre un problème peut en avertir l'émetteur en utilisant la primitive P-U-EXCEPTION-REPORT.req.

Le fournisseur qui a un problème, avertit l'entité paire en utilisant la primitive P-P-EXCEPTION-REPORT.req

. Cette procédure n'utilise aucun APDU.

3.8.3.7 Traitement d'erreurs

. Procédure

Selon la gravité de l'erreur, le traitement à accomplir sera différent.

Si un point de synchronisation mineur a déjà été posé, l'émetteur peut interrompre l'activité en utilisant un P-ACTIVITY-INTERRUPT pour traiter son erreur (cas 1).

Si aucun point de synchronisation n'a encore été posé, l'émetteur utilisera le P-ACTIVITY-DISCARD (Cas 2).

Si le problème est trop grave, et que le RT-TRANSFER.conf n'arrive pas, l'utilisateur émetteur devra utiliser la primitive RT-ABORT.req pour abandonner l'association (Cas 3).

Si c'est le fournisseur RTSE qui se trouve face à un problème insurmontable, il exécutera un A-P-ABORT pour en avertir son utilisateur (Cas 4).

. Un APDU RTAB sera utilisé dans les deux derniers cas. Celui-ci est décrit en 3.8.3.9

3.8.3.8 Récupération d'erreurs

. Procédure

Selon le type d'erreur rencontré, la reprise sera différente.

Dans les cas 1 et 3, une fois que tout est rentré dans l'ordre, il suffit d'exécuter un P-ACTIVITY-RESUME; ensuite, il suffit de reprendre l'émission en utilisant le P-DATA.

Dans le deuxième cas, une nouvelle tentative de transfert est nécessaire; il faut donc refaire un P-ACTIVITY-START.req.

Dans les autres cas, il faudra recommencer une nouvelle association et donc repartir au RT-OPEN.req.

. Pas d'APDU pour cette phase.

3.8.3.9 Abandon d'association

. Procédure

Quand aucune reprise n'est possible, il faut abandonner l'association. Pour cela, on utilise le RT-U-ABORT.req pour avertir l'entité paire.

Le fournisseur génère alors un APDU RTAB qui sera transmis dans le paramètre données de l'utilisateur.

. L'ABRT

L'APDU RTAB est composé des champs suivants :

- la raison de l'**abandon.**
- le **paramètre** faisant problème
- les **données de l'utilisateur**

Cet APDU est décrit en ASN.1 à la figure 25.

```
ABRTapdu ::= SET {
   abortReason[0] IMPLICIT AbortReason OPTIONAL,
   reflectedParameter [1] IMPLICIT BIT STRING OPTIONAL,
   }

AbortReason ::= INTEGER {
   localSystemProblem(0),
   invalidParameter(1),
   unrecognizedActivity(2),
   temporaryProblem(3),
   protocolError(4),
   permanentProblem(5),
   userError(6),
   transferCompleted(7)
   }
```

Figure 25 : la description ASN1 de l'APDU RTAB

3.9 Le ROSE (Remote Operation Service Element)

3.9.1 Objectif

Une méthode bien connue dans la construction d'application distribuée est celle des opérations à distance (deux programmes sont en coopération en se demandant d'exécuter des opérations).

Cette idée est déjà très répandue dans le monde des télécommunications; ses précurseurs semblent être les concepteurs du protocole RPC (Remote Procedure Call)[RPC].

ISO a également défini son protocole d'opérations à distance. Celui-ci est un des services possibles offerts aux applications; c'est donc un élément de service d'application, un ASE.

Les normes relatives définissent: le modèle ROSE (3.9.2), le service ROSE (3.9.3), le protocole à utiliser (3.9.4) mais aussi et surtout un notation ROSE (3.9.5) et la correspondance avec le service (3.9.6). Nous terminerons ce paragraphe par une comparaison entre le protocole ROSE et le protocole RPC (3.9.7).

exemple

Il sera très facile d'utiliser des programmes qu'on ne possède pas sur son site grâce à cet élément de service; les programmes peuvent être distribués.

3.9.2 Le modèle ROSE

Ce modèle est défini dans [CCITTX219] et [ISO9072-1].

3.9.2.1 Le demandeur et le répondeur

Typiquement, une opération à faire exécuter à distance peut se concevoir en deux étapes.

Premièrement, une entité application demande à une entité paire qu'elle exécute une opération (la demande).
Secondement, une fois celle-ci terminée, l'entité paire fait connaître la réponse au demandeur (la réponse).

La figure 26 schématise ces concepts.

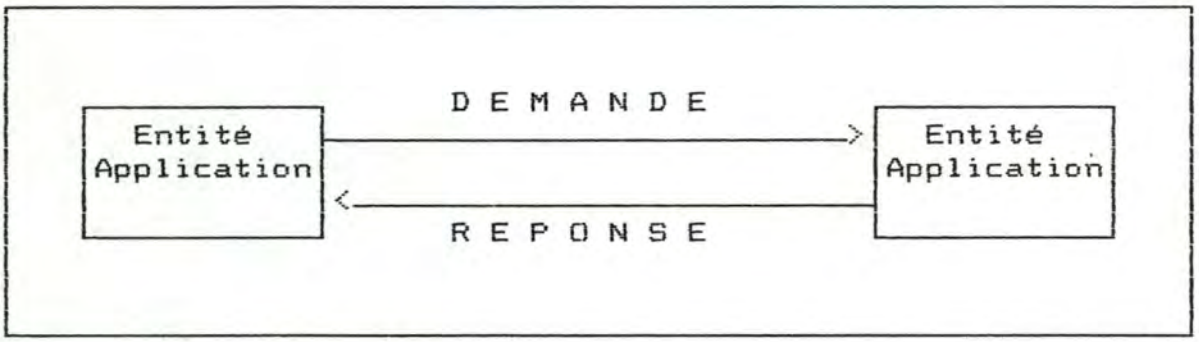

Figure 26 : Modèle d'une opération à distance

3.9.2.2 Classification des opérations à distance

Une opération à distance peut être soit une :

- opération synchrone, c'est à dire que le demandeur attend une réponse avant de demander l'exécution d'une autre opération (demande bloquante).
- opération asynchrone, c'est à dire que le demandeur peut demander l'exécution de plusieurs opérations à distance sans attendre les réponses respectives (demande non bloquante).
- opération nécessitant un réponse en cas de succès comme en cas d'échec.
- opération nécessitant une réponse en cas d'échec seulement.
- opération nécessitant une réponse en cas de succès seulement.
- opération ne nécessitant pas de réponse.

A partir de ces distinctions, le modèle ROSE se donne cinq classes d'opérations à distance :

classe 1: synchrones avec rapport de réussite et d'échec.
classe 2: asynchrones avec rapport de réussite et d'échec.
classe 3: asynchrones avec rapport d'échec seulement.
classe 4: asynchrones avec rapport de réussite seulement.
classe 5: asynchrones sans réponse.

3.9.2.3 Les opérations liées

Une autre notion est également mise en évidence dans ce modèle: celle de l'opération liée (pour signaler qu'une opération est liée à une autre).

Une opération est liée à une autre opération si la première est éxecutée lors de à l'exécution de la seconde.

3.9.2.4 Les types d'association

trois types d'association sont définis :

- l'association où l'initiateur de la connexion est le demandeur de l'exécution des opérations.
- l'association où le répondeur à la connexion est le demandeur de l'exécution des opérations.
- l'association où les deux entités peuvent demander l'exécution d'opérations.

3.9.2.5 Notation d'opérations à distance

Une caractéristique très importante du ROSE est qu'il permet une indépendance avec la communication ISO.

Il utilise à cet effet une notation (3.9.5) basée sur les principes de la programmation orientée objet. Des outils automatiques peuvent ainsi être créés pour lier automatiquement la notation ROSE à un environnement d'exécution d'applications.

De tels outils utilisent une fonction de correspondance entre les opérations définies avec la notation ROSE et les ASE sous-jacents (ACSE, RTSE, ROSE).

Le ROSE sera utilisé pour l'exécution des opérations à distance proprement dites.

L'ACSE (ou le RTSE, si l'utilisateur souhaite une association fiable) sera utilisé pour le contrôle de l'association.

Ceci est schématisé par la figure 27.

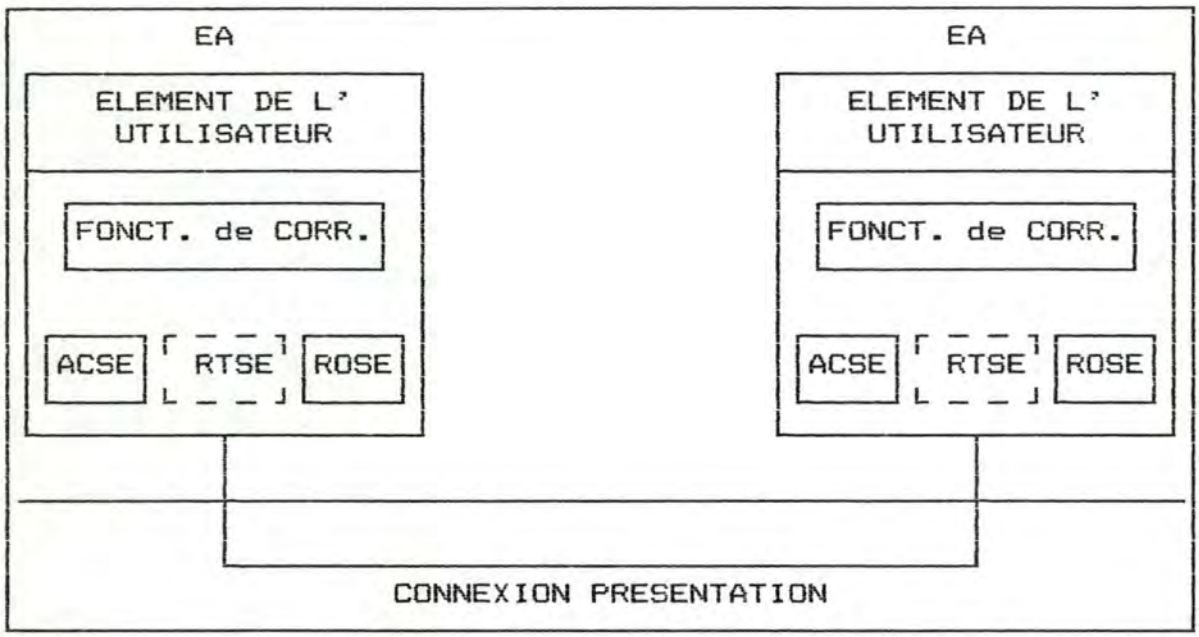

Figure 27: la couche application

3.9.3 Définition du service

Cinq primitives sont offertes à l'utilisateur du service ROSE.

- signaler une demande d'exécution d'une opération à distance.
- signaler le résultat de l'exécution d'une opération à distance.
- signaler l'erreur de l'exécution d'une opération à distance.
- signaler un problème de l'utilisateur ROSE.
- signaler un problème du fournisseur ROSE.

Ce service est défini dans [CCITTX219] et [ISO9072-1].

3.9.3.1 Signaler une demande d'exécution d'une opération à distance

La primitive associée, représentée à la figure 28, est le RO-INVOKE. C'est une primitive en deux temps (service non confirmé).

Elle est à utiliser par un utilisateur ROSE pour demander l'exécution d'une opération sur un site paire. Une association doit cependant avoir déjà été établie avant d'utiliser le RO-INVOKE.Request

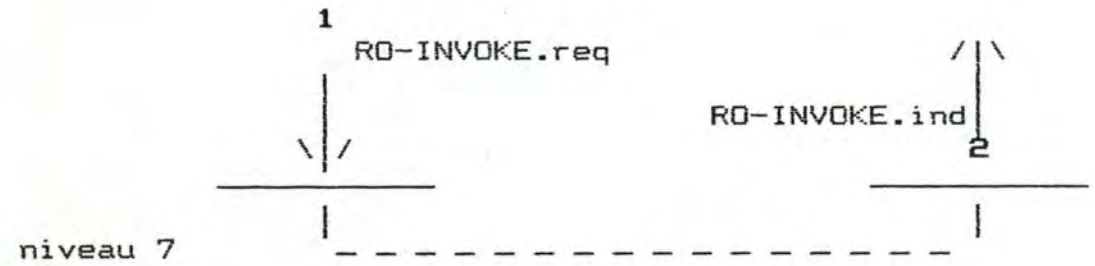

Figure 28 : la primitive RO-INVOKE

Les paramètres et leur signification

. La valeur de l'opération

- Toutes les opérations qu'il est possible de demander d'exécuter à une application ont un numéro identifiant connu des deux entités. C'est ce numéro qui est donné dans ce paramètre.
- Ce paramètre est obligatoire.
- Il apparaît dans les deux temps de la primitive.

. La classe de l'opération

- Ce paramètre donne la classe de l'opération à exécuter (cfr modèle ROSE).
- Ce paramètre est optionnel.
- Il apparaît dans les deux temps de la primitive.

. L'argument

- L'opération peut nécessiter un paramètre qui est fourni par le demandeur. C'est l'argument.
- Ce paramètre est optionnel.
- Il apparaît dans les deux temps du RO-INVOKE.

. L'identificateur de demande

- Ce paramètre est un identificateur de la demande d'exécution. Il réapparaît dans la réponse.
- Ce paramètre est obligatoire.
- Il apparaît dans les deux phases.

. L'identificateur lié

- Si ce paramètre est présent, cela indique que l'opération est une opération liée, au sens défini dans le modèle et le numéro apparaissant ici est celui de l'opération à qui celle-ci est liée.
- Ce paramètre est optionnel.
- Il apparaît dans les deux phases.

. La priorité

- Ce paramètre donne la priorité avec laquelle l'APDU construit devra être crée. Plus la valeur est petite, plus la priorité est grande.
- Ce paramètre est optionnel.
- Il apparaît dans le request.

3.9.3.2 Signaler le résultat de l'exécution d'une opération à distance

Le service RO-RESULT est utilisé parle répondeur pour signaler, en cas de succès, le résultat de l'exécution d'une opération. C'est une primitive de service non confirmé; elle est représentée à la figure 29.

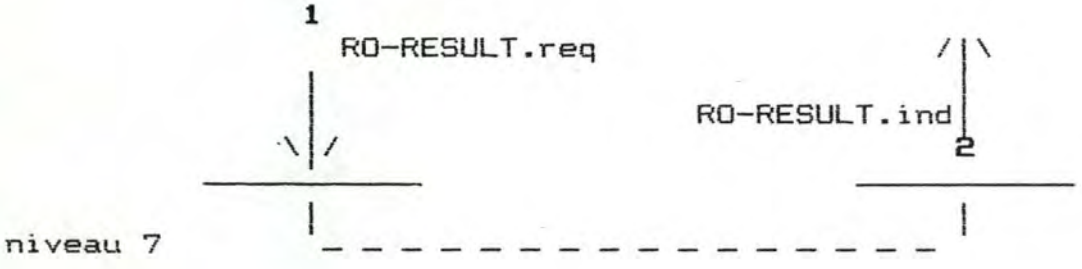

Figure 29 : la primitive RO-RESULT

Les paramètres et leur signification

. Le résultat

- Ce paramètre contient le résultat de l'exécution de l'opération. Son type doit être défini en commun accord entre les deux utilisateurs ROSE.
- Ce paramètre est optionnel.
- Il apparaît dans les deux phases.

. L'identificateur de la demande

- C'est le paramètre rappelant quel numéro portait la demande.
- Ce paramètre est obligatoire.
- Il apparaît dans les deux phases.

. La priorité

- Ce paramètre donne la priorité avec laquelle l'APDU construit devra être crée. Plus la valeur est petite, plus la priorité est grande.
- Ce paramètre est optionnel.
- Il apparaît dans le request.

3.9.3.3 Signaler l'erreur de l'exécution d'une opération à distance

Le service RO-ERROR est utilisé par le répondeur pour signaler, en cas d'erreur , le mauvais fonctionnement de l'exécution d'une opération. C'est une primitive de service non confirmé.

Cette primitive est représentée par la figure 30.

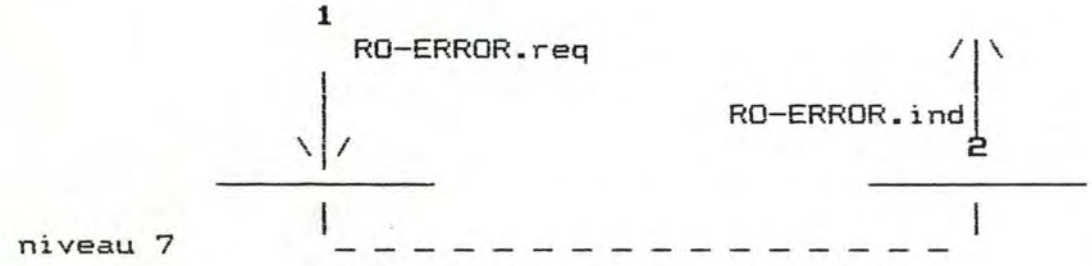

Figure 30 : la primitive RO-ERROR

Les paramètres et leur signification

. La valeur de l'erreur

- Ce paramètre identifie l'erreur produite. Sa valeur est définie d'un commun accord entre les deux utilisateurs ROSE.
- Ce paramètre est obligatoire.
- Il apparaît dans les deux phases.

. Le paramètre erroné

- Ce paramètre fournit des informations supplémentaires sur l'erreur. Son type est connu des deux utilisateurs ROSE.
- Ce paramètre est obligatoire.
- Il apparaît dans les deux phases.

. L'identificateur de la demande

- C'est le paramètre rappelant quel numéro portait la demande.
- Paramètre obligatoire.
- Il apparaît dans les deux phases.

. La priorité

- Ce paramètre donne la priorité avec laquelle l'APDU construit devra être crée. Plus la valeur est petite, plus la priorité est grande.
- Paramètre optionnel.
- Il apparaît dans le request.

3.9.3.4 Signaler un problème de l'utilisateur ROSE

Ce service est utilisé pour permettre le rejet d'une demande d'exécution d'une opération; c'est le RO-REJECT-U. C'est un service non confirmé.

Cette primitive est représentée par la figure 31.

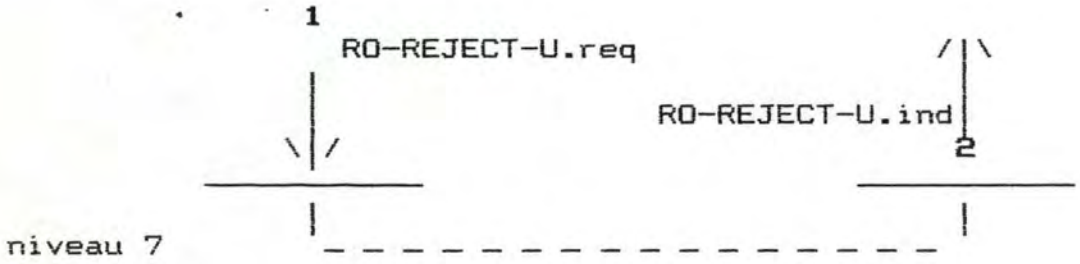

Figure 31 : la primitive RO-REJECT-U

Les paramètres associés et leur signification

. La raison du rejet

- Ce paramètre spécifie une des raisons définie dans la norme correspondante.
- Ce paramètre est obligatoire.
- Il apparaît dans les deux phases.

. L'identificateur de la demande

- C'est le paramètre rappelant quel numéro portait la demande.
- Ce paramètre est obligatoire.
- Il apparaît dans les deux phases.

. La priorité

- Ce paramètre donne la priorité avec laquelle l' APDU construit devra être crée. Plus la valeur est petite, plus la priorité est grande.
- Ce paramètre est optionnel.
- Il apparaît dans le request.

3.9.3.5 Signaler un problème du fournisseur ROSE

Ce service est fourni pour signaler au demandeur un problème détecté par le ROSE: le RO-REJECT-P. Cette primitive ne possède que la phase indication.

Cette primitive est représentée par la figure 32.

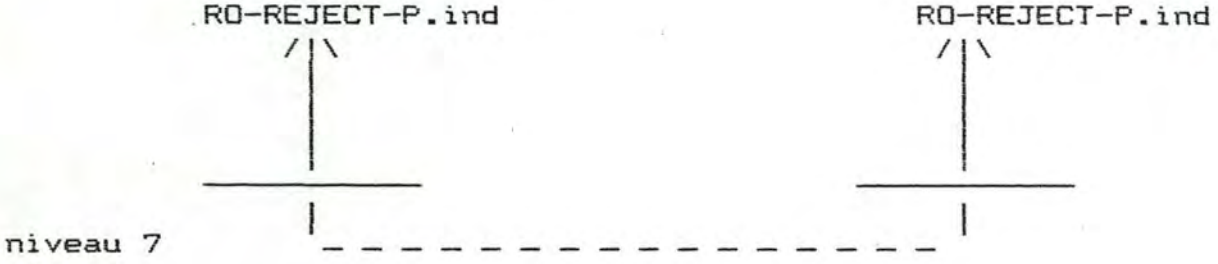

Figure 32 : la primitive RO-REJECT-P

Les paramètres associés et leur signification

. L'identificateur de la demande

- C'est le paramètre rappelant quel numéro portait la demande.
- Ce paramètre estfourni selon la réalisation du ROSE.

. Le paramètre retourné

- Ce paramètre contient le paramètre problème.
- Paramètre fourni selon la réalisation du ROSE.

. La raison du rejet

- Ce paramètre contient une des raisons spécifiées dans la norme correspondante [CCITTX219] et [ISO9072-1].
- Ce paramètre est fourni selon la réalisation du ROSE.

3.9.4 Définition du protocole

Le protocole ROSE, [CCITTX229] et [ISO9072-2], définit les procédures à suivre pour l'exécution d'opération sur des systèmes de télécommunications éloignés.

Nous analysons ici, en détail, comment la demande d'exécution et l'envoi de la réponse sont exécutés.

3.9.4.1 Demande d'exécution d'une opération à distance.

. Objectif

L'objectif de ce protocole est de décrire comment l'exécution d'opérations à distance peut être réalisée.

Auparavant, une association doit avoir été établie en utilisant un des deux protocoles prévu à cet égard: le RTSE ou l'ACSE (cfr figures 33 et 34).

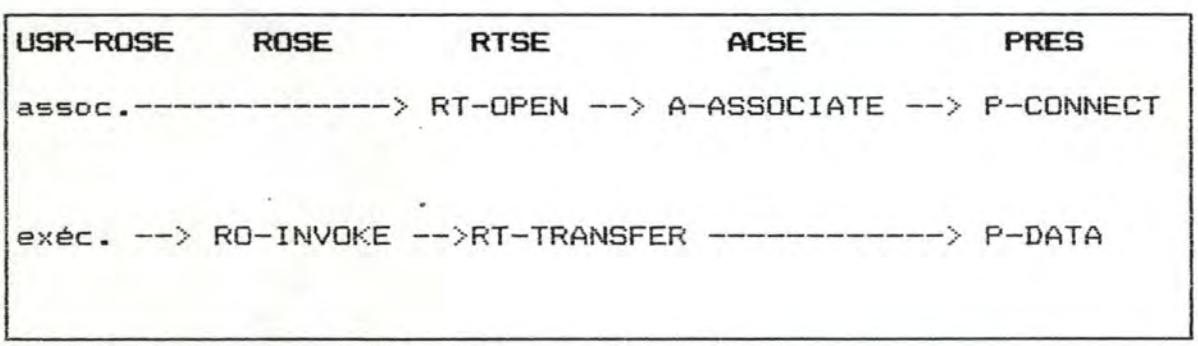

Figure 33 : Correspondance des ASE en service fiable

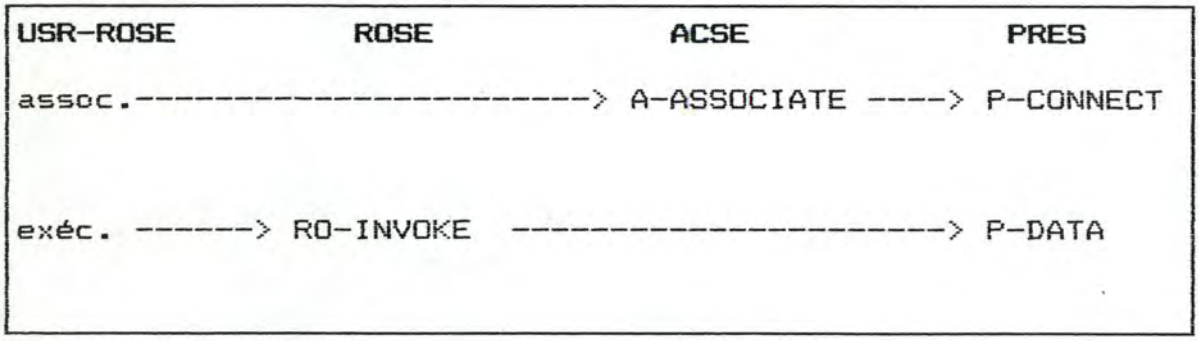

Figure 34 : Correspondance des ASE en service normal

. Procédure

La demande d'exécution d'une opération est faite par le demandeur; cette opération sera effectuée par le répondeur.

Le demandeur utilise la primitive RO-RESULT.Request pour sa demande. Son fournisseur ROSE construit alors un APDU ROIV qu'il transmet à l'entité paire, via la primitive de transfert d'APDU (P-DATA ou RT-TRANSFER).

Le répondeur reçoit cet APDU et exécute, s'il le peut, l'opération demandée.

La description de cet APDU en ASN.1 se trouve à la figure 35.

. Le ROIV (Remote Operation InVoke)

Les champs sont dans l'ordre :

- l'**identificateur de la demande** pour permettre à l'entité de différencier les différentes demandes.

- l'**identificateur lié** qui signale un lien avec une opération précédente; cet élément est facultatif.

- la **valeur de l'opération** représente l'opération à effectuer; cette valeur est déterminée d'un commun accord dans le protocole application utilisé.

- l'**argument** que peut nécessiter l'opération; il est facultatif et comprend l'ensemble des arguments de l'opération.

```
ROIVapdu  ::= SEQUENCE {
                  invokeId InvokeIdType,
                  linked-Id [0] IMPLICIT INTEGER OPTIONAL,
                  operation-value OPERATION,
                  argument ANY OPTIONAL }

invokeIdType ::= INTEGER
```

Figure 35 : Description en ASN.1 de l'APDU ROIV

3.9.4.2 Envoi du résultat de l'exécution d'une opération à distance

. Procédure

Une fois qu'il a terminé l'exécution d'une opération, le répondeur peut communiquer le résultat (si cela était spécifié dans le protocole en cours). Pour ce faire, il exécute un RO-RESULT.Request.

Son fournisseur construira alors un APDU RORS qu'il transmettra via la primitive de transfert de données.

. Le RORS (Remote Operation ReSult)

il comprend dans l'ordre les champs suivants :

- **l'identificateur de la demande** correspondant à celui du ROIV.
- le **résultat** de l'opération; ce champ est optionnel selon le protocole.

Cet APDU est décrit en ASN.1 à la figure 36.

```
RORSapdu  ::= SEQUENCE {
                  invokeId InvokeIdType,
                  result   ANY OPTIONAL }

invokeIdType ::= INTEGER
```

Figure 36 : Description en ASN.1 de l'APDU RORS

3.9.4.3 Envoi de l'erreur de l'exécution d'une opération à distance

. Procédure

Une fois qu'il a terminé l'exécution d'une opération, le répondeur peut signaler une erreur (si cela était spécifié dans le protocole en cours). Pour ce faire, il exécute un RO-ERROR.Request.

Son fournisseur construira alors un APDU ROER qu'il transmettra via la primitive de transfert de données.

. Le ROER (Remote Operation ERror)

il comprend dans l'ordre les champs suivants :

- l'**identificateur de la demande** correspondant à celui du ROIV.
- la **valeur de l'erreur** déterminée d'un commun accord.
- le **paramètre** posant problème; ce champ est optionnel.

Cet APDU est décrit en ASN.1 à la figure 37.

```
ROERapdu  ::= SEQUENCE {
                  invokeId InvokeIdType,
                  error-value ERROR,
                  parameter ANY OPTIONAL }

invokeIdType ::= INTEGER
```

Figure 37 : Description en ASN.1 de l'APDU ROER

3.9.4.4 Signal d'un refus de l'exécution d'une opération à distance

. Procédure

Le répondeur peut refuser d'exécuter une opération. Pour ce faire, il exécute un RO-REJECT.Request.

Son fournisseur construira alors un APDU RORJ qu'il transmettra via la primitive de transfert de données.

. Le RORJ (Remote Operation ReJect)

il comprend dans l'ordre les champs suivants :

- l'**identificateur de la demande** correspondant à celui du ROIV.
- le **problème** qui s'est posé.

Cet APDU est décrit en ASN.1 à la figure 38.

```
RORJapdu  ::= SEQUENCE {
                  invokeId InvokeIdType,
                  problem CHOICE {
                   [0] IMPLICIT GeneralProblem
                   [1] IMPLICIT InvokeProblem
                   [2] IMPLICIT ReturnResultProblem
                   [3] IMPLICIT ReturnErrorProblem

GeneralProblem ::= INTEGER {
                     unrecognisedAPDU (0),
                     mistypedAPDU (1),
                     badlystructuredAPDU (2)}

InvokeProblem  ::= INTEGER {
                     duplicateinvocation (0),
                     unreconisedOperation (1),
                     mistypedArgument (2),
                     ressourceLimitation (3),
                     initiatorReleasing (4),
                     unrecognizedLinkedID (5),
                     linkedResponseUnexpected (6),
                     unexpectedChildOperation (7)}

ReturnResultProblem ::= INTEGER {
                       unrecogisedinvocation (0),
                       resultResponseUnexpected (1),
                       unrecognisedError (2),
                       unexpectedError (3),
                       mistypedParameter (4)}
```

Figure 38 : Description en ASN.1 de l'APDU RORJ

3.9.4.5 Signal d'un refus du fournisseur ROSE

. Procédure

Le fournisseur ROSE qui rencontre un problème le signale par l'exécution d'un RO-REJECT-P.Indication

Il construit alors un APDU RORJ qu'il transmettra via la primitive de transfert de données pour en avertir l'entité paire.

. Le RORJ (Remote Operation ReJect)

il comprend dans l'ordre les champs suivants :

- l'**identificateur de la demande** correspondant à celui du ROIV.
- le **problème** qui s'est posé de type général.

3.9.5 Définition de la notation

La norme de définition de la notation ROSE se trouve dans [CCITTX219] et [ISO9072-1].

Celle-ci définit la notation ROSE pour la spécification d'un contexte d'application et les composants de la syntaxe abstraite relatifs au contexte de présentation qui sera utilisée par l'utilisateur ROSE.

Les fonctionnalités du contexte application seront fournies à l'élément de l'utilisateur au moyen des opérations à distance; ces dernières forment l'interface avec les opérations.

Différents types d'opérations existent et à chacun d'eux correspond une macro de la notation ASN.1. Ces macros sont définies en annexe 1 et résumée ici.

L'opération de liaison pour établir une association sera représentée par la macro BIND. C'est par cette opération que sera spécifié le contexte application.

L'opération de fin d'association sera représentée par la macro UNBIND.

Les opérations à faire exécuter seront représentées par la macro OPERATION et les erreurs associées par la macro ERROR.

3.9.6 Correspondance de la notation et du service

Les opérations spécifiées par la macro OPERATION correspondent au RO-INVOKE, pour la demande, et RO-RESULT ou RO-ERROR pour la réponse.

Selon que le service fiable a été demandé ou non, la correspondance s'établira avec le RTSE ou l'ACSE.

3.9.6.1 Service fiable

L'opération d'ouverture d'association, spécifiée par la macro BIND, correspondra au RT-OPEN.

L'opération de fermeture d'association, spécifiée par la macro UNBIND, correspondra au RT-CLOSE.

3.9.6.2 Service normal

L'opération d'ouverture d'association, spécifiée par la macro BIND, correspondra au A-ASSOCIATE.

L'opération de fermeture d'association, spécifiée par la macro UNBIND, correspondra au A-RELEASE.

3.9.7 Comparaison avec le RPC

Le protocole RPC (Remote Procedure Call) [RPC] est également un protocole d'exécution d'opération à distance. Utilisé notamment par SUN, il est souvent employé dans le développement d'applications réparties.

Il est intéressant de comparer le service qu'il offre avec celui du RTSE.

3.9.7.1 Le RPC

La méthode utilisée par ce service est celle des rendez-vous, où un client et un serveur communiquent.

Le client est le programme qui souhaite faire exécuter une opération sur un système éloigné. Il appelle alors un serveur en lui envoyant un paquet.

Celui-ci sait ainsi ce que veut le client, il fait exécuter la routine correspondante et renvoie le réponse.

3.9.7.2 Les couches du RPC

L'interface RPC est divisé en trois couches.

La couche supérieure fournit une transparence totale à l'utilisateur. Celui-ci peut employer des fonctions faisant appel à des opérations à distance sans devoir se préoccuper de la communication.

Ces quelques fonctions se trouvent dans une librairie particulière.

La couche du milieu est celle qui est le plus souvent utilisée. Le programme client utilise les routines registerrpc() et callrpc() pour faire exécuter des opérations sur un autre système.

La couche inférieure permet une utilisation des mécanismes de bas niveau telle que la manipulation de soquets. Programmer à ce niveau est à éviter autant que possible.

3.9.7.3 Les types de données

Dans les paramètres de callrpc() et registerrpc(), il est possible de définir le type des arguments et résultats de l'opération à exécuter, ainsi que leurs adresses.

Un certain nombre de routines permettent le codage et le décodage des données dans une syntaxe de transfert commune (XDR); cela permet des représentations différentes d'une même donnée sur deux systèmes coopérants.

3.9.7.4 Comparaison avec ROSE

. Les services

Dans la librairie librosy de ISODE, il existe un ensemble de routines déjà développées, spécifiques aux opérations à distance du service RTSE.

Le mécanisme de niveau inférieur n'est d'aucune utilité pour le service ROSE. Il n'y a, à ce niveau, aucune comparaison possible.

Par contre, le deuxième niveau est fort semblable dans les deux services puisque tous deux font appel à des routines de demande d'exécution d'opérations à distance: RO-INVOKE et RPC-CALL.

ROSE nécessite l'ensemble de l'architecture ISO et RPC se base directement sur le protocole TCP.

. Les types de données

Grâce à la structure en couche du modèle de référence ISO, les données manipulables par ROSE seront également codées et décodées dans une syntaxe de transfert au niveau de la couche présentation.

3.9.7.5 Conclusion

Nous pouvons donc conclure cette brève comparaison en disant que les deux protocoles partagent le même objectif de départ : permettre de faire exécuter une opération sur une machine éloignée et ce, en totale transparence avec les représentations propres utilisées pour le passage des arguments et résultats.

Néanmoins, les principes de mise en oeuvre sont relativement différents, excepté pour la procédure de demande du service.

CHAPITRE 4 : IMPLEMENTATION DES ASE DANS ISODE-4.0

4.1 Introduction

Le programme ISODE est une implémentation du modèle de référence de ISO. Actuellement, il est basé, pour ce qui est des couches inférieures, sur le modèle TCP-IP car celui-ci est beaucoup plus performant à ce niveau.

Il permet donc de communiquer avec n'importe quelle autre implémentation du modèle de référence en utilisant une passerelle entre TCP et la couche transport (cfr figure 39).

Dans la version 4 de ISODE se trouvent trois librairies où sont implémentés les différents éléments de service: ACSE, ROSE et RTSE [ISODE881].

Il nous semble intéressant, avant de regarder en détail les différentes routines, de préciser quelques uns des grands principes inhérents à l'architecture de ISODE.

- ISODE est développé en C.
- Les primitives d'application Request et Response sont des routines directement appelables, elles ont un nom de fonction.
- Les primitives d'application Indication et Confirmation sont, en réalité, implémentées par des modifications dans un des champs de la structure indication de la primitive Request ou Response initiatrice.
- Souvent, dans une routine C, au premier paramètre correspondra le descripteur de l'association courante, et au dernier, correspondra un pointeur vers la structure indication où s'indiqueront les divers événements.

Grille de présentation

Nous avons choisi de présenter les différentes routines de ISODE dans une grille où sont repris :

- Dans le première colonne, se trouve le nom de la primitive OSI.

- Dans la deuxième colonne, se trouve son nom dans ISODE-4.0.
- La troisième colonne donne le nom de la structure C modifiée lorsque tout s'est déroulé normalement.

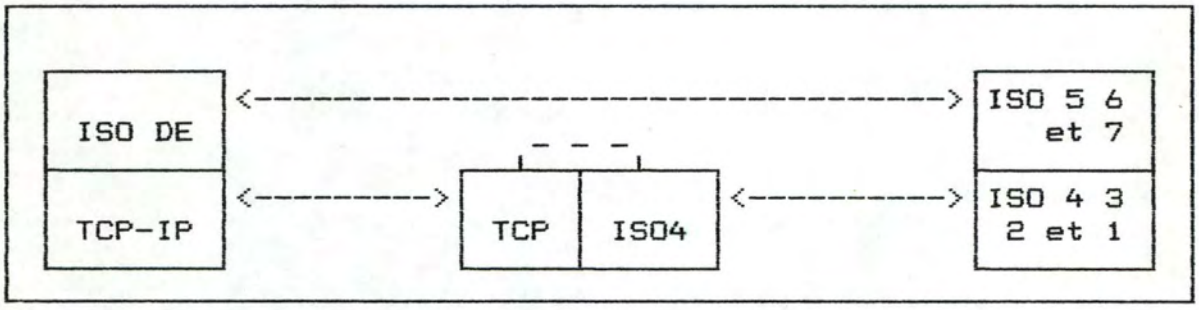

Figure 39 : ISODE

4.2 L'ACSE

4.2.1 Tableau des primitives ISO et routines ISODE

A C S E		
PRIMITIVE OSI	**ROUTINE ISODE**	**STRUCTURE MODIFIEE**
A-ASSOCIATE.req	AcAssocRequest	
A-ASSOCIATE.ind	(AcInit)	AcSAPstart
A-ASSOCIATE.resp	AcAssocResponse	
A-ASSOCIATE.conf		AcSAPconnect
A-RELEASE.req	AcRelRequest	
A-RELEASE.ind		AcSAPindication
A-RELEASE.resp	AcRelResponse	
A-RELEASE.conf		AcSAPrelease
A-P-ABORT.ind	AcABORTser	AcSAPindication
A-U-ABORT.req	AcUAbortRequest	
A-U-ABORT.ind	AcABORTser	AcSAPindication

4.2.2 Les routines ISODE et leurs paramètres

Nous allons maintenant reprendre l'ensemble des routines ISODE de l'ACSE et les présenter.

Notons que les types structurés sont repris en annexe.

4.2.2.1 AcAssocRequest

AcAssocRequest (context, callingtitle, calledtitle, callingaddr, calledaddr, ctxlist, defctxname, prequirements, srequirements, isn, settings, ref, data, ndata, qos, acc, aci)

```
/* Cette primitive implémente le A-ASSOCIATE.request        */

OID context;            /* le contexte application à utiliser */
AEI callingtitle,       /* inform. sur l'entité appl appelante*/
    calledtitle;        /* inform. sur l'entité appl appelée  */
struct PSAPaddr *callingaddr,/* l'adr. PSAP de l'appelant     */
                *calledaddr; /* l'adr. PSAP de l'appelé       */
struct PSAPctxlist *ctxlist; /* liste ctxt prés. proposés     */
OID defctxname;         /* le nom de contexte par défaut      */
int prequirements,      /* exigence du niveau présentation    */
    srequirements,      /* exigence du niveau session         */
    settings,           /* installation initiale des jetons   */
    ndata;              /* nbre de données du tabl. de données*/
long isn;               /* numéro de série initial            */
struct SSAPref *ref;    /* l'identf. de conn. de l'initiateur */
PE * data;              /* tableau des données de l'util.     */
struct qostype *qos;    /* qualité de service de la connexion */
struct AcSAPconnect *acc; /* modifié si l'appel est accepté*/
struct AcSAPindication *aci; /* modifié en cas de problème */
```

4.2.2 AcAssocResponse

AcAssocResponse (sd, status, reason, context, respondtitle, respondaddr, defctxresult, prequirements, srequirements, isn, settings, ref, data, ndata, aci)

```
/* Cette primitive implémente le A-ASSOCIATE.response       */

int sd;                 /* le descripteur de l'association    */
int status,             /* indicateur d'acceptation ou rejet  */
    reason;             /* le diagnostic                      */
OID context;            /* le contexte application à utiliser */
AEI respondtitle;       /* inform. sur l'entité appl répond.  */
struct PSAPaddr *respondaddr;/* l'adr. PSAP du répondeur      */
struct PSAPctxlist *ctxlist; /* liste ctxt prés. décidés      */
OID defctxresult;       /* le nom de contexte par défaut      */
int prequirements,      /* exigence du niveau présentation    */
    srequirements,      /* exigence du niveau session         */
long isn;               /* numéro de série initial            */
int settings;           /* installation initiale des jetons   */
struct SSAPref *ref;    /* l'identif. de conn. du répondeur   */
```

```
PE * data;              /* tableau des données de l'util.     */
ndata;              /* nbre de données du tabl. de données    */
struct AcSAPindication *aci; /* modifié en cas de problème */
```

4.2.2.3 AcInit

AcInit (vecp, vec, acs, aci)

```
                                                              */
/*   ISODE est basé sur TCP-IP.                               */
/*   Lorsque le démon de la couche transport (tsapd) a        */
/* accepté une connexion d'un hôte appelant, il consulte sa*/
/* base de données des services ISO développés               */
/* (/ETC/ISOSERVICES), pour déterminer quel programme, sur */
/* le système local, implémente l'action désirée.            */

/*   Ensuite, il exécute celui-ci avec tous les arguments  */
/* nécessaires. En fait, la première action effectuée par  */
/* ce programme est de recapturer l'état du dialogue ACSE  */
/* déjà entamé avec le démon.                                */

/*   Ceci est fait par la routine AcInit; on peut           */
/* considérer que, une fois qu'elle est exécutée, la        */
/* primitive Indication s'est produite.                      */

int vecp;                   /* la longueur du vecteur argument*/
char ** vec;                /* le vecteur argument            */
struct AcSAPstart *acs    /* modifié si l'appel est accepté */
struct AcSAPindication *aci; /* modifié en cas de problème */
```

4.2.2.4 AcRelRequest

AcRelRequest (sd, reason, data, ndata, acr, aci)

```
/* Cette primitive implémente le A-RELEASE.request          */

int sd,                 /* le descripteur de l'association    */
    reason;             /* la raison d'abandon de l'assoc.    */
PE * data;              /* tableau des données de l'util.     */
int ndata;              /* nbre de données du tabl. de données*/
struct AcSAPrelease *acr;     /* modifié si déconnexion       */
struct AcSAPindication *aci; /* modifié en cas de problème */
```

4.2.2.5 AcRelResponse

AcRelResponse (sd, status, reason, data, ndata, aci)

```
/* Cette primitive implémente le A-RELEASE.response         */

int sd,                 /* le descripteur de l'association    */
    status,             /* l'indicateur sur l'accord de décon.*/
    reason;             /* la raison d'abandon de l'assoc.    */
PE * data;              /* tableau des données de l'util.     */
int ndata;              /* nbre de données du tabl. de données*/
struct AcSAPindication *aci; /* modifié en cas de problème */
```

4.2.2.6 AcUAbortRequest

AcUAbortRequest (sd, data, ndata, aci)

```
/* Cette primitive implémente le A-U-ABORT.req                    */

int sd;                  /* le descripteur de l'association      */
PE * data;               /* tableau des données de l'util.       */
int ndata;               /* nbre de données du tabl. de données*/
struct AcSAPindication *aci; /* modifié en cas de problème */
```

4.2.2.7 AcABORTser

AcUAbortRequest (sd, pa, aci)

```
/* Cette primitive implémente le A-P-ABORT.ind                    */

int sd;                   /* le descripteur de l'association     */
struct PSAPabort *pa; /* pointeur vers structure PSAP            */
struct AcSAPindication *aci; /* modifié en cas de problème */
```

4.2.3 Comparaison avec le standard

4.2.3.1 Le A-ASSOCIATE.Request

A-ASSOCIATE.Request	
I S O	**I S O D E**
mode (X410 ou normal)	
contexte application	context
titre de PA appelant	callingtitle
id.d'invoc. de PA appelant	
qualif. de EA appelante	
id.d'invoc. de EA appelante	
titre de PA appelé	calledtitle
id.d'invoc. de PA appelé	
qualif. de EA appelée	
id.d'invoc. de EA appelée	
adresse prés. de l'appelant	callingaddr
adresse prés. de appelé	calledaddr
liste de contexte prés.	ctxlist
nom de cont. prés. par déf.	defctxname
qualité de service	qos
exigence présentation	prequirements
exigence session	srequirements
num. du point de sync. init	isn
distributions init. jetons	settings
identif. de conn. session	ref
données de l'utilisateur	data
	ndata
	acc
	Aci

4.2.3.2 Le A-ASSOCIATE.Response

A-ASSOCIATE.Response	
I S O	**I S O D E**
mode (X410 ou normal)	
contexte application	context
titre de PA répondant	respondtitle
id.d'invoc. de PA répondant	
qualif. de EA répondante	
id.d'invoc. de EA répond.	
adresse prés. du répondeur	respondaddr
liste de contexte prés.	ctxlist
nom de cont. prés. par déf.	defctxresult
exigence présentation	prequirements
exigence session	srequirements
num. du point de sync. init	isn
distributions init. jetons	settings
identif. de conn. session	ref
données de l'utilisateur	data
	ndata
	reason
	aci
	sd
résultat	status

4.2.3.3 Le A-RELEASE.Request

A-RELEASE.Request	
I S O	**I S O D E**
raison	reason
	sd
données de l'utilisateur	data
	ndata
	acr
	aci

4.2.3.4 Le A-RELEASE.Response

A-RELEASE.Response	
I S O	**I S O D E**
raison	reason
résultat	status
	sd
données de l'utilisateur	data
	ndata
	aci

4.2.3.5 Le A-U-ABORT.Request

A-U-ABORT.Response	
I S O	**I S O D E**
source	
	sd
données de l'utilisateur	data
	ndata
	aci

4.3 Le RTSE

4.3.1 Tableau des primitives ISO et routines ISODE

R T S E		
PRIMITIVE OSI	ROUTINE ISODE	STRUCTURE MODIFIEE
RT-OPEN.req	RtOpenRequest	
RT-OPEN.ind	(RtInit)	RtSAPstart
RT-OPEN.resp	RtOpenResponse	
RT-OPEN.conf		RtSAPconnect
RT-CLOSE.req	RtCloseRequest	
RT-CLOSE.ind		RtSAPindication
RT-CLOSE.resp	RtCloseResponse	
RT-CLOSE.conf		AcSAPrelease
RT-P-ABORT.ind		RtSAPindication
RT-U-ABORT.req	RtUAbortRequest	
RT-U-ABORT.ind		RtSAPindication

R T S E		
PRIMITIVE OSI	ROUTINE ISODE	STRUCTURE MODIFIEE
RT-TRANSFER.req	RtTransferRequest	
RT-TRANSFER.ind		RtSAPindication
RT-PLEASE-TURN.req	RtPTurnResquest	
RT-PLEASE-TURN.ind		RtSAPindication
RT-GIVE-TURN.req	RtGTurnRequest	
RT-GIVE-TURN.ind		RtSAPindication

4.3.2 Les routines ISODE et leurs paramètres

Nous allons maintenant reprendre l'ensemble des routines ISODE du RTSE et les présenter.

Notons que les types structurés sont repris en annexe.

4.3.2.1 RtOpenRequest

RtOpenRequest (mode, turn, context, callingtitle, calledtitle, callingaddr, calledaddr, ctxlist, defctxname, data, qos, rtc, rti)

```
/* Cette primitive implémente le RT-OPEN.request                    */

int mode,                  /* le mode du dialogue                   */
    turn;                  /* celui qui possède le tour initial     */
OID context;               /* le contexte application à utiliser    */
AEI callingtitle,          /* inform. sur l'entité appl appelante   */
    calledtitle;           /* inform. sur l'entité appl appelée     */
```

```
struct PSAPaddr *callingaddr,/* l'adr. PSAP de l'appelant  */
                *calledaddr; /* l'adr. PSAP de l'appelé    */
struct PSAPctxlist *ctxlist; /* liste ctxt prés. proposés  */
OID defctxname;       /* le nom de contexte par défaut      */
PE data;             /* tableau des données de l'util.      */
struct qostype *qos; /* qualité de service de la connexion */
struct RtSAPconnect *rtc; /* modifié si l'appel est accepté*/
struct RtSAPindication *aci; /* modifié en cas de problème */
```

4.3.2 RtOpenResponse

RtOpenResponse (sd, status, context, respondtitle, respondaddr, ctxlist, defctxresult, data, rti)

```
/* Cette primitive implémente le RT-OPEN.response      */

int sd;                /* le descripteur de l'association    */
int status;            /* indicateur d'acceptation ou rejet  */
OID context;           /* le contexte application à utiliser */
AEI respondtitle;      /* inform. sur l'entité appl répond.  */
struct PSAPaddr *respondaddr;/* l'adr. PSAP du répondeur     */
struct PSAPctxlist *ctxlist; /* liste ctxt prés. décidés     */
int defctxresult;      /* le nom de contexte par défaut      */
PE  data;              /* les données de l'utililisateur     */
struct RtSAPindication *aci; /* modifié en cas de problème */
```

4.3.2.3 RtInit

RtInit (vecp, vec, rts, rti)

```
                                                            */
/*   ISODE est basé sur TCP-IP.                             */
/*   Lorsque le démon de la couche transport (tsapd) a      */
/* accepté une connexion d'un hôte appelant, il consulte sa*/
/* base de données des services ISO développés             */
/* (/ETC/ISOSERVICES), pour déterminer quel programme, sur */
/* le système local, implémente l'action désirée.          */

/*   Ensuite, il exécute celui-ci avec tous les arguments  */
/* nécessaires. En fait, la première action effectuée par  */
/* ce programme est de recapturer l'état du dialogue RTSE  */
/* déjà entamé avec le démon.                              */


/*   Ceci est fait par la routine RtInit; on peut          */
/* considérer que, une fois qu'elle est exécutée, la       */
/* primitive Indication s'est produite.                    */

int vecp;                  /* la longueur du vecteur argument*/
char ** vec;               /* le vecteur argument            */
struct RtSAPstart *rts     /* modifié si l'appel est accepté */
struct RtSAPindication *rti; /* modifié en cas de problème */
```

4.3.2.4 RtCloseRequest

RtCloseRequest (sd, reason, data, acr, rti)

```
/* Cette primitive implémente le RT-CLOSE.request          */

int sd,                 /* le descripteur de l'association  */
    reason;             /* la raison d'abandon de l'assoc.  */
PE * data;              /* tableau des données de l'util.   */
struct AcSAPrelease *acr;     /* modifié si déconnexion     */
struct RtSAPindication *rti; /* modifié en cas de problème */
```

4.3.2.5 RtCloseResponse

RtCloseResponse (sd, reason, data, aci)

```
/* Cette primitive implémente le RT-CLOSE.response         */

int sd,                 /* le descripteur de l'association  */
    reason;             /* la raison d'abandon de l'assoc.  */
PE * data;              /* tableau des données de l'util.   */
struct RtSAPindication *aci; /* modifié en cas de problème */
```

4.3.2.6 RtUAbortRequest

RtUAbortRequest (sd, data, rti)

```
/* Cette primitive implémente le RT-U-ABORT.req            */

int sd;                 /* le descripteur de l'association  */
PE * data;              /* tableau des données de l'util.   */
struct RtSAPindication *rti; /* modifié en cas de problème */
```

4.3.2.7 RtTransferRequest

RtTransferRequest (sd, data, secs, rti)

```
/* Cette primitive implémente le RT-TRANSFER.req           */

int sd;                 /* le descripteur de l'association  */
PE * data;              /* tableau des données de l'util.   */
int secs;/* le nombre de secondes permis pour le transfert */
struct RtSAPindication *rti; /* modifié en cas de problème */
```

4.3.2.8 RtPTurnRequest

RtPTurnRequest (sd, priority, rti)

```
/* Cette primitive implémente le RT-PLEASE-TURN.req        */

int sd;                 /* le descripteur de l'association  */
int priority; /* priorité de l'action à effectuer          */
struct RtSAPindication *rti; /* modifié en cas de problème */
```

4.3.2.9 RtGTurnRequest

RtGTurnRequest (sd, rti)

```
/* Cette primitive implémente le RT-GIVE-TURN.req          */

int sd;                 /* le descripteur de l'association    */
struct RtSAPindication *rti; /* modifié en cas de problème */
```

4.3.2.10 RtWaitRequest

RtWaitRequest (sd, secs, rti)

```
/* La routine RtWaitRequest est utilisée pour attendre     */
/* soit une répose soit une demande d'un autre utilisateur.*/

/* A l'encontre des autres routines, elle renvoie soit la  */
/* valeur OK (si une lecture a été faite, soit la valeur   */
/* NOTOK (en cas d'échec), soit la valeur DONE (si quelque */
/* chose d'autre s'est produit).                           */


int sd;                 /* le descripteur de l'association    */
int secs;/* le nombre de secondes permis pour le transfert */
struct RtSAPindication *rti; /* modifié en cas de problème */
```

4.3.2.11 RtSetindication

RtSetindication (sd, indication, rti)

```
/* Cette routine fournit une interface asynchrone.         */

int sd;                 /* le descripteur de l'association    */
IFP indication; /* l'adresse de la routine de trt d'évén.  */
struct RtSAPindication *rti; /* modifié en cas de problème */
```

4.2.2.12 RtSelectMask

RtSelectMask (sd, mask, nfds, rti)

```
/* Cette routine fournit une possiblité de pouvoir gérer   */
/* plusieurs descripteurs d'associations simultanément.    */

int sd;                 /* le descripteur de l'association    */
fd_set *mask; /* un masque de descripteur de fichier       */
              /* compréhensible par xselect.               */
int *nfds; /* le lieu d'une valeur d'un entier pour xselect*/
struct RtSAPindication *rti; /* modifié en cas de problème */
```

4.3.3 Comparaison avec le standard

4.3.3.1 Le RT-OPEN.Request

RT-OPEN.Request	
I S O	**I S O D E**
mode du dialogue	mode
tour initial	tour
protocole application	
mode (normal - X410)	
contexte application	context
titre de PA appelant	callingtitle
id.d'invoc. de PA appelant	
qualif. de EA appelante	
id.d'invoc. de EA appelante	
titre de PA appelé	calledtitle
id.d'invoc. de PA appelé	
qualif. de EA appelée	
id.d'invoc. de EA appelée	
adresse prés. de l'appelant	callingaddr
adresse prés. de appelé	calledaddr
liste de contexte prés.	ctxlist
nom de cont. prés. par déf.	defctxname
	qos
données de l'utilisateur	data
	rtc
	rti

4.3.3.2 Le RT-OPEN.Response

RT-OPEN.Response	
I S O	**I S O D E**
	sd
contexte application	context
titre de PA répondant	respondtitle
id.d'invoc. de PA répondant	
qualif. de EA répondante	
id.d'invoc. de EA répond.	
résultat	status
diagnostic	
adresse prés. du répondeur	respondaddr
liste de contexte prés.	ctxlist
nom de cont. prés. par déf.	defctxresult
données de l'utilisateur	data
	rti

4.3.3.3 Le RT-CLOSE.Request

RT-CLOSE.Request	
I S O	**I S O D E**
raison	reason
	sd
données de l'utilisateur	data
	acr
	rti

4.3.3.4 Le RT-CLOSE.Response

RT-CLOSE.Response	
I S O	**I S O D E**
raison données de l'utilisateur	reason sd data rti

4.3.3.5 Le RT-TRANSFER.Request

RT-TRANSFER.Req	
I S O	**I S O D E**
APDU temps de transfert max.	data sd secs rti

4.3.3.6 Le RT-TRANSFER.Confirmation

RT-TRANSFER.Conf	
I S O	**I S O D E**
APDU résultat	**non réalisé dans ISODE**

4.3.3.7 Le RT-TURN-PLEASE.Request

RT-TURN-PLEASE.Req	
I S O	I S O D E
priorité	priority sd rti

4.3.3.8 Le RT-TURN-GIVE.req

RT-TURN-GIVE.Req	
I S O	I S O D E
	sd rti

4.3.3.9 Le RT-U-ABORT.Request

RT-U-ABORT.Response	
I S O	I S O D E
données de l'utilisateur	sd data rti

4.4 Le ROSE

4.4.1 Tableau des primitives ISO et routines ISODE

R O S E		
PRIMITIVE OSI	ROUTINE ISODE	STRUCTURE MODIFIEE
RO-INVOKE.req	RoInvokeRequest	
RO-RESULT.req	RoResultRequest	
RO-ERROR.req	RoErrorRequest	
RO-INVOKE.ind		RoSAPindication
RO-RESULT.ind		RoSAPindication
RO-ERROR.ind		RoSAPindication
RO-REJECT-U.req	RoURejectRequest	
RO-REJECT-U.ind		RoSAPindication
RO-REJECT-P.ind		RoSAPindication

4.4.2 Les routines ISODE et leurs paramètres

Nous allons maintenant reprendre l'ensemble des routines ISODE du RTSE et les présenter.

Notons que les types structurés sont repris en annexe.

4.4.2.1 RoInvokeRequest

RoInvokeRequest (sd, op, class, args, invoke, linked, priority, roi)

```
/* Cette primitive implémente le RO-INVOKE.request          */

int sd;         /* le descripteur de l'association          */
int op,         /* le numéro de l'opération à exécuter      */
    class,      /* la classe de l'opération, cfr modèle     */
    invoke,     /* l'identificateur de cette requête        */
    *linked,    /* l'identificateur lié à cette requête     */
    priority;   /* la priorité de cette requête             */
PE args;        /* les arguments de l'opération             */
struct RoSAPindication *roi; /* modifié si opér. synchrone */
                             /* ou modifié en cas d'échec  */
```

4.4.2.2 RoResultRequest

RoResultRequest (sd, invoke, op, result, priority, roi)

```
/* Cette primitive implémente le RO-RESULT.request          */

int sd;         /* le descripteur de l'association          */
int op,         /* le numéro de l'opération exécutée        */
    invoke,     /* l'identificateur de la requête corresp.  */
    priority;   /* la priorité de cette requête             */
PE result;      /* les résultats de l'opération             */
struct RoSAPindication *roi; /* modifié en cas de problème */
```

4.4.2.3 RoErrorRequest

RoErrorRequest (sd, invoke, error, params, priority, roi)

```
/* Cette primitive implémente le RO-ERROR.request           */

int sd;         /* le descripteur de l'association          */
int invoke,     /* l'identificateur de la requête corresp.  */
    error,      /* le code erreur (prédefini)               */
    priority;   /* la priorité de cette requête             */
PE params;      /* les paramètres de l'erreur               */
struct RoSAPindication *roi; /* modifié en cas de problème */
```

4.4.2.4 RoURejectRequest

RoURejectRequest (sd, invoke, reason, priority, roi)

```
/* Cette primitive implémente le RO-U-REJECT.req            */

int sd;                 /* le descripteur de l'association  */
int *invoke,/* un pteur vers ident. de la requête corresp.  */
    reason,     /* un des codes de rejet non fatal          */
    priority;   /* la priorité de cette requête             */
struct RtSAPindication *roi; /* modifié en cas de problème */
```

4.4.2.5 RoWaitRequest

RoWaitRequest (sd, secs, roi)

```
/* La routine RoWaitRequest est utilisée pour attendre     */
/* soit une répose soit une demande d'un autre utilisateur.*/

/* A l'encontre des autres routines, elle renvoie soit la  */
/* valeur OK (si une lecture a été faite, soit la valeur   */
/* NOTOK (en cas d'échec), soit la valeur DONE (si quelque */
/* chose d'autre s'est produit).                           */


int sd;                 /* le descripteur de l'association    */
int secs;/* le nombre de secondes permis d'attente de don. */
struct RtSAPindication *rti; /* modifié en cas de problème */
```

4.4.2.6 RoSetindication

RoSetindication (sd, indication, rti)

```
/* Cette routine fournit une interface asynchrone.         */

int sd;                 /* le descripteur de l'association    */
IFP indication; /* l'adresse de la routine de trt d'évén.  */
struct RtSAPindication *rti; /* modifié en cas de problème */
```

4.4.2.7 RoSelectMask

RoSelectMask (sd, mask, nfds, rti)

```
/* Cette routine fournit une possiblité de pouvoir gérer   */
/* simultanément plusieurs descripteurs d'associations.    */

int sd;                 /* le descripteur de l'association    */
fd_set *mask; /* un masque de descripteur de fichier       */
              /* compréhensible par xselect.               */
int *nfds; /* le lieu d'une valeur d'un entier pour xselect*/
struct RtSAPindication *rti; /* modifié en cas de problème */
```

4.4.3 Comparaison avec le standard

4.4.3.1 Le RO-INVOKE.Request

RO-INVOKE.Req	
I S O	**I S O D E**
	sd
opération	op
classe de l'opération	class
argument	args
indentif. de requête	invoke
identificateur lié	linked
priorité	priority
	roi

4.4.3.2 Le RO-RESULT.Request

RO-RESULT.Req	
I S O	**I S O D E**
	sd
	op
résultat	result
indentif. de requête	invoke
priorité	priority
	roi

4.4.3.3 Le RO-ERROR.Request

RO-ERROR.Req	
I S O	**I S O D E**
	sd
erreur	error
paramètres	params
indentif. de requête	invoke
priorité	priority
	roi

4.4.3.4 Le RO-U-REJECT.Request

RO-U-REJECT.Req	
I S O	**I S O D E**
	sd
raison	reason
indentif. de requête	invoke
priorité	priority
	roi

4.5 Le CCRSE

Cet élément de service n'est pas développé dans la version 4 de ISODE.

CHAPITRE 5 : EXEMPLE D'UTILISATION DES ASE GRACE A UNE METHODOLOGIE UTILISANT LES OPERATIONS A DISTANCE.

5.1 Introduction

Etant donné l'importance du langage de syntaxe abstraite ASN.1 dans la méthodologie des opérations à distance, nous commencerons ce chapitre par une brève synthèse de ses principales caractéristiques (5.2). Nous exposerons ensuite la méthodologie de développement d'applications distribuées, proposée par les concepteurs du programme ISODE, ainsi que les outils sous-jacents (5.3 et 5.4).

Nous l'appliquerons enfin dans un exemple (5.5, 5.6, 5.7 et 5.8).

5.2 Le langage ASN.1

5.2.1 Objectifs du langage

L'OSI a donc pour objectif (comme nous l'avons expliqué au chapitre 2) de proposer un certain nombre de protocoles permettant l'interconnexion de différents ordinateurs voulant coopérer à une tâche commune. Pour cela, elle définit des protocoles d'échange d'information.

Hélas, l'emploi d'une langue classique peut amener des imprécisions ou des ambiguïtés qui seront probablement traitées différemment selon les concepteurs de programmes distribués. Ceci conduira alors inévitablement à des confusions lors de leurs exécutions.

Pour pallier à ce problème, il est envisagé d'utiliser des langages formels de spécification de définition des protocoles d'échange.

C'est dans ce cadre que vient se positionner un langage tel que l'Abstract Syntax Notation One (ASN.1) [ISO8824], [CCITTSYN] et [BROSSEL87]. Il permet une description des données utilisées par les différents systèmes informatiques indépendamment des représentations physiques réelles.

De plus, ce langage offre la possibilité de décrire non seulement le type des données manipulées d'une façon abstraite (la notation ASN.1), mais définit également la représentation qu'il faudra utiliser pour l'échange de ces données (le codage T-L-V).

Nous sommes conscients de la brièveté de la synthèse faite ici sur ce langage. Toutefois, le lecteur intéressé peut se référer à [BROSSEL87], à [ISO8824].

Un exemple de déclaration en ASN.1 est donné à la figure 40.

5.2.2 La représentation des données : le codage T-L-V

Le principe est d'associer à chaque valeur à coder, un ensemble unique d'octets.

Pour une même valeur précise, il faudra définir son type, sa longueur et la valeur elle-même.

Le type est défini par sa classe (universelle, application, contexte ou privée), sa forme (primitive ou constructrice) et son identificateur, le tout sur un ou plusieurs octets.

Le type UNIVERSEL référence un type standardisé.

Le type APPLICATION référence un type défini pour toute une application.

Le type CONTEXTE est valable uniquement dans le contexte d'une application.

Le type PRIVE est utilisé par telle personne ou telle autre et n'a de sens que pour ces personnes.

La longueur est codée sur un ou plusieurs octets. Elle peut parfois être indéfinie.

La valeur est définie en respectant la règle de codage de son type.

5.2.3 La notation ASN.1

Celle-ci permet de définir le type des données échangées, sous une forme abstraite.

Un certain nombre de types sont prédéfinis : Boolean, Integer, Bit String, Octet String et Null.

Le langage offre aussi des constructeurs de types (Sequence, Set, Choice) ainsi qu'une possibilité d'étiquetage (les Tags).

Différents types standards ont également été définis étant donné leur fréquente utilisation : IA5String, NumericString, PrintableString, T.61String, VideoString et UTCTime.

5.2.4 La syntaxe du langage ASN.1

- Les blancs sont ignorés.
- Les mots clés sont écrits en lettres majuscules.
- Les noms de règles (types et valeurs) débutent par une majuscule.
- Les noms des étiquettes commencent par une minuscule.
- Un nom est un caractère suivi de un ou plusieurs signes alphanumériques.
- Les chaînes de caractères sont entourées de ".
- Les nombres peuvent prendre deux formes:
  - une chaîne de caractères chiffre
  - une chaîne de caractères entourée de ' et suivie par B,b,H ou h.

```
PERSONNEL DEFINITION ::=
BEGIN

PersonnelRecord ::=
   [APPLICATION 0]
     IMPLICIT SET {
         Name,

       title[0]
         IA5String,

         EmployeeNumber,

       dateofHire[1]
         Date,

       nameofSpouse[2]
         Name,

      [3]IMPLICIT SEQUENCE OF ChildInformation DEFAULT {}
   }

ChildInformation ::=
   SET {
         Name,

       dateofBirth[0]
         Date
   }

Name ::= [APPLICATION 1] IMPLICIT SEQUENCE {

       givenName
         IA5String,

       initial
         IA5String,

       familyName
         IA5String
}

EmployeeNumber ::= [APPLICATION 2] IMPLICIT INTEGER

Date ::= [APPLICATION 3] IMPLICIT IA5String

END
```

Figure 40 : Exemple de spécification en ASN.1

Explication de l'exemple

L'objectif est de définir rigoureusement ce qu'est une personne de type PERSONNEL.

Nous définissons un élément de type PersonnelRecord; nous identifierons ce type par APPLICATION 0. Il s'agit d'un ensemble de données comprenant :

- un nom de type Name (précisé en dessous)
- un titre (title) identifié par CONTEXTE 0
- un numéro de type EmployeeNumber (précisé en dessous)
- une date d'engagement de type Date (à préciser), identifié par CONTEXTE 1
- un nom d'épouse de type Name identifié par CONTEXTE 2
- une séquence d'informations sur les enfants (de type ChildInformation) identifiée par CONTEXTE 3. Cette séquence est vide si l'employé n'a pas d'enfant.

Précisons maintenant les types non-définis.

- Les informations sur les enfants sont composées d'un ensemble de données :
  - le nom, de type Name.
  - la date de naissance, de type Date et identifiée par CONTEXTE 0.
- Un nom de type Name, identifié par APPLICATION 1, comprend une suite ordonnée de :
  - prénom (givenName), de type IA5String.
  - initiale (initial), de type IA5String.
  - nom de famille (familyName), de type IA5String.
- Un numéro d'employé (étiquette APPLICATION 2) est de type entier.
- Une date (étiquette APPLICATION 3) est de type IA5String.

Nous avons ainsi spécifié complètement le type PersonnelRecord en n'utilisant que des types primitifs ou constructeurs.

5.3 Exposé de la méthodologie

La méthodologie exposée ci-dessous est celle proposée par les concepteurs du programme ISODE [ISODE884]. Elle provient initialement de [ECMATR/31].

Son objectif est de simplifier le développement d'application utilisant le modèle de référence ISO.

Elle se base sur l'exécution d'opérations à distance (ROSE) et comprend six étapes que nous résumons ci-après.

Parallèlement, nous développerons, en guise d'exemple, une application utilisant la plupart des notions présentées.

5.3.1 Définir le nouveau service

Il faut que le concepteur précise les informations relatives au nom et à l'adressage de l'application. Notamment,

- la syntaxe abstraite : celle-ci définit les structures de données qui seront échangées par le service.
- le nom du protocole application utilisé.
- le nom de l'entité sur le réseau.
- l'adresse présentation de l'entité.
- le nom du programme sur le système local qui implémente le service.

exemple

L'application que nous allons développer a comme objectif de permettre à un utilisateur du modèle de référence ISO d'un ordinateur de pouvoir demander qu'un message soit transmis à un autre utilisateur via le courrier électronique X400 qui se trouve dans un troisième site où se trouve également une implémentation du modèle de référence.

L'implémentation du modèle de référence que nous utiliserons est le programme ISODE [ISODE] et celle du courrier électronique est le programme EAN [EAN].

Ceci est schématisé à la figure 41.

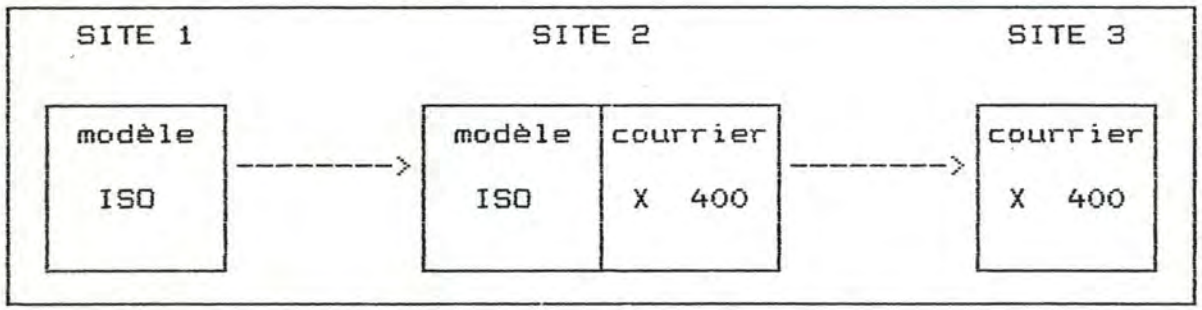

Figure 41 : schématisation de l'application

- la syntaxe abstraite utilisée : L'ASN.1

- le nom du protocole application utilisé : roseanstore

- le nom de l'entité sur le réseau : "tsap/roseanstore"

- l'adresse transport de l'entité : #525

- le nom du programme sur le système local : ros.rosean

5.3.2 Définir un module d'opérations à distance

Un module d'opérations à distance définit les opérations (et erreurs associées) ainsi que la syntaxe abstraite des structures de données échangées par le service.

Il est écrit en notation ASN1, en utilisant la notation des opérations définie dans le protocole ROSE.

Le programme ROSY lit la description d'un tel module et produit les fichiers en langage C correspondants, qui sont utilisés par le concepteur pour le développement du service.

exemple

Une seule opération est à définir pour notre application. C'est celle qui permet de demander l'exécution de l'envoi d'un message en utilisant EAN.

Elle est déclarée dans le module d'opération à distance à la figure 42.

```
ROSEAN DEFINITIONS ::=

BEGIN

 -- operation

 ean OPERATION
     ARGUMENT    IA5List -- l'adresse et le message
     RESULT      Empty
     ::= 0

 -- types

IA5List ::= SEQUENCE OF IA5String

Empty    ::= NULL


END
```

Figure 42: Le fichier rosean.ry

5.3.3 Définir les structures de données concrètes

Un module de syntaxe abstraite définit les structures de données échangées par le service. Il est plus ou moins équivalent à un module d'opération à distance où ne se trouverait pas la définition des opérations et des erreurs.

Ce module de syntaxe abstraite est généré automatiquement par le programme ROSY.

Le programme POSY lit la description d'un tel module et produit les définitions des structures C correspondantes.

exemple

Le module généré se trouve à la figure 43.

```
ROSEAN DEFINITIONS ::=

BEGIN

IA5List ::= SEQUENCE OF IA5String

Empty    ::= NULL

END
```

Figure 43: Le module généré par ROSY

5.3.4 Construire un initiateur

Il faut, ensuite, construire le programme initiateur du service, c'est-à-dire le programme qui est l'initiateur de l'application. Celui-ci peut être interactif ou emboîté.

5.3.4.1 Un initiateur emboîté

Un initiateur emboîté est caractérisé par le fait qu'il dirige complètement une association et les opérations à effectuer avec un hôte éloigné.

Quatre parties distinctes peuvent être relevées: l'établissement de l'association, la demande d'exécutions d'opérations à distance, la déconnexion et le traitement d'erreur.

Ce programme utilisera les morceaux de texte C générés par ROSY et POSY.

. établissement de l'association

La routine AcAssocRequest peut être utilisée pour l'établissement. On utilise ainsi l'élément de service ACSE défini par ISO et implémenté dans la version 4 de ISODE.

. demande d'exécutions d'opérations à distance

Les opérations pourront être exécutées de façon synchrone ou asynchrone. Dans les deux cas, nous utiliserons les routines de l'élément de service d'applications ROSE.

. déconnexion

Nous pouvons de nouveau utiliser les services de ACSE et, en particulier, la routine AcRelRequest.

. traitement d'erreurs

Le traitement des erreurs est à prévoir dans n'importe quel programme.

5.3.4.2 Un initiateur interactif

Dans ce cas-ci, l'utilisateur exécute un programme interactif. Il dirige, comme il le veut, les opérations à distance à effectuer.

Le programmeur peut faire appel à la routine ryinitiator (5.5.3.1) fournie par ISODE, pour simplifier son programme. Cette routine est un initiateur interactif générique.

exemple

Dans notre application, nous utiliserons cet initiateur pour la communication.

Nous reprenons à la figure 44 la procédure principale (main) de l'initiateur de notre application.

```
main(argc, argv, envp)

int     argc;
char    **argv,
        **envp;
{
    (void) ryinitiator (argc, argv, myservice, mycontext,
                        mypci, table_ROSEAN_operation,
                        dispatches, do_quit)
}
```

Figure 44: La fonction main de rosean

Argc et argv sont les paramètres transmis lors de l'appel de l'initiateur.

Myservice, mycontext et mypci sont les variables définissant l'application. Elles sont initialisées au début de programme initiateur.

La variable table_ROSEAN_operation est la table des opérations à distance possibles, construite par le compilateur ROSY.

La variable dispatches est une table nécessaire à ryinitiator; elle reprend l'ensemble de ce que peut faire l'initiateur. Elle est définie au début du programme.

La variable do_quit est l'adresse d'une routine à exécuter lorsque l'application est terminée.

exemple : L'initiateur de l'application, rosean

Nous allons maintenant donner l'algorithme de rosean, l'initiateur de notre application. Ce programme est fourni en annexe.

1°) Les fichiers à inclure

Il faut inclure les fichiers de définition : ryinitiator.h, ROSEAN-ops.h, ROSEAN-types.h

2°) Initialiser les variables de base

C'est à dire définir les trois variables suivantes : myservice, mycontext et mypci.

3°) Définir le type des fonctions de rosean: do_ean, ean_result, do_help, do_quit.

4°) Définir la table dispatches utilisée par ryinitiator. Celle-ci comprend les fonctions définies ci-dessus.

5°) Définir l'ensemble des fonctions de rosean.

- la fonction main (cfr figure 44)
- la fonction do_ean qui comprend l'ensemble des instructions à effectuer avant l'exécution de l'opération à distance ean.
- la fonction do_help assurant l'aide à l'utilisateur de rosean.
- la fonction do_quit qui exécute les instructions de fermeture après l'exécution de l'opération à distance.
- la fonction ean_result exécute les instructions à effectuer après l'exécution de l'opération à distance.

5.3.5 Construire un répondeur

Il faut également construire le programme répondeur. Deux approches pour sa réalisation sont également possibles : statiquement (faire correspondre à chaque nouvelle association, une simple instanciation pré-existante du programme répondeur) et dynamiquement (créer une nouvelle instanciation du programme lors de chaque appel).

Ce programme utilisera également les morceaux de textes C générés par ROSY et POSY.

Trois parties apparaissent dans ce genre de programme : la gestion de l'association, la réponse aux opérations et le traitement d'erreur.

5.3.5.1 La gestion de l'association

On peut utiliser la routine ryresponder qui se trouve dans le programme ISODE. C'est un serveur générique qui s'occupe tant de l'approche statique que dynamique.

5.3.5.2 La réponse aux opérations

En cas de succès (c'est-à-dire que l'opération s'est déroulée normalement), on peut utiliser la routine RyDsResult; en cas d'échec, la routine RyDsUreject.

5.3.5.3 Le traitement d'erreurs

Il est à prévoir dans tout programme...

exemple

Nous utiliserons aussi ce répondeur générique de ISODE pour répondre à ryinitiator().

La figure 45 représente la fonction principale du répondeur de l'application (main).

```
main(argc, argv, envp)

int     argc;
char    **argv,
        **envp;
{
    (void) ryresponder (argc, argv, PLocalHostName (),
                        myservice, dispatches,
                        table_ROSEAN_operation, NULLIFP,
                        NULLIFP)
}
```

Figure 45 : la fonction main de roseand

Argç et argv sont les paramètres transmis lors de l'appel du répondeur.

PLocalHostName est une fonction donnant le nom de l'hôte local.

Myservice est la variable définissant l'application. Elle est initialisée au début de programme répondeur et est la même que pour l'initiateur.

La variable table_ROSEAN_operation est la table des opérations à distance possibles, construite par le compilateur ROSY.

La variable dispatches est une table nécessaire à ryresponder. Elle reprend l'ensemble de ce que peut faire le répondeur. Elle est définie au début du programme.

Les variables do_start et do_quit sont les adresses de routines à exécuter lorsque l'application commence ou se termine. Dans notre application,, nous n'avons pas eu le besoin de telles routines. C'est pourquoi la valeur des paramètres est NULLIFP.

exemple : Le répondeur de l'application, roseand

Nous allons maintenant donner l'algorithme de roseand, le répondeur de notre application. Ce programme est fourni en annexe.

1°) Les fichiers à inclure

Il faut inclure les fichiers de définition :
ryresponder.h, ROSEAN-ops.h,
ROSEAN-types.h

2°) Initialiser la variable de base myservice avec la même valeur que celle donnée dans l'initiateur.

3°) Définir le type des fonctions de roseand: op_ean. En effet, notre application ne permet l'exécution que d'une seule opération à distance : exécuter l'envoi d'un message en utilisant ean.

4°) Définir la table dispatches utilisée par ryresponder. Celle-ci comprend les fonctions définies ci-dessus (ean, en l'occurrence).

5°) Définir l'ensemble des fonctions de roseand.

- la fonction main (cfr figure 45)
- la fonction op_ean qui exécute l'opération sur le site répondeur.
- les fonctions error et ureject qui traitent les problèmes. Elles sont recopiées telles quelles des autres initiateurs.

5.3.6 Rendre le tout cohérent

Il faudra finalement avoir une politique solide pour les noms de fichiers, les règles du make (utilitaire de gestion de programmes)...

exemple

le programme initiateur de l'application s'appellera **rosean.**

Le programme répondeur s'appellera **roseand.**

Le Makefile utilisé pour cette application se trouve en annexe.

5.4 Outils de support à la méthodologie

5.4.1 Le programme ROSY

5.4.1.1 Le module d'opérations à distance

Le programme ROSY lit une description du module d'opérations à distance et produit les textes C correspondants.

Ce module comprend trois parties : les définitions des opérations, la définition des erreurs et la définition des types de données. Il est écrit selon la notation ROSE [ROSE88].

5.4.1.1.1 Définition des opérations

Les opérations sont définies par un nom commençant par une minuscule, suivi du mot opération.

Ensuite peuvent apparaître :

ARGUMENT: pour définir les types de données ASN.1 que l'opération attend comme argument.

RESULTS : pour définir les types des résultats ASN.1 que l'opération donne en cas de succès.

ERRORS : pour définir les erreurs que l'opération peut renvoyer en cas d'échec.

Finalement, le signe ::= et un entier identifiant l'opération dans le module sont à indiquer.

5.4.1.1.2 Définition des erreurs

Les erreurs sont définies par un nom commençant par une minuscule, suivi du mot ERROR.

Ensuite peut apparaître :

PARAMETRE : pour définir le type de données ASN.1 que l'erreur retourne.

Finalement, le signe ::= et un entier identifiant l'erreur dans le module sont à indiquer.

5.4.1.1.3 Définition des types de données

Les types de données sont définis par un nom commençant par une majuscule, suivi du signe ::= et de la définition du type en ASN.1.

5.4.1.2 Résultats du programme ROSY

Le programme ROSY fournit quatre fichiers en résultats.

5.4.1.2.1 Un module de syntaxe abstraite

Le module de syntaxe abstraite produit par ROSY est simplement une copie des définitions de type du module d'opérations à distance.

exemple

Le fichier généré pour notre application se trouve à la figure 46.

```
ROSEAN DEFINITIONS ::=

BEGIN

  IA5List ::= SEQUENCE OF IA5String

  Empty   ::= NULL

END
```

Figure 46 : Le fichier rosean.py

5.4.1.2.2 Un fichier pré-processeur C

Celui-ci contient les définitions de chaque code opération, de chaque code erreur et de chaque morceau de routine.

exemple

Le fichier généré pour notre application se trouve à la figure 47.

Il contient la définition de la table table_ROSEAN_operation qui sera utilisée par le répondeur et l'initiateur génériques pour connaître l'ensemble des opérations exécutables à distance.

```
/* automatically generated by rosy 4.0 #3 (alma) */

#include "rosy.h"


                                        /* OPERATIONS */

extern struct RyOperation table_ROSEAN_Operations[];

                                        /* OPERATION ean */
#define operation_ROSEAN_ean    0

#ifdef  INVOKER
#define encode_ROSEAN_ean_argument      encode_ROSEAN_IA5Li
#define decode_ROSEAN_ean_result        decode_ROSEAN_Empty
#define free_ROSEAN_ean_result  free_ROSEAN_Empty
#else
#define encode_ROSEAN_ean_argument      NULLIFP
#define decode_ROSEAN_ean_result        NULLIFP
#define free_ROSEAN_ean_result  NULLIFP
#endif

#ifdef  PERFORMER
#define decode_ROSEAN_ean_argument   decode_ROSEAN_IA5List
#define free_ROSEAN_ean_argument     free_ROSEAN_IA5List
#define encode_ROSEAN_ean_result     encode_ROSEAN_Empty
#else
#define decode_ROSEAN_ean_argument      NULLIFP
#define free_ROSEAN_ean_argument        NULLIFP
#define encode_ROSEAN_ean_result        NULLIFP
#endif


#ifndef lint

#define stub_ROSEAN_ean(sd,id,in,rfx,efx,class,roi)\
RyStub ((sd),table_ROSEAN_Operations,operation_ROSEAN_ean,
       (id), NULLIP,(caddr_t) (in), (rfx), (efx), (class),
       (roi))

#define op_ROSEAN_ean(sd,in,out,rsp,roi)\
RyOperation ((sd), table_ROSEAN_Operations,
             operation_ROSEAN_ean,
             (caddr_t) (in), (out), (rsp), (roi))
#endif


                                        /* ERRORS */

extern struct RyError table_ROSEAN_Errors[];
```

Figure 47 : Le fichier ROSEAN-ops.h

5.4.1.2.3 Un fichier de définitions des structures de données en langage C

Ce fichier contient une table des opérations et une table des erreurs. Celles-ci seront utilisées au moment de l'exécution du service de l'entité application.

exemple

Le fichier généré pour notre application se trouve à la figure 48.

```
/* automatically generated by rosy 4.0 #3 (alma) */

#include <stdio.h>
#include "ROSEAN-ops.h"


#include "ROSEAN-types.h"

                                    /* OPERATIONS */

                                    /* OPERATION ean */
int     encode_ROSEAN_IA5List (),
        decode_ROSEAN_IA5List (),
        free_ROSEAN_IA5List ();
int     encode_ROSEAN_Empty (),
        decode_ROSEAN_Empty (),
        free_ROSEAN_Empty ();

struct RyOperation table_ROSEAN_Operations[] = {
                                    /* OPERATION ean */
    "ean", operation_ROSEAN_ean,
        encode_ROSEAN_ean_argument,
        decode_ROSEAN_ean_argument,
        free_ROSEAN_ean_argument,
        1, encode_ROSEAN_ean_result,
           decode_ROSEAN_ean_result,
           free_ROSEAN_ean_result,
        errors_ROSEAN_ean,

    NULL
};


                                          /* ERRORS */

struct RyError table_ROSEAN_Errors[] = {
};
```

Figure 48 : Le fichier ROSEAN-ops.c

5.4.1.2.4 Un fichier à utiliser pour la détection d'erreur de compilation

Ce fichier peut servir pour la recherche d'erreurs lors de la construction d'applications.

exemple

Le fichier généré pour notre application se trouve à la figure 49.

```
/* automatically generated by rosy 4.0 #3 (alma) */

#include <stdio.h>
#include "ROSEAN-ops.h"


#include "ROSEAN-types.h"

                                    /* OPERATIONS */

                                    /* OPERATION ean */
int     encode_ROSEAN_IA5List (),
        decode_ROSEAN_IA5List (),
        free_ROSEAN_IA5List ();
int     encode_ROSEAN_Empty (),
        decode_ROSEAN_Empty (),
        free_ROSEAN_Empty ();

struct RyOperation table_ROSEAN_Operations[] = {
                                    /* OPERATION ean */
    "ean", operation_ROSEAN_ean,
        encode_ROSEAN_ean_argument,
        decode_ROSEAN_ean_argument,
        free_ROSEAN_ean_argument,
        1, encode_ROSEAN_ean_result,
           decode_ROSEAN_ean_result,
           free_ROSEAN_ean_result,
        errors_ROSEAN_ean,

    NULL
};


                                              /* ERRORS */

struct RyError table_ROSEAN_Errors[] = {
};
```

Figure 49 : Le fichier ROSEAN-stubs.c

5.4.1.3 Exécution du programme ROSY

Par convention, le nom du fichier argument du programme ROSY a le suffixe .ry.

ex: rosy module.ry

Les fichiers résultats se répartissent en :

- module.py, le module de syntaxe abstraite (5.4.2.1)
- MODULE-ops.h, le module de définition des opérations (5.4.2.2)
- MODULE-ops.c, le module de définition des tables (5.4.2.3)
- MODULE-stubs.c, le module de définition pour lint (5.4.2.4)

5.4.1.4 Pour plus de renseignements

Le programme ROSY est présenté dans le chapitre 4 de [ISODE4].

5.4.2 Le programme POSY

5.4.2.1 Le module de syntaxe abstraite

Le programme POSY est un compilateur de syntaxe abstraite.

Il lit une description d'un module de syntaxe abstraite et génère les structures C correspondantes ainsi qu'un module de syntaxe abstraite augmenté.

La syntaxe de ce module argument correspondant à la notation ASN.1 est définie dans la norme X409 du CCITT.

exemple

Dans notre application, nous avons défini deux nouveaux types : IA5List et Empty.
Dès-lors, il est nécessaire de construire des fonctions de codage et décodage de ces types puisque notre application ne pourra pas utiliser seulement celles pré-existantes, déjà construites dans ISODE 4-0.

Celles-ci se trouvent en annexe.

5.4.2.2 Résultats du programme POSY

Ce programme fournit deux fichiers en résultat.

5.4.2.2.1 Les structures en langage C

Pour chaque nouveau type de données défini dans le module de syntaxe abstraite, le programme générera une structure en langage C (struct).

exemple

Le fichier généré pour notre application se trouve à la figure 50.

```
/* automatically generated by posy 4.0 #4 (alma) */

#ifndef _module_ROSEAN_defined_
#define _module_ROSEAN_defined_

#include "psap.h"
#ifndef PEPYPATH
#define PEPYPATH
#endif
#include "../pepy/UNIV-types.h"


struct type_ROSEAN_IA5List {
        struct type_UNIV_IA5String *IA5String;

        struct type_ROSEAN_IA5List *next;
};
int     free_ROSEAN_IA5List ();

struct type_ROSEAN_Empty {
    char    parm;
};
int     free_ROSEAN_Empty ();
#endif
```

Figure 50 : Le fichier ROSEAN-types.h

5.4.2.2.2 Le module de syntaxe abstraite augmenté

Ce module est construit pour être utilisé avec le programme PEPY.

L'emploi de ce fichier est cependant transparent dans la méthodologie. Les auteurs de ISODE en déconseillent, d'ailleurs, l'utilisation.

exemple

Le fichier généré pour notre application est à la figure 51.

```
-- automatically generated by posy 4.0 #4 (alma)

ROSEAN DEFINITIONS ::=

%{
#include <stdio.h>
#include "ROSEAN-types.h"
%}

PREFIXES encode decode print

BEGIN


ENCODER encode

IA5List [[P struct type_ROSEAN_IA5List *]]
    %{
    %}
    ::=
        SEQUENCE OF
            <<; parm; parm = parm -> next>>
            IA5String
            [[p parm -> IA5String ]]

Empty [[P struct type_ROSEAN_Empty *]] ::=
        NULL

DECODER decode

IA5List [[P struct type_ROSEAN_IA5List **]]
    %{
    %}
    ::=
        SEQUENCE OF
            %{
              if ((*(parm) = (struct type_ROSEAN_IA5List*)
                  calloc (1, sizeof **(parm))) ==
                  ((struct type_ROSEAN_IA5List *) 0)) {
                      advise (NULLCP, "out of memory");
                      return NOTOK;
                  }
            %}
```

```
            IA5String
            [[p &((*parm) -> IA5String)]]
            %{ parm = &((*parm) -> next); %}

Empty [[P struct type_ROSEAN_Empty **]] ::=
    %{
        if ((*(parm) = (struct type_ROSEAN_Empty *)
            calloc (1, sizeof **(parm))) ==
            ((struct type_ROSEAN_Empty *) 0)) {
                advise (NULLCP, "out of memory");
                return NOTOK;
        }
    %}
        NULL

END

%{

free_ROSEAN_IA5List (arg)
struct type_ROSEAN_IA5List *arg;
{
    struct type_ROSEAN_IA5List *parm = arg;

    if (parm == NULL)
        return;

    for (parm = parm;
            parm;
            parm = parm -> next) {
        if (parm -> IA5String)
            free_UNIV_IA5String (parm -> IA5String),
                parm -> IA5String = NULL;

        if (parm)
            free ((char *) parm);
    }

    free ((char *) arg);
}

free_ROSEAN_Empty (arg)
struct type_ROSEAN_Empty *arg;
{
    struct type_ROSEAN_Empty *parm = arg;

    if (parm == NULL)
        return;


    free ((char *) arg);
}

%}
```

Figure 51 : Le fichier ROSEAN-types.py

5.4.2.3 Exécution du programme POSY

Par convention, le nom de fichier argument du programme POSY a le suffixe .py.

ex : posy module.py

Les fichiers résultats se répartissent comme suit :

- MODULE-types.py, le module de syntaxe abstraite augmenté
- MODULE-types.h, les structures en langage C

5.4.2.4 Pour plus de renseignements

Le programme POSY est présenté dans le chapitre 4 de [ISODE4].

5.4.3 Le programme PEPY

Le programme PEPY est un compilateur d'éléments de présentation, c'est-à-dire d'éléments manipulables par la couche présentation du programme ISODE.

exemple

Ces fonctions se trouvent en annexe.

5.4.3.1 Argument du programme PEPY

PEPY lit une description d'un module de syntaxe abstraite ASN.1, définissant des types de données, et produit un ensemble de fonctions C qui reconnaissent, génèrent ou impriment les objets décrits dans le module.

5.4.3.2 Résultat du programme PEPY

PEPY génère un fichier C. Pour appeler les routines d'impression et de décodage de types, il faut appeler la routine suivante :

PREFIXE_MODULE_TYPE (pe, 1, len, buffer, parm)

avec **PREFIXE_MODULE_TYPE** qui indique que le type TYPE dans le module MODULE doit être analysé. Le préfixe PREFIXE est fourni à partir de la directive DECODER ou PRINTER, ou a partir du préfixe correspondant dans la directive SECTIONS.

pe l'élément présentation à examiner.

1 est un élément magique.

len est fourni quand une longueur est à fournir lors du décodage. NULLIP dans les autres cas.

buffer est fourni pour les routines de décodage en string. NULLVP dans le cas contraire.

parm est destiné à l'utilisateur, il est de type PEPYPARM. il prend la valeur NULLCP dans le cas ou il n'est pas utilisé.

Pour appeler les routines de codage, il faut faire appel à :

PREFIXE_MODULE_TYPE (&pe, 1, len, buffer, parm)

où **pe** est cette fois un pointeur vers l'élément de présentation à construire.

len est fourni pour les routines qui encodent des booléens ou des entiers. Sa valeur est NULL dans le cas ou len n'est pas utilisé.

buffer est le paramètre string pour les routines d'encodage de string. Buffer vaut NULLCP s'il n'est pas utilisé.

5.4.3.3 Limites du programme PEPY

PEPY n'est encore qu'au stade de prototype. C'est pourquoi il présente pour l'instant certaines limites :

- des limites d'ordre syntaxique, car il ne connaît pas la notion de macro.
- des limites d'ordre sémantique quand il s'agit des notions de COMPONENT OF, DEFAULT ou CHOICE.

5.4.3.4 Exécution du programme PEPY

Par convention, le nom de fichier argument du programme PEPY a comme suffixe .py.

ex : pepy module.py

Les fichiers résultats se répartissent en :

- module.c, le fichier C généré.
- module.ph, un fichier contenant des informations décrivant le module.

Plusieurs options sont possibles.

-a pour donner le nom de la routine advise (facilité en cas d'erreur).
-d pour ignorer les déclarations d'actions.
-h pour permettre l'ajout d'heuristiques lors de la génération d'un programme d'impression.
-o pour forcer le nom du fichier résultat.
-p pour enlever les actions du fichier argument et imprimer les informations résultantes.
-r pour générer un programme robuste. Le programme C généré ne s'arrêtera pas face à des termes inconnus.
-S pour installer le mode de fonctionnement initial de PEPY. Le mode par défaut est -S DECODE.

5.4.3.5 Exemple d'utilisation

L'objectif de l'exemple illustré ci-dessous, est de montrer comment le programme PEPY est utilisable.

5.4.5.1 Spécification

Le programme que nous allons construire a comme objectif de fabriquer un élément de présentation contenant la chaîne de caractère "hello world".

5.4.5.2 Le module de données abstraites

Le module de données abstraites, appelé module.py est écrit en ASN.1. Il se trouve ci-dessous, à la figure 52.

```
MODULE DEFINITIONS ::=

BEGIN
SECTIONS build none print

Texte ::= IA5String

END
```

Figure 52 : Le fichier module.py

5.4.5.3 Génération des routines

On exécute la commande **pepy module.py**

Le fichier C qui a été généré s'appelle **module.c**. Deux fonctions ont été créées comme il a été demandé dans la partie SECTIONS.

- build_MODULE_Texte
- print_MODULE_Texte

5.4.5.4 Construction du programme

Il reste maintenant à réaliser le programme principal, c'est-à-dire la fonction main() (figure 53).

```
#include "module.c"

main()
{
  PE    pe;  /* l'élément présentation a généré */
  char  *s;  /* la chaîne de caractères à transmettre */

  s = "hello world";
  build_MODULE_Texte (&pe, 1, NULLIP, s, NULLCP);
  print_MODULE_Texte (pe, 1, NULLIP, NULLCP, NULLCP);

}
```

Figure 53 : La fonction main

5.4.5.5 Exécution du programme créé

Nous avons placé la fonction main dans le fichier main.c. Il reste à compiler notre programme en exécutant la commande :

cc main.c libisode.a

La librairie libisode.a se trouve dans ISODE.

En exécutant ensuite le programme résultant de la compilation (a.out), on constate l'affichage à l'écran :

"hello world"

Le programme a donc bien encodé notre texte, et l'a prouvé, en utilisant la fonction d'élément de présentation print_MODULE_Texte.

5.5 Utilisation de ISODE pour la conception de nouvelles applications

5.5.1 Introduction

Le programme ISODE-4.0 contient un ensemble d'outils destinés à aider le concepteur d'applications.

- le compilateur PEPY génère un ensemble de routines de codage et de décodage de types décrits en ASN.1.
- le compilateur POSY génère les structures C correspondantes aux types ASN.1 fournis.
- le compilateur ROSY génère les tables qui seront employées par les routines initiatrices et répondeuses de l'association.
- la librairie LIBROSY contient ces routines génériques de communication basées sur l'exécution d'opérations à distance.

L'ensemble de la méthodologie utilisant ces outils est à la figure 54.

5.5.2 Construction d'une nouvelle application

1°) définir le module d'opération à distance en notation ROSE, sans utiliser le BIND . et le UNBIND, qui ne sont pas encore compris.

2°) compiler ce module avec le compilateur ROSY.

3°) compiler le fichier résultat de ROSY contenant les types en ASN.1 avec le compilateur POSY.

4°) compiler le fichier résultat de POSY contenant les types en ASN.1 avec le compilateur PEPY.

5°) - construire l'initiateur de l'application en utilisant la routine ryinitiator() de ISODE.
- définir le contenu de ce qu'il faut faire avant d'exécuter l'opération, après l'exécution de l'opération et en cas d'erreur.

6°) - construire le répondeur de l'application en utilisant le routine ryresponder() de ISODE.
- définir le contenu des opérations à effectuer.

7°) compiler l'initiateur et le répondeur avec la librairie isode.

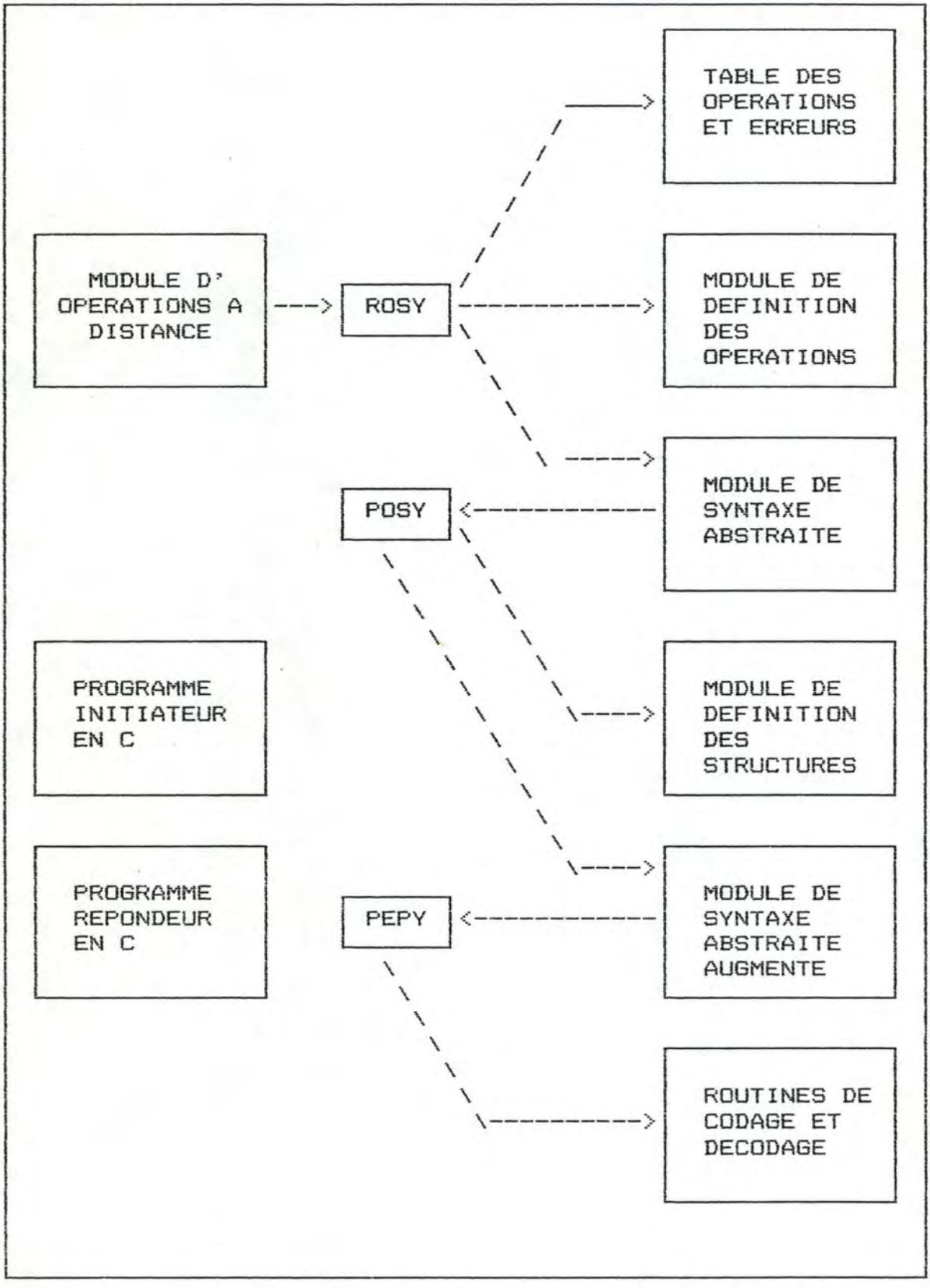

Figure 54 : récapitulatif de la méthodologie

5.5.3 Utilisation de ryinitiator et ryresponder

Voici la liste des paramètres des routines, initiatrice et répondeuse.

5.5.3.1 Ryinitiator

Ryinitiator (argc, argv, myservice, mycontext, mypci, ops, dispatches, quit)

/* routine initiatrice générique de ISODE */

argc et **argv** proviennent du vecteur argument que le programme aura reçu au moment où il aura été lancé.
myservice est la partie, ne se trouvant pas sur l'hôte, de l'information sur l'entité application.
mycontext est le nom de contexte application du service.
mypci est la syntaxe abstraite du service.
ops est la table des opérations possibles du service. Elle est générée par ROSY.
dispatches est un pointeur vers une table de dispatching.
quit est une routine à appeler quand l'association est terminée.

5.5.3.1 Ryresponder

Ryresponder (argc, argv, myservice, dispatches, ops, start, stop)

/* routine répondeuse générique de ISODE */

argc et **argv** proviennent du vecteur argument que le programme aura reçu au moment où il aura été lancé.
myservice est la partie, non de l'hôte, de l'information sur l'entité application.
dispatches est un pointeur vers une table de dispatching.
ops est un pointeur vers la table des opérations RyOperation.
start est l'adresse d'une routine qui décide si l'association est acceptable ou NULLIFP si elle doit toujours l'être.
stop est l'adresse d'une routine qui note qu'une demande de fin d'association ou un signal d'abandon est parvenu; Elle vaut NULLIFP si l'association n'a rien de remarquable.

5.5.3.3 Exemple

Le chapitre 10 du volume 4 du manuel utilisateur de ISODE-4.0 présente un exemple complet. Nous en conseillons la lecture si vous souhaitez plus d'informations sur le fonctionnement [ISODE884].

CONCLUSION

Ce mémoire visait essentiellement deux objectifs. Nous voulions, d'une part, présenter de façon aussi complète que précise, les éléments de service de la couche application et, d'autre part, apporter une preuve de l'utilité du modèle ISO lors de la réalisation d'une application non standardisée.

Le premier objectif (présentation des ASE) a été réalisé dans le chapitre 3.

Nous avons essayé d'expliquer l'intérêt de chacun des éléments de service tant individuellement que collectivement.

Nous avons également présenté les différents standards qui définissent le service et le protocole de chacun d'eux.

Le second objectif (applicabilité du modèle ISO) a été réalisé dans les chapitres 4 et 5.

Nous avons, tout d'abord, choisi un programme qui a été implémenté en utilisant le modèle de référence : ISODE.

Nous avons, ensuite, décrit comment les éléments de service s'y trouvaient.

Nous avons, finalement, pu réaliser, assez facilement, une application basée sur l'exécution d'opérations à distance, tout en utilisant ISODE. Nous avons suivi la méthode de conception proposée par ses concepteurs.

C'est ainsi que nous n'avons du recourir, à aucun moment, aux notions de soquet, codage de type, mécanisme de transfert ou même d'APDU.

Au vu de ces quelques lignes, on pourrait aisément imaginer que ISO est devenu le nec plus utra de la télécommunication.

Il n'en n'est évidemment rien puisqu'il ne faut pas oublier que les couches inférieures de ISODE sont basées sur le modèle TCP-IP. En effet, il semble que, dans l'état actuel de la normalisation, les niveaux inférieurs du modèle de référence sont trop peu performants.

TABLE DES FIGURES

LISTE DES ABREVIATIONS

AARE	Application-Association REsponse
AARQ	Application-Association ReQuest
ABRT	ABoRT
ACSE	Association Control Service Element
AEI	Application Entity Information
APDU	Application Protocol Data Unit
ASE	Application Service Element
ASN.1	Abstract Syntax Notation One
CCITT	Comité de Consultation International des Téléphones et Télégraphes
CCRSE	Commitment, Concurrency and Recovery Service Element
DIS	Draft International Standard
ECMA	European Computer Maufacturers Association
IS	Internationnal Standard
ISO	Interconnection de systèmes ouverts
ISODE	ISO Development Environment
OSI	Organisation de standardisation internationale
PEPY	Presentation Element Parser Yacc-based
POSY	PEPY Optional Structure-generator Yacc-based
RLRE	ReLease REsponse
RLRQ	ReLease ReQuest
ROER	Remote Operation ERror
ROIV	Remote Operation INvoke
RORJ	Remote Operation ReJect
RORS	Remote Operation ReSult
ROSE	Remote Operation Service Element
ROSY	Remote Operation Stub-generator Yacc-based
RPC	Remote Procedure Call
RTAB	Reliable Transfer ABort
RTOAC	Reliable Transfer Open Accept
RTORQ	Reliable Transfer Open ReQuest
RTSE	Reliable Transfer Service Element
RTTP	Reliable Transfer Turn Please
RTTR	Reliable Transfer TRansfer
SAP	Service Access Point
TCP	Transmission Control Protocol
TSAPD	Transport Service Access Point Deamon
XDR	eXternal Data Representation

BIBLIOGRAPHIE

[CCITTX218] : CCITT, Reliable Transfer : Model and Service Définition. Genève, Mars 1988. Draft Recommendation X218.

[CCITTX228] : CCITT, Reliable Transfer : Protocol Specification. Genève, Mars 1988. Draft Recommendation X228.

[CCITTX219] : CCITT, Remote Operation : Model, Notation and Service Définition. Genève, Mars 1988. Draft Recommendation X219.

[CCITTX229] : CCITT, Remote Operation : Protocol Specification. Genève, Mars 1988. Draft Recommendation X229.

[CCITTX200] : CCITT, Reference model. Genève, Mars 1988. International Standard X200.

[ISO8649] : ISO, Information Processing System - Open Systems Interconnection - Service Definition for the Association Control Service Element. Avril 1988. Revised Final Text of Draft International Standard 8649.

[ISO8650] : ISO, Information Processing System - Open Systems Interconnection - Protocol Specification fot the Association Control Service Element. Avril 1988. Revised Final Text of Draft International Standard 8650.

[ISO9066-1] : ISO, Information Processing System - Text Communication - Reliable Transfer Part 1 : Model and Service Definition. Mars 1988. Working Document for International Standard 9066-1.

[ISO9066-2] : ISO, Information Processing System - Text Communication - Reliable Transfer Part 2 : Protocol Specification. Mars 1988. Working Document for International Standard 9066-2.

[ISO9072-1] : ISO, Information Processing System - Text Communication - Remote Operations Part 1 : Model, Notation and Service Definition. Mars 1988. Working Document for International Standard 9072-1.

[ISO9072-2] : ISO, Information Processing System - Text Communication - Remote Operations Part 2 : Protocol Specification. Mars 1988. Working Document for International Standard 9072-2.

[ISO7498] : ISO, Open Systems Interconnection - Basic Reference Model. International Standard 7498.

[BROSSEL87] : Ph Brossel, "Contribution à la mise en oeuvre d'un logiciel X400 : utilisation des langages formels de spécification et réalisation d'un outil d'analyse et de production d'unités de données de protocole". Mémoire présenté en vue de l'obtention du titre de licencié et maître en informatique, Facultés Universitaires Notre-Dame de la Paix, Namur, 1987.

[ECMATR/31] : ECMA TR/31, "Remote Operations : Concepts, Notation and connection-Oriented Mappings". December 1985.

[ISO8824] : ISO, Information Processing - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1). International Standard 8824, 1987.

[CCITTSYN] : CCITT, Message Handling Systems : Presentation Transfer Syntax and Notation3. Malaga-Torremolinos, Octobre 1984.

[ISODE881] : Marshall T.Rose, "The ISO Development Environment; User's Manual, Volume 1". USA, 1988.

[ISODE884] : Marshall T.Rose, "The ISO Development Environment; User's Manual, Volume 4: The Application Cookbook". USA, 1988.

[HESHSHOW] : John Henshall et Sandy Show, "OSI explained : end to end computer communication standards". Ellis Horwood Series in Computer Communications and Networking. International Computers Ltd (UK), Edinburgh, 1988.

[STALLINGS] : William Stallings, "Data and Computer Communications", second edition. MacMillan Publishers. USA, 1985.

[ROSECASS] : Marshall T.Rose and Dwight E. Cass, "OSI Transport Services on top of the TCP". Northrop Research and Technology Center, Palos Verdes Peninsula, CA 90274, USA. 1988.

[EAN] : Message Handling System. DSN - EAN V2.1. Administrator Manual, June 88.

[ISODE] : Marshall T.Rose, "The ISO Development Environment" Version 4. Northrop Research and Technology Center One Research Park, Palos Verdes Peninsula, CA 90274 USA, July 88.

[TCP-IP] : Transmission Control Protocol. DDN Protocol Handbook, Volume One : DoD Military Standards Protocols, DDN Network Information Center, SRI International, September, 1981.

ANNEXES

Annexe 1 : Notation ROSE : définition des macros

```
Remote-Operation-Notation { joint-iso-ccitt remoteOperations(4) notation (0) }
DEFINITIONS ::=
BEGIN

EXPORTS BIND, UNBIND, OPERATION, ERROR;

-- macro definition for bind-operations

BIND MACRO ::=
BEGIN

TYPE NOTATION          ::= Argument Result Error
VALUE NOTATION         ::= value (VALUE NULL)

Argument               ::= "ARGUMENT" NamedType | empty

Result                 ::= "RESULT" NamedType | empty

Error                  ::= "BIND-ERROR" NamedType | empty

NamedType              ::= identifier type | type

END

-- macro definition for unbind-operations

UNBIND MACRO ::=

BEGIN

TYPE NOTATION          ::= Argument Result Errors
VALUE NOTATION         ::= value (VALUE NULL)

Argument               ::= "ARGUMENT" NamedType | empty

Result                 ::= "RESULT" NamedType | empty

Errors                 ::= "UNBIND-ERROR" NamedType | empty

NamedType              ::= identifier type | type

END
```

```
-- macro definition for operations

OPERATION MACRO ::=
BEGIN

TYPE NOTATION       ::= Argument Result Errors LinkedOperations
VALUE NOTATION      ::= value (VALUE CHOICE{
                                    localValue    INTEGER,
                                    globalValue   SEQUENCE { OBJECT IDENTIFIER, INTEGER}})

Argument            ::= "ARGUMENT" NamedType | empty
Result              ::= "RESULT"ResultType | empty
ResultType          ::= NamedType | empty
Errors              ::= "ERRORS" "{" ErrorNames "}" | empty
LinkedOperations    ::= "LINKED" "{" LinkedOperationNames "}" | empty
ErrorNames          ::= ErrorList | empty
ErrorList           ::= Error | ErrorList "," Error
Error               ::= value ( ERROR )
LinkedOperationNames ::= OperationList | empty
OperationList       ::= Operation | OperationList "," Operation
Operation           ::= value ( OPERATION )
NamedType           ::= identifier type | type
END

-- macro definition for operations errors

ERROR MACRO ::=
BEGIN

TYPE NOTATION       ::= Parameter
VALUE NOTATION      ::= value (VALUE CHOICE{
                                    localValue    INTEGER,
                                    globalValue   SEQUENCE { OBJECT IDENTIFIER, INTEGER}})
Parameter           ::= "PARAMETER" NamedType | empty
NamedType           ::= identifier type | type
END

END     -- end of Remote Operations Notation
```

Annexe 1 : Notation ROSE : définition des macros

```
Remote-Operations-Notation-extension { joint-iso-ccitt remoteOperations(4) notation-extension (2) }
DEFINITIONS ::=
BEGIN

EXPORTS    APPLICATION-SERVICE-ELEMENT, APPLICATION-CONTEXT;

IMPORTS    OPERATION, BIND, UNBIND FROM    Remote-Operation-Notation
                                           { joint-iso-ccitt remoteOperations(4) notation(0)};

-- macro definition for ASEs

APPLICATION-SERVICE-ELEMENT MACRO ::=
BEGIN

TYPE NOTATION          ::= SymmetricAse | ConsumerInvokes SupplierInvokes | empty
VALUE NOTATION         ::= value (VALUE OBJECT IDENTIFIER)

SymmetricAse           ::= "OPERATIONS" "{" OperationList "}"

ConsumerInvokes        ::= "CONSUMER INVOKES" "{" OperationList "}" | empty

SupplierInvokes        ::= "SUPPLIER INVOKES"   "{" OperationList "}" | empty

OperationList          ::= Operation | OperationList "," Operation

Operation              ::= value (OPERATION)

END

APPLICATION-CONTEXT MACRO ::=
BEGIN

TYPE NOTATION          ::= NonROelements Binding ROelements AbstractSyntaxes
VALUE NOTATION         ::= value (VALUE OBJECT IDENTIFIER)

NonROelements          ::= "APPLICATION SERVICE ELEMENTS" "{" AseList "}"

Binding                ::= "BIND" value (BIND)
                           "UNBIND" value (UNBIND)

ROelements             ::= "REMOTE OPERATIONS" "{" AseID "}" -- identifying ROSE
                           SymmetricAses AsymmetricAses | empty

SymmetricAses          ::= "OPERATIONS OF" "{" AseList "}" | empty

AsymmetricAses         ::= InitiatorConsumerOf ResponderConsumerOf

InitiatorConsumerOf    ::= "INITIATOR CONSUMER OF" "{" AseList "}" | empty

ResponderConsumerOf    ::= "RESPONDER CONSUMER OF" "{" AseList "}" | empty

AbstractSyntaxes       ::= "ABSTRACT SYNTAXES" "{" AbstractSyntaxList "}"

AseList                ::= AseID | AseList "," AseID

AseID                  ::= value ( APPLICATION-SERVICE-ELEMENT)

AbstractSyntaxList     ::= AbstractSyntax | AbstractSyntaxList "," AbstractSyntax

AbstractSyntax         ::= value (OBJECT IDENTIFIER)-- identifying abstract syntax

END

END                        -- end of Remote Operations Notation extension
```

```
struct AEInfo
    PE   aei_ap_title;       /* titre du processus application */
    PE   aei_ae_qualifier; /*qualification de l'entité application */
    int    aei_ap_id ;       /* l'identif.de l'appel du processus applic.*/
    int    aei_ae_id;        /* l'identificateur de l'appel de l'entité applic */
    int    aei_flags;        /* pour indiquer si les identif.sont présents */


struct AcSAPstart
   int       acs_sd;          /* le descripteur d'association à utiliser */
   OID     acs_context;     /* le nom du contexte application */
   AEInfo acs_callingtitle; /* information sur l'entité application appelante*/
   AEInfo acs_calledtitle;  /* information sur l'entité appelée */
   struct PSAPstart acs_start; /* une structure PSAPstart */
   int       acs_ninfo ;      /* longueur des données initiales */
   PE       acs_info;         /* données initiales */


struct AcSAPindication
   int     aci_type; /* type de l'événement arrivé */
   union {
      struct AcSAPfinish  aci_un_finish;
      struct AcSAPabort  aci_un_abort;
   } aci_un;


struct AcSAPabort
    int    aca_source;  /* la source de l'abort : user, provider,local */
    int    aca_reason;  /* la raison de l'abort pour le provider */
    int    aca_ninfo;    /* longueur des données finales */
    PE    aca_info      /* données finales */
    int    aca_cc        /* diagnostic pour le provider */
    int    aca_data     /*                "                  */


struct AcSAPconnect
    int    acc_sd;           /* le descripteur d'association à utiliser */
    int    acc_result;       /* la réponse à l'association */
    int    acc_diagnostic;   /* la diagnostique de l'association */
    OID  acc_context       /* le nom de contexcte application */
    AEInfo  acc_respondtitle;  /*inform. sur l'entité application répondant */
    struct PSAPconnect    acc_connect;  /* une structure PSAPconnect */
    int    acc_ninfo; /* longueur des données initiales */
    PE    acc_info;  /* données initiales */
```

```
struct AcSAPrelease
    int     acr_affirmative;  /* l'indicateur d'accord */
    int     acr_reason;       /* la raison de l'indicateur */
    int     acr_ninfo;        /* longueur des données finales */
    char    acr_info;         /* données finales */


struct AcSAPfinish
    int     acf_reason;    /* la raison de la demande d'abandon */
    int     acf_ninfo;     /* longueur des données finales */
    char    acf_info;      /* données finales */
```

```
struct RoSAPindication
    int    roi_type;  /* type de l'événement */
    union {
      struct RoSAPinvoke     roi_un_invoke;
      struct RoSAPresult     roi_un_result;
      struct RoSAPerror      roi_un_error;
      struct RoSAPureject    roi_un_ureject;
      struct RoSAPpreject    roi_un_preject
      struct RoSAPend        roi_un_end;
      struct AcSAPfinish     roi_un_finish;
    } roi_un;
```

```
struct RoSAPinvoke
    int  rox_id;        /* l'identificateur d'appel pour cette requête */
    int  rox_linkid;    /* si rox_nolinked n'est pas completé, il contient */
                        /* l'identificateur d'appel de la requête liée   */
    int  rox_nolinked;/* l'indicateur 'identificateur liée */
    int  rox_op;        /* le code opération */
    PE  rox_args;       /* les arguments de l'opération */
```

```
struct RoSAPresult
    int  ror_id;       /* l'identificateur d'appel de cette réponse */
    int  ror_op;       /* le code opération de l'opération exécutée */
    PE  ror_result;  /* Les resultats de l'opération */
```

```
struct RoSAPerror
    int  roe_id;       /* l'identificateur de l'appel de cette réponse */
    int  roe_error;    /* le code d'erreur */
    PE  roe_param;   /* les paramètres de l'erreur */
```

```
struct RoSAPureject
    int  rou_id;        /* si rou_noid n'est pas complété, il contient l' */
                        /* identificateur d'appel de la requête */
    int  rou_noid;      /* l'indicateur de l'identificateur d'appel */
    int  rou_reason; /* la raison du rejet */
```

```
struct RoSAPend
   int  roe_dummy;  /*depend du mode de connexion, sinon RoSAPfinish*/
```

```
struct RtSAPstart
    int  rts_sd;      /* le descripteur d'association à utiliser */
    int  rts_mode;  /* le mode de dialogue */
    int  rts_turn;   /* le possesseur du tour initial */
    PE  rts_data;   /* des données inirtiales */
    struct AcSAPstart rts_start;  /* une structure AcSAPstart */


struct RtSAPconnect
    int  rtc_sd;      /* le descripteur d'association à utiliser */
    int  rtc_result;  /* la réponse à l'association */
    int  rtc_data;   /* des données initiales */
    struct AcSAPconnect rtc_connect; /* structure AcSAPconnect */


struct RtSAPindication
    int  rti_type;  /* type de l'événement */
    union {
      struct RtSAPturn    rti_un_turn;
      struct RtSAPresult rti_un_transfer;
      struct RtSAPabort  rti_un_abort;
      struct RtSAPclose  rti_un_close;
      struct AcSAPfinish rti_un_finishj;
   } rti_un;


struct RtSAPabort
    int   rta_peer;    /* si complêté, indique un abort de l'utilisateur initié */
    int   rta_reason; /* la raison de l'abort */
    PE   rta_data;    /* donnée utilisateur, optionnel */
    int   rta_cc;       /* diagnostique pour le provider */
    char rta_data;    /*            "                    */


struct RtSAPtransfer
    PE   rtt_data;


struct RtSAPturn
    int  rtu_please;  /* si complêté, RT-PLEASE-TURN.ind s'est produit*/
                      /* sinon RT-GIVE-TURN.ind */
    int  rtu_priority;  /* la priorité à laquelle le touer est demandé */


struct RtSAPclose
    int  rtc_dummy; /* selon mode de connexion, pour X410-CLOSE */
```

Annexe 3 : Le Makefile de l'application rosean

```
#############################################################################
#   Instructions to Make, for compilation of ROS-based ean services
#  Ph.CAYPHAS 31/05/89
#############################################################################

.SUFFIXES:      .ry .py .c .o

 c.o:;          $(CC) $(CFLAGS) -DPEPYPATH -c $*.c

.py.c:;         $(TOPDIR)pepy/xpepy -m $(PYFLAGS) $<


LIBES   =       librosean.a $(TOPDIR)libisode$(LPP).a
 LIBS   =       $(TOPDIR)llib-lisode$(LPP)
CFILES  =       roseand.c rosean.c ROSEAN-ops.c ryinitiator.c ryresponder.c
RYFILES =       rosean.ry
HFILES  =       $(HDIR)rosy.h $(HDIR)rosap.h $(HDIR)acsap.h $(HDIR)psap2.h \
                $(HDIR)psap.h $(HDIR)ssap.h $(HDIR)isoaddrs.h \
                $(HDIR)manifest.h $(HDIR)general.h $(HDIR)config.h

ROS     =       ros.


#############################################################################
# Here it is...
#############################################################################

all:            roseand rosean
inst-all:       inst-roseand inst-rosean


#############################################################################
# roseand
#############################################################################

inst-roseand:   $(ETCDIR)$(ROS)rosean

$(ETCDIR)$(ROS)rosean:  xroseand$(LPP)
                -cp $@ z$(ROS)rosean
                -rm -f $@
                cp xroseand$(LPP) $@
                -@ls -gls $@
                -@echo ""

roseand:                xroseand$(LPP)

xroseand$(LPP): roseand.o ROSEAN-Rops.o ryresponder.o $(LIBES)
                $(LDCC) $(LDFLAGS) -o $@ roseand.o ROSEAN-Rops.o ryresponder.o )


roseand.o:      ryresponder.h ROSEAN-ops.h ROSEAN-types.h $(HFILES) \
                        $(HDIR)logger.h

ROSEAN-Rops.o:  ROSEAN-ops.c ROSEAN-ops.h $(HFILES)
                $(CC) $(CFLAGS) -DPERFORMER -DPEPYPATH -c ROSEAN-ops.c
                mv ROSEAN-ops.o $@

ryresponder.o:  ryresponder.h $(HFILES)
```

Annexe 3 : Le Makefile de l'application rosean

```
#####################################################################
# rosean
#####################################################################

inst-rosean:    $(BINDIR)rosean$(LPP)

$(BINDIR)rosean$(LPP):  xrosean$(LPP)
                -cp $@ zxrosean$(LPP)
                -rm -f $@
                cp xrosean$(LPP) $@
                -@ls -gls $@
                -@echo ""

rosean:         xrosean$(LPP)

xrosean$(LPP):  rosean.o ROSEAN-Iops.o ryinitiator.o $(LIBES)
                $(LDCC) $(LDFLAGS) -o $@ rosean.o ROSEAN-Iops.o ryinitiator.o \
                        $(LIBES) $(LSOCKET)

rosean.o:       ryinitiator.h ROSEAN-ops.h ROSEAN-types.h $(HFILES)

ROSEAN-Iops.o:  ROSEAN-ops.c ROSEAN-ops.h $(HFILES)
                $(CC) $(CFLAGS) -DINVOKER -DPEPYPATH -c ROSEAN-ops.c
                mv ROSEAN-ops.o $@

ryinitiator.o:  ryinitiator.h $(HFILES)


#####################################################################
# librosean
#####################################################################

librosean.a:    ROSEAN-O
                -rm -f $@
                @$(UTILDIR)make-lib.sh $(SYSTEM) $(ARFLAGS) $@ $(ROSEAN-O)
                -@ls -l $@
                -@echo "ROSEAN library built normally"

ROSEAN-O        =       ROSEAN-[0-9]*.o
ROSEAN-C        =       ROSEAN-[0-9]*.c

ROSEAN-O:       ROSEAN-C
                @$(MAKE) `/bin/ls $(ROSEAN-C) | sed 's/\.c$$/.o/'`
                -@touch $@

ROSEAN-C:       ROSEAN-types.py $(TOPDIR)pepy/xpepy
                -@rm -f $(ROSEAN-C) $(ROSEAN-O)
                $(TOPDIR)pepy/xpepy -a ryr_advise -m -A -b ROSEAN $(PYFLAGS) \
                        ROSEAN-types.py
                -@touch $@

ROSEAN-types.py:        ROSEAN-asn.py $(TOPDIR)pepy/xposy
                $(TOPDIR)pepy/xposy -f -h -m -o $@ $(POFLAGS) ROSEAN-asn.py
ROSEAN-types.h: ROSEAN-types.py

ROSEAN-asn.py:  rosean.ry $(TOPDIR)rosy/xrosy
                $(TOPDIR)rosy/xrosy -m $(RYFLAGS) -o $@ rosean.ry
ROSEAN-ops.c:   rosean.ry
ROSEAN-ops.h:   rosean.ry
ROSEAN-stubs.c: rosean.ry
```

ROSEAN-1.c

```
/* automatically generated by pepy 4.0 #4 (alma), do not edit! */

#include "psap.h"

#define advise  ryr_advise

void    advise ();

/* Generated from module ROSEAN */

#include <stdio.h>
#include "ROSEAN-types.h"

#ifndef PEPYPARM
#define PEPYPARM char *
#endif /* PEPYPARM */
extern PEPYPARM NullParm;

/* ARGSUSED */

int     encode_ROSEAN_IA5List (pe, explicit, len, buffer, parm)
register PE     *pe;
int     explicit;
int     len;
char   *buffer;
struct type_ROSEAN_IA5List * parm;
{
# line 18 "ROSEAN-types.py"


    PE  p0 = NULLPE;
    PE  p1_z = NULLPE;
    register PE *p1 = &p1_z;

    if (((*pe) = pe_alloc (PE_CLASS_UNIV, PE_FORM_CONS, PE_CONS_SEQ)) == NULLPE{
        advise (NULLCP, "IA5List: out of memory");
        return NOTOK;
    }
    for (; parm; parm = parm -> next) {
        if (encode_UNIV_IA5String (p1, 0, 0, NULLCP, parm -> IA5String ) == NOT)
            return NOTOK;

#ifdef TESTDEBUG
        (void) testdebug ((*p1), "element");
#endif

        seq_addon ((*pe), p0, (*p1));
        p0 = (*p1);
    }

#ifdef TESTDEBUG
    (void) testdebug ((*pe), "ROSEAN.IA5List");
#endif


    return OK;
}
```

ROSEAN-2.c

```
/* automatically generated by pepy 4.0 #4 (alma), do not edit! */

#include "psap.h"

#define advise  ryr_advise

void    advise ();

/* Generated from module ROSEAN */

#include <stdio.h>
#include "ROSEAN-types.h"

#ifndef PEPYPARM
#define PEPYPARM char *
#endif /* PEPYPARM */
extern PEPYPARM NullParm;

/* ARGSUSED */

int     encode_ROSEAN_Empty (pe, explicit, len, buffer, parm)
PE     *pe;
int     explicit;
int     len;
char   *buffer;
struct type_ROSEAN_Empty * parm;
{
    if (((*pe) = pe_alloc (PE_CLASS_UNIV, PE_FORM_PRIM, PE_PRIM_NULL))
       == NULLPE) {
        advise (NULLCP, "Empty: out of memory");
        return NOTOK;
    }

#ifdef TESTDEBUG
    (void) testdebug ((*pe), "ROSEAN.Empty");
#endif


    return OK;
}
```

ROSEAN-3.c

```
/* automatically generated by pepy 4.0 #4 (alma), do not edit! */

#include "psap.h"

#define advise  ryr_advise

void    advise ();

/* Generated from module ROSEAN */

#include <stdio.h>
#include "ROSEAN-types.h"

#ifndef PEPYPARM
#define PEPYPARM char *
#endif /* PEPYPARM */
extern PEPYPARM NullParm;

/* ARGSUSED */

int     decode_ROSEAN_IA5List (pe, explicit, len, buffer, parm)
register PE     pe;
int     explicit;
int    *len;
char  **buffer;
struct type_ROSEAN_IA5List ** parm;
{
# line 32 "ROSEAN-types.py"


    register PE p2;

#ifdef TESTDEBUG
    (void) testdebug (pe, "ROSEAN.IA5List");
#endif

    if (explicit) {
        if (pe -> pe_class != PE_CLASS_UNIV
                || pe -> pe_form != PE_FORM_CONS
                || pe -> pe_id != PE_CONS_SEQ) {
            advise (NULLCP, "IA5List bad class/form/id: %s/%d/0x%x",
                    pe_classlist[pe -> pe_class], pe -> pe_form, pe -> pe_id);
            return NOTOK;
        }
    }
    else
        if (pe -> pe_form != PE_FORM_CONS) {
            advise (NULLCP, "IA5List bad form: %d", pe -> pe_form);
            return NOTOK;
        }

    if ((p2 = prim2seq (pe)) == NULLPE) {
        advise (NULLCP, "IA5List bad sequence: %s",
                pe_error (pe -> pe_errno));
        return NOTOK;
    }
    pe = p2;
```

ROSEAN-3.c (suite)

```
    for (p2 = first_member (pe); p2; p2 = next_member (pe, p2)) {
        {
# line 36 "ROSEAN-types.py"

                if ((*(parm) = (struct type_ROSEAN_IA5List *)
                        calloc (1, sizeof **(parm))) ==
                        ((struct type_ROSEAN_IA5List *) 0)) {
                    advise (NULLCP, "out of memory");
                    return NOTOK;
                }

        }
        {
#ifdef TESTDEBUG
            (void) testdebug (p2, "element");
#endif

          if (decode_UNIV_IA5String (p2,1,NULLIP,NULLVP,&((*parm)->IA5String))
             == NOTOK)
                return NOTOK;
             {
# line 45 "ROSEAN-types.py"
 parm = &((*parm) -> next);
             }
        }
    }

    return OK;
}
```

ROSEAN-4.c

```
/* automatically generated by pepy 4.0 #4 (alma), do not edit! */

#include "psap.h"

#define advise  ryr_advise

void    advise ();

/* Generated from module ROSEAN */

#include <stdio.h>
#include "ROSEAN-types.h"

#ifndef PEPYPARM
#define PEPYPARM char *
#endif /* PEPYPARM */
extern PEPYPARM NullParm;

/* ARGSUSED */

int     decode_ROSEAN_Empty (pe, explicit, len, buffer, parm)
PE      pe;
int     explicit;
int    *len;
char  **buffer;
struct type_ROSEAN_Empty ** parm;
{
#ifdef TESTDEBUG
    (void) testdebug (pe, "ROSEAN.Empty");
#endif

    if (explicit) {
      if (pe -> pe_class != PE_CLASS_UNIV || pe -> pe_id != PE_PRIM_NULL) {
          advise (NULLCP, "Empty bad class/form/id: %s/%d/0x%x",
               pe_classlist[pe -> pe_class], pe -> pe_form, pe -> pe_id);
          return NOTOK;
      }
    }

    {
# line 48 "ROSEAN-types.py"

      if ((*(parm) = (struct type_ROSEAN_Empty *)
           calloc (1, sizeof **(parm))) == ((struct type_ROSEAN_Empty *) 0))
      {
          advise (NULLCP, "out of memory");
          return NOTOK;
      }

    }

    return OK;
}
```

ROSEAN-5.c

```
/* automatically generated by pepy 4.0 #4 (alma), do not edit! */

#include "psap.h"

#define advise  ryr_advise

void    advise ();

/* Generated from module ROSEAN */

#include <stdio.h>
#include "ROSEAN-types.h"

#ifndef PEPYPARM
#define PEPYPARM char *
#endif /* PEPYPARM */
extern PEPYPARM NullParm;
# line 59 "ROSEAN-types.py"


free_ROSEAN_IA5List (arg)
struct type_ROSEAN_IA5List *arg;
{
    struct type_ROSEAN_IA5List *parm = arg;

    if (parm == NULL)
        return;

    for (parm = parm;
         parm;
         parm = parm -> next) {
        if (parm -> IA5String)
            free_UNIV_IA5String (parm -> IA5String),
                parm -> IA5String = NULL;

        if (parm)
            free ((char *) parm);
    }

    free ((char *) arg);
}

free_ROSEAN_Empty (arg)
struct type_ROSEAN_Empty *arg;
{
    struct type_ROSEAN_Empty *parm = arg;

    if (parm == NULL)
        return;


    free ((char *) arg);
}
```

```
/* rosean.c - ISODE - EAN  service -- initiator */


#include <stdio.h>
#include <pwd.h>
#include "ryinitiator.h"        /* for generic interactive initiators */
#include "ROSEAN-ops.h"         /* ROSEAN operation definitions */
#include "ROSEAN-types.h"       /* ROSEAN type definitions */


/* VARIABLES DE L'APPLICATION */

static char *myservice = "roseanstore";
static char *mycontext = "isode ean demo";
static char *mypci = "isode ean demo pci";


/* TYPES DEFINIS DANS L'APPLICATION */

struct type_ROSEAN_IA5List *vec2ia5list ();


/* FONCTIONS APPELABLES PAR L'UTILISATEUR */
int     do_ean (), do_help (), do_quit ();


/* FONCTIONS RESULTATS UTILISEES APRES L'EXECUTION D'OPERATIONS*/

int     ean_result ();

/* FONCTION DE TRAITEMENT DES ERREURS */
int     rosean_error ();



/* TABLE RECAPITULATIVE UTILISEE PAR L'INITIATEUR GENERIQUE */
static struct dispatch dispatches[] = {
    "ean", operation_ROSEAN_ean,
    do_ean, free_ROSEAN_IA5List,
    ean_result, rosean_error,
    "send a message with ean",

    "help", 0,
    do_help, NULLIFP,
    NULLIFP, NULLIFP,
    "print this information",

    "quit", 0,
    do_quit, NULLIFP,
    NULLIFP, NULLIFP,
    "terminate the association and exit",

    NULL
};



/*
```

```
      FONCTION MAIN */


main (argc, argv, envp)
int     argc;
char  **argv,
      **envp;
{
    ryinitiator (argc, argv, myservice, mycontext, mypci,
                 table_ROSEAN_Operations, dispatches, do_quit);

    exit (0);                       /* NOTREACHED */
}

/* FONCTIONS DE TRANSFORMATION DES TYPES */

struct type_ROSEAN_IA5List *vec2ia5list (vec)
char  **vec;
{
    struct type_ROSEAN_IA5List  *ia5;
    register struct type_ROSEAN_IA5List **ia5p;

    ia5 = NULL;
    ia5p = &ia5;

    for (; *vec; vec++) {
        if ((*ia5p = (struct type_ROSEAN_IA5List *) calloc (1, sizeof **ia5p))
                == NULL)
            adios (NULLCP, "out of memory");

        if (((*ia5p) -> IA5String = str2qb (*vec, strlen (*vec), 1)) == NULL)
            adios (NULLCP, "out of memory");

        ia5p = &((*ia5p) -> next);
    }

    return ia5;
}



static  print_ia5list (ia5)
register struct type_ROSEAN_IA5List *ia5;
{
    register struct qbuf *p,
                         *q;

    for (; ia5; ia5 = ia5 -> next) {
        p = ia5 -> IA5String;
        for (q = p -> qb_forw; q != p ; q = q -> qb_forw)
            printf ("%*.*s", q -> qb_len, q -> qb_len, q -> qb_data);
        printf ("\n");
    }
}

/*
```

```
    */

/* OPERATIONS PERMISES A L'UTILISATEUR  */


/* EXECUTER L'OPERATION A DISTANCE EAN */

static int  do_ean (sd, ds, args, ia5)
int     sd;
struct dispatch *ds;
char  **args;
register struct type_ROSEAN_IA5List **ia5;
{
    char   *cp,
           *dp,
            buffer[BUFSIZ];
    register struct type_ROSEAN_IA5List  *ia52;
    register struct passwd *pw;

    if (args[0] == NULL || args[1] == NULL) {
        advise (NULLCP, "usage: ean user message ...");
        return NOTOK;
    }

    *ia5 = vec2ia5list (args);

    cp = (pw = getpwuid (getuid ())) ? pw -> pw_name : "anon";
    dp = PLocalHostName ();

    if ((ia52 = (struct type_ROSEAN_IA5List *) calloc (1, sizeof *ia52))
            == NULL)
        adios (NULLCP, "out of memory");
    (void) sprintf (buffer, "%s@%s", cp, dp);
    if ((ia52 -> IA5String = str2qb (buffer, strlen (buffer), 1)) == NULL)
        adios (NULLCP, "out of memory");

/* kludge this arg onto front of list - HACK ATTACK */
    ia52 -> next = *ia5;
    *ia5 = ia52;

    return OK;
}

/*
```

```
    */

/* DEMANDER DE L'AIDE */

static int  do_help (sd, ds, args, dummy)
int     sd;
register struct dispatch *ds;
char  **args;
caddr_t *dummy;
{
    printf ("\nCommands are:\n");
    for (ds = dispatches; ds -> ds_name; ds++)
        printf ("%s\t%s\n", ds -> ds_name, ds -> ds_help);

    return NOTOK;
}




/* QUITTER LE PROGRAMME */

static int  do_quit (sd, ds, args, dummy)
int     sd;
struct dispatch *ds;
char  **args;
caddr_t *dummy;
{
    struct AcSAPrelease acrs;
    register struct AcSAPrelease   *acr = &acrs;
    struct AcSAPindication  acis;
    register struct AcSAPindication *aci = &acis;
    register struct AcSAPabort *aca = &aci -> aci_abort;

    if (AcRelRequest (sd, ACF_NORMAL, NULLPEP, 0, acr, aci) == NOTOK)
        acs_adios (aca, "A-RELEASE.REQUEST");

    if (!acr -> acr_affirmative) {
        (void) AcUAbortRequest (sd, NULLPEP, 0, aci);
        adios (NULLCP, "release rejected by peer: %d", acr -> acr_reason);
    }

    ACRFREE (acr);

    exit (0);
}


/*
```

```
    */

/* FONCTIONS A UTILISER APRES L'EXECUTION DES OPERATIONS A DISTANCE */

static int  ean_result (sd, id, dummy, result, roi)
int     sd,
        id,
        dummy;
caddr_t result;
struct RoSAPindication *roi;
{
    printf ("done.\n");
    return OK;
}


/* FONCTION A UTILISER EN CAS D'ERREUR */


static int  rosean_error (sd, id, error, parameter, roi)
int     sd,
        id,
        error;
register struct type_ROSEAN_IA5List *parameter;
struct RoSAPindication *roi;
{
    register struct RyError *rye;

    if (error == RY_REJECT) {
        advise (NULLCP, "%s", RoErrString ((int) parameter));
        return OK;
    }

    if (rye = finderrbyerr (table_ROSEAN_Errors, error))
        advise (NULLCP, "%s",  rye -> rye_name);
    else
        advise (NULLCP, "Error %d", error);

    if (parameter)
        print_ia5list(parameter);

    return OK;
}
```

```
/* roseand.c - ISODE - EAN  service -- responder */
/* version avec traitement d'erreur              */
/* 31/05/89 PH.CAYPHAS                           */

#include <stdio.h>
#include "ryresponder.h"        /* for generic idempotent responders */
#include "ROSEAN-ops.h"         /* ROSEAN operation definitions */
#include "ROSEAN-types.h"       /* ROSEAN type definitions */

#define LMAX (sizeof (ut -> ut_line))
extern int  errno;

/* INITIALISATION DES VARIABLES DE DEFINITION DE L'APPLICATION */

static char *myservice = "roseanstore";
static int execuid = 1;
static int execgid = 1;


/* DECLARATIONS DES OPERATIONS */
int     op_ean ();

/* TABLE RECAPITULATIVE DES OPERATIONS */

static struct dispatch dispatches[] = {
    "ean", operation_ROSEAN_ean, op_ean,

    NULL
};


/* DECLARATION DES FONCTIONS DE MANIPULATION DES TYPES */
struct type_ROSEAN_IA5List *str2ia5list ();



/* MAIN */

main (argc, argv, envp)
int     argc;
char  **argv,
      **envp;
{
    ryresponder (argc, argv, PLocalHostName (), myservice, dispatches,
            table_ROSEAN_Operations, NULLIFP, NULLIFP);

    exit (0);                   /* NOTREACHED */
}


/*
```

```
      LES OPERATIONS A EFFECTUER POUR L'AUTRE SITE */

/* ENVOYER UN MESSAGE VIA UN INTERFACE EAN : INTEAN */

static int  op_ean (sd, ryo, rox, in, roi)
int     sd;
struct RyOperation *ryo;
struct RoSAPinvoke *rox;
caddr_t in;
struct RoSAPindication *roi;
{


    int     fd,
            i,
            result,
            vecp,
            pd[2];
    register char *bp,
                  *dp;
    char    *fromuser,
            *touser,
             buffer[BUFSIZ],
             data[BUFSIZ],
           **vec,
            *vecl[NVEC + 1],
            *prl[NVEC + 1];
    struct type_ROSEAN_IA5List  *ia5;
    register struct type_ROSEAN_IA5List  **ia5p;


    vecp = 0;
    if (rox -> rox_nolinked == 0) {
        advise (NULLCP, LOG_INFO,
                "RO-INVOKE.INDICATION/%d: %s, unknown linkage %d",
                sd, ryo -> ryo_name, rox -> rox_linkid);
        result = ureject (sd, ROS_IP_LINKED, rox, roi);
        goto out;
    }
    if (debug)
        advise (NULLCP, LOG_DEBUG, "RO-INVOKE.INDICATION/%d: %s",
                sd, ryo -> ryo_name);

    for (ia5 = (struct type_ROSEAN_IA5List  *) in; ia5; ia5 = ia5 -> next)
        if (vecp >= NVEC
                || (vecl[vecp++] = qb2str (ia5 -> IA5String)) == NULLCP)
            goto congested;
    vecl[vecp] = NULLCP;

    if (vecp < 3) {
        advise (NULLCP, LOG_INFO,
                "too few arguments (got %d, wanted at least 3)", vecp);
        result = ureject (sd, ROS_IP_MISTYPED, rox, roi);
        goto out;
    }
    fromuser = vecl[0];
    touser = vecl[1];
    vec = &vecl[1], vecp -= 1;
     prl[0] = vecl[1];
    ia5 = NULL;
    ia5p = &ia5;
    if (access ("/users/isode-4.0/rosean/intean",1) == NOTOK){
         result = error_ROSEAN_unableToAccessFile;
```

```
cops: ;
        free_ROSEAN_IA5List (ia5);
        ia5 = NULL;
        ia5p = &ia5;

        (void) sprintf (buffer, "%s: %s", "/users/isode-4.0/rosean/intean", sys;
        if ((*ia5p = str2ia5list (buffer)) == NULL)
            goto congested;
        ia5p = &((*ia5p) -> next);
        result = error (sd, result, (caddr_t) ia5, rox, roi);
        free_ROSEAN_IA5List (ia5);
        goto out;
    }
    if (pipe (pd) == NOTOK) {
        result = error_ROSEAN_unableToPipe;
        goto oops;
    }

    switch (fork ()) {
        case -1:
            (void) close (pd[0]);
            (void) close (pd[1]);
            result = error_ROSEAN_unableToFork;
            goto oops;

        case 0:
            closelog ();
            if ((fd = open ("/dev/null", 2)) != NOTOK) {
                if (fd != 0)
                    (void) dup2 (fd, 0), (void) close (fd);
            }
            (void) dup2 (pd[1], 1);
            (void) dup2 (pd[1], 2);
            (void) close (pd[0]);
            (void) close (pd[1]);
            if (execuid != 0) {
                (void) setgid (execgid);
                (void) setuid (execuid);
            }

            execvp("/users/isode-4.0/rosean/intean",vecp, prl);
            _exit (1);

        default:
            (void) close (pd[1]);


            for (dp = data;;)
                switch (i = read (pd[0], buffer, sizeof buffer)) {
                    case NOTOK:
                        i = errno;
                        (void) close (pd[0]);
                        errno = i;
                        result = error_ROSEAN_errorReading;
                        goto oops;

                    case OK:
                        (void) close (pd[0]);
                        if (dp != data) {
                            *dp = NULL;
                            if ((*ia5p = str2ia5list (data)) == NULL)
                                goto congested;
                            ia5p = &((*ia5p) -> next);
                        }
```

```
                    if (RyDsResult (sd, rox -> rox_id, (caddr_t) ia5,
                            ROS_NOPRIO, roi) == NOTOK)
                        ros_adios (&roi -> roi_preject, "RESULT");
                    free_ROSEAN_IA5List (ia5);
                    result = OK;
                    goto out;

/*
    */
                default:
                    for (bp = buffer; i > 0; bp++, i--)
                        switch (*bp) {
                            case '\n':
                                *dp = NULL;
                                if ((*ia5p = str2ia5list (data)) == NULL)
                                    goto congested;
                                ia5p = &((*ia5p) -> next);
                                dp = data;
                                break;

                            case NULL:
                                break;

                            default:
                                *dp++ = *bp;
                                break;
                        }
                    continue;
            }
    }

congested: ;

    free_ROSEAN_IA5List (ia5);
    result = error(sd, error_ROSEAN_congested, (caddr_t) NULL, rox, roi);

out: ;
    for (vecp = 0; bp = vecl[vecp]; vecp++)
        free (bp);
    return result;
}


/* FONCTIONS DE TRAITEMENT D'ERREURS */

/* ERREUR */

static int  error (sd, err, param, rox, roi)
int     sd,
        err;
caddr_t param;
struct RoSAPinvoke *rox;
struct RoSAPindication *roi;
{
    if (RyDsError (sd, rox -> rox_id, err, param, ROS_NOPRIO, roi) == NOTOK)
        ros_adios (&roi -> roi_preject, "ERROR");

    return OK;
}
```

```
/* REJECT */

static int  ureject (sd, reason, rox, roi)
int     sd,
        reason;
struct RoSAPinvoke *rox;
struct RoSAPindication *roi;
{
    if (RyDsUReject (sd, rox -> rox_id, reason, ROS_NOPRIO, roi) == NOTOK)
        ros_adios (&roi -> roi_preject, "U-REJECT");

    return OK;
}

/* FONCTION DE MANIPULATION DES TYPES */

struct type_ROSEAN_IA5List *str2ia5list (s)
char    *s;
{
    register struct type_ROSEAN_IA5List *ia5;

    if ((ia5 = (struct type_ROSEAN_IA5List  *) calloc (1, sizeof *ia5)) == NULL
        return NULL;

    if ((ia5 -> IA5String = str2qb (s, strlen (s), 1)) == NULL) {
        free ((char *) ia5);
        return NULL;
    }

    return ia5;
}
```