Technologie des applications client-serveur

UE RSX 102

Support de cours Tome 1

G. Florin, E. Gressier, S. Natkin

BIBLIOGRAPHIE

Les RFC UNIX/INTERNET

Les serveurs WEB ...

Coulouris , Dollimore "Distributed systems concepts and design" Addison Wesley 2000"

S. Natkin, "Les protocoles de sécurité de l'Internet" Dunod 2002.

Ed. Krol, “The whole Internet”, O’Reilly, 1992

R Orfali, D Harkey, J. Edwards, "The essential Client Server Survival Guide" Wiley 1995
En francais "Client-Serveur Guide de survie".

G Gardarin, O Gardarin, "Le client-serveur"
1996 Eyrolles

Introduction

Notions générales

Introduction
Situation actuelle

Mutation permanente des **concepts**, des **techniques** et des **organisations** associées aux applications informatiques.

Source principale un effort de **recherche** et de **production industrielle** sans précédent au niveau mondial qui permet:

- l'apparition de composants de rapidité et de complexité en croissance continue.

- la possibilité de développement de solutions logicielles et organisationnelles irréalistes quelques années auparavant.

Systèmes disponibles

On dispose à prix accessible pour les entreprises et le grand public de calculateurs puissants munis :

- de **systèmes d'exploitation** évolués

- d'interfaces **graphiques** évoluées

- de **capacités de stockage** énormes

- de moyens d'interconnexion "**réseaux locaux**" très performants

- dans un futur prévisible de moyens d'interconnexion à "**longue distance**" à bas prix et de bande passante importante.

Informatique "coopérative"

Applications coopératives ("Cooperative work")

Un ensemble d'entités logicielles **coopérant** au moyen d'un **réseau** à la réalisation d'une **tâche informatique**.

Algorithmique répartie ("Distributed Computing")

Les applications systèmes et réseaux indispensables au fonctionnement des machines en réseau.

IAD Intelligence artificielle distribuée, Multi-agent ("Distributed Artificial Intelligence")

Application coopérative mettant en relation des agents qui fonctionnent selon des approches dérivées de l'intelligence artificielle.

Informatique "coopérative" (suite)

Le calcul massivement parallèle ("Grid Computing")

Application de calcul scientifique réalisé par le travail en parallèle et en coopération d'un nombre élevé de processeurs.

Les systèmes répartis de contrôle commande de procédés industriels

Application de contrôle en temps réel de procédés tenant compte des contraintes de sûreté de fonctionnement.

Le client/serveur

Principal objet du cours

Le client serveur

Définir des **modèles de répartition**
des **données**
et des **traitements**
dans une **architecture réseau**.

Selon une approche **dissymétrique**:
- un client émet des **requêtes,**
- le serveur rend le **service** demandé.

Applicable à de nombreux problèmes de coopération,
en informatique de **gestion**
également aux autres domaines
algorithmique **répartie**,
informatique **industrielle**, ...

Dans son acception la plus complète:
possibilité de définir n'importe quelle **architecture de communication**.

- Chaque entité communicante est à la fois client et serveur
- Chaque entité émet et traite des requêtes

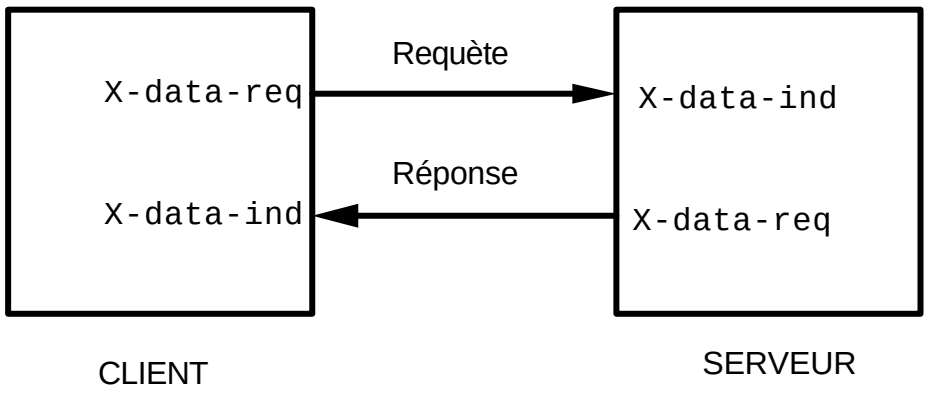
Communications et approche client-serveur

Client serveur en mode message

. Le client envoie dans un **premier message une requête** d'exécution d'un traitement à un serveur.

. Le serveur effectue le travail et fournit dans **un second message la réponse**.

On peut utiliser la communication par **messages** de données en mode **asynchrone** offerte par les protocoles de **transport**.

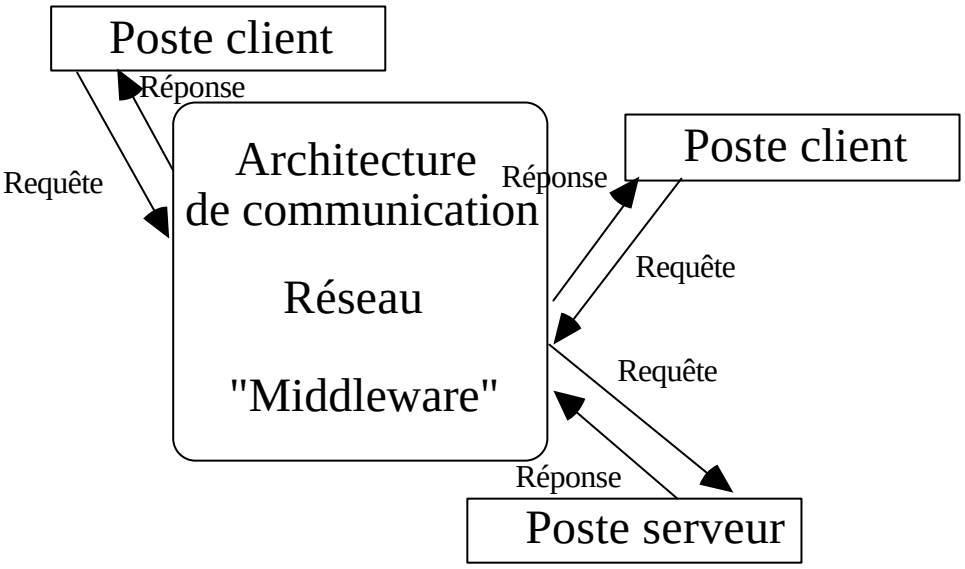


Client serveur en appel de procédure distante

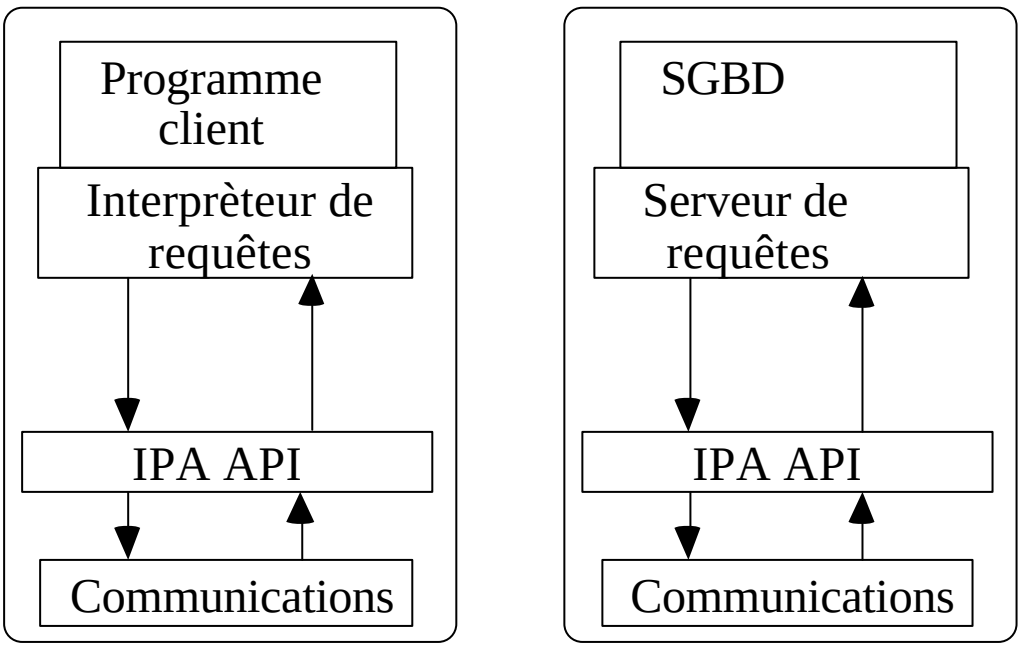
Présentation **syntaxique et sémantique** du mécanisme précédent **en terme d'appel de procédure distante.**

Architecture générale

Fonctionnement le plus fréquent en client-serveur: mode requête réponse pour une population de clients et de serveurs.



IPA Interface de programmation d'application API ("Application Programming Interface")



Applications types.

Transactionnel

OLTP "On Line Transaction Processing"

Les applications comportent:
- des opérations d'accès en lecture écriture à des bases de données,
- des traitements
- des affichages en mode graphique sur des postes de travail.

Aide à la décision

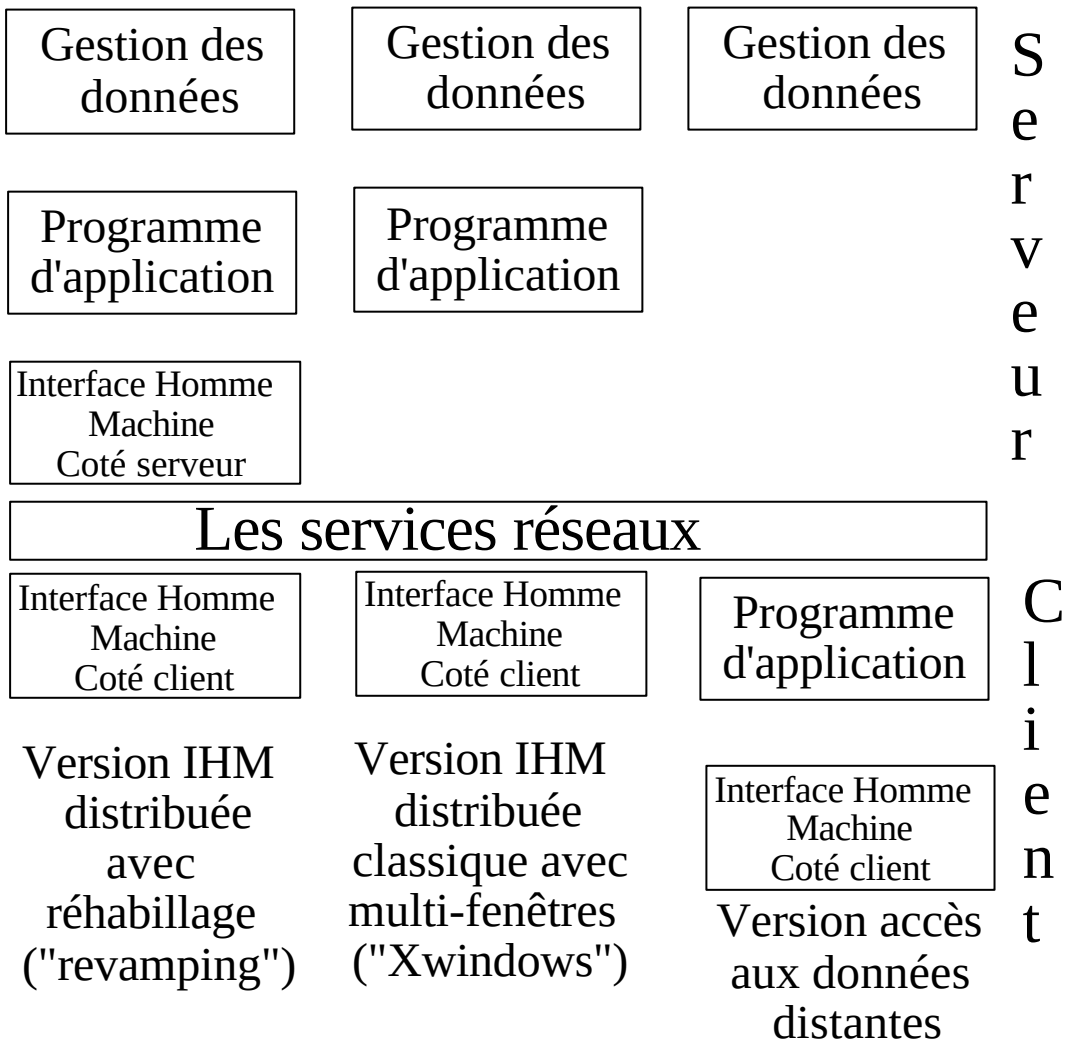
EIS/DS "Executive Information System/Decision Support")

Les applications comportent:
des accès complexes bases de données,
des traitements de synthèse/valorisation
des affichages (tableaux, histogrammes etc)

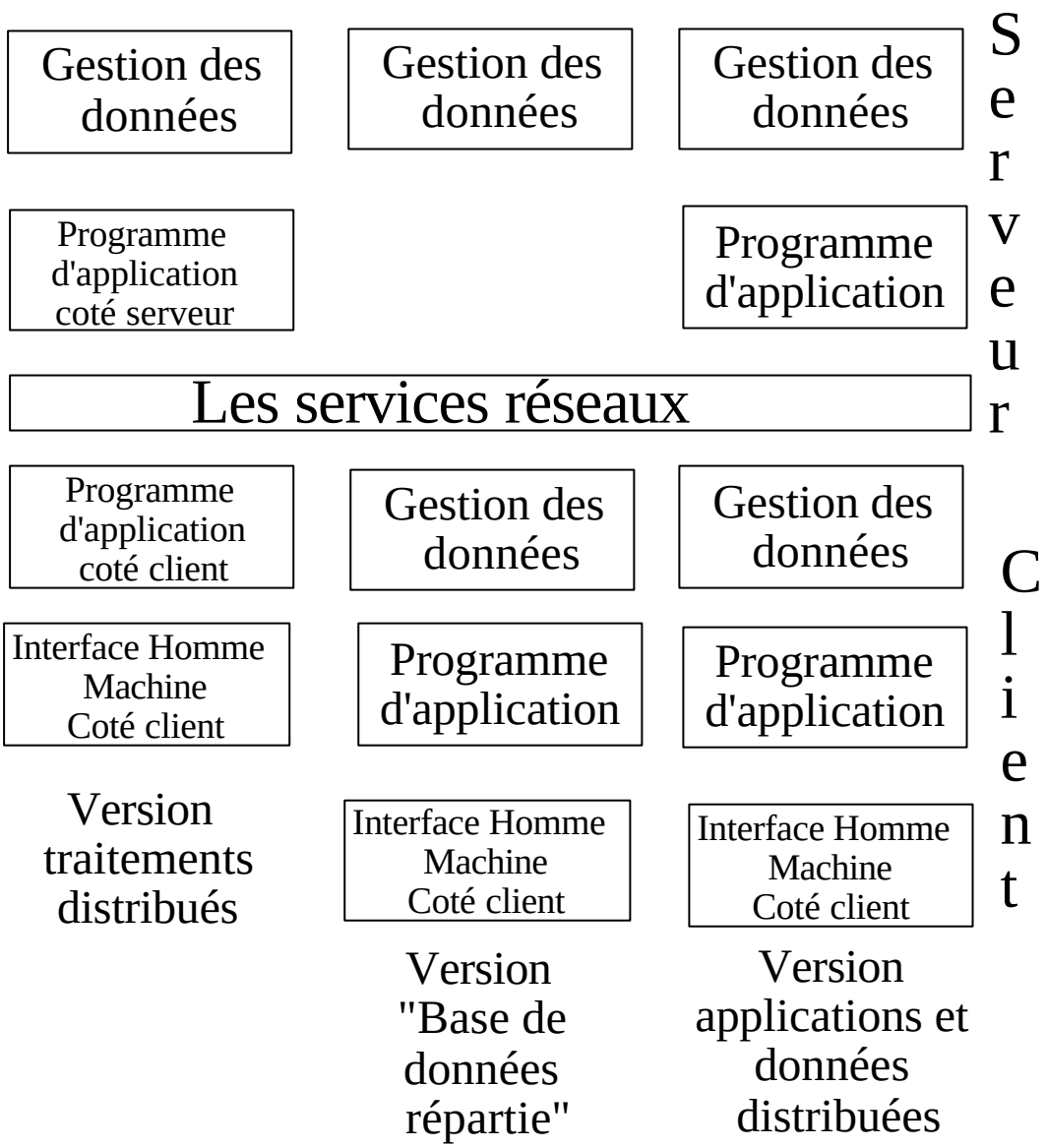
Le client-serveur en informatique de gestion

Les six versions du client serveur selon l'implantation des fonctions (Classification du Gartner Group)

Les trois versions de base



Les trois versions avancées



Les services réseaux orientés données ("Middleware de données")

Assurer les échanges de données entre un client et un serveur **en masquant les différents problèmes** potentiels liés à:

- la **répartition** des données et traitements (accès distant, baisse de performance)
- **l'hétérogénéité** des matériels et des logiciels en opération.

Exemple de travail le plus fréquent en client/serveur: envoyer des requêtes d'accès à des données (type SQL) d'un client vers un serveur et recevoir les résultats.

- **Ouvrir une connexion** entre entités
- Envoi de **requêtes SQL** vers le serveur
- **Conversion des formats** de requêtes
- **Exécution** des requêtes
- **Envoi** des résultats
- **Conversion des résultats**
- (Gestion des erreurs)
- **Fermeture de connexion**.

Problèmes à résoudre

Qui réalise les fonctions classiques des systèmes centralisés que l'on rencontre maintenant en univers réparti.

Quelques exemples

- Accès et partage des ressources

- Intégrité et cohérence des données

- Sécurité du système

- Gestion des performances

- Administration de l'ensemble

Fonctions qui regroupent des aspects multiples

- Contrôle d'exécution
- Contrôle de l'accès aux données
- Contrôle des communications

(à considérer simultanément).

Traitées par des outils différents

- Système d'exploitation local.

- Architecture de communication.

- Base de donnée (locale ou distribuée)

- Moniteur transactionnel

- Protocoles additionnels spécifiques

Un domaine considérable en cours d'établissement avec une terminologie considérable et très opaque (concepts, standards, produits)

Quelques exemples d'outils

- Les SGBD relationnels centralisés utilisés en accès distant au moyen du réseau par des stations (EDA-SQL + liaison APILINK).

- Les SGBD répartis (SGBD distribués accès à distance) (ORACLE).

- Les architectures de réseaux
(Internet avec l'application WEB + java).

- Les micro noyaux de systèmes (windows NT) répartis (Chorus).

- Les systèmes d'objets distribués
(OMG-CORBA, OLE COM).

Besoin d'une classification

La classification des réseaux

Différentes propositions de structuration des logiciels réseaux:

Solution **hiérarchique** (en couche).
Solution **"objet"**.

Normalisation / Système **propriétaire**

Expérimentation / Solution **confirmée**

Les piles de protocole

Une solution encore nécessaire pour comprendre les fonctions à réaliser:

- Choix du découpage des fonctions à réaliser entre **différentes couches de logiciels**.
- Choix de définition des **fonctions à réaliser par les différents niveaux**.
- **Implantation** des niveaux.

Exemple d'architecture importante:

INTERNET

INTERNET

<table>
<tr><td></td><td colspan="2">Applications TCP/IP directes</td><td>Applications pile SUN/OS</td></tr>
<tr><td>7. Application</td><td colspan="2" rowspan="3">EXEMPLES
SMTP "Simple Mail Transfer Protocol"
FTP: "File Transfer Protocol"</td><td>NFS: "Network File System"</td></tr>
<tr><td>6. Présentation</td><td>XDR:"External Data Representation"</td></tr>
<tr><td>5. Session</td><td>RPC:"Remote Procedure Call"</td></tr>
<tr><td>4. Transport</td><td colspan="3">TCP: Transmission Control Protocol (connecté)
UDP: User Datagram Protocol (non connecté)</td></tr>
<tr><td>3. Réseau</td><td colspan="3">IP: Internet Protocol</td></tr>
<tr><td rowspan="2">2. Liaison</td><td colspan="3">Encapsulation IP (sur LAN ou liaisons SLIP,PPP)</td></tr>
<tr><td rowspan="2">1. Physique</td><td colspan="3">Pratiquement tout support de transmission
Réseaux Publics Lignes spécialisées Point à Point Réseaux Locaux Réseau téléphonique RNIS, ATM</td></tr>
</table>

NIVEAU TRANSPORT (rappel)

. Le premier des niveaux utilisable par l'utilisateur pour développer des applications.

. **Il réalise un service de transmission fiable entre processus (transmission de bout en bout, "end to end").**

. Selon les options de conception il assure:

- **Gestion des connexions.**
  . protocoles en mode connecté
  . protocoles en mode non connecté.
- **Négociation de qualité de service**.
- **Multiplexage/éclatement**
- **Contrôle d'erreur**.
- **Contrôle de flux**.
- **Contrôle de séquence.**
- **Segmentation.**

Exemples de transports: Internet

TCP "Transmission Control Protocol".

UDP "User Datagram Protocol".

Novell

SPX "Sequenced Packet eXchange"

En fait XEROX XNS: SPProtocol

NIVEAU SESSION

Dans sa définition de base (OSI) le niveau session structure et synchronise les dialogues point à point en mode message.

L'approche OSI.

. Le transport offre **une voie logique** de communication ("**un tube**").

. Le mode de communication utilisé est **le mode message asynchrone.**

. **La session** structure les échanges pour y ajouter de la **tolérance aux pannes et de la synchronisation** :

- **Activités** (travail important)
- **Dialogue** (partie d'un travail)
- Notion de **point de synchronisation** pour délimiter des parties d'un échange en vue de la reprise.

Difficulté majeure de l'approche OSI Manque de **fonctions de synchronisation**. **Normes** CCITT X215 ISO 8026 Service X225, 8027 Protocole de session

APD Appel de Procédure Distante "RPC Remote Procedure Call"

. Le mode de communication du transport étant **le mode message asynchrone,** la session est définie pour offrir à l'usager un mode de dialogue de type synchrone.

=> Permettre à un usager d'exécuter une procédure ou une fonction sur un autre site:
. **Problème délicat :** Assurer en environnement réparti une sémantique pour l'appel de procédure distante voisine de celle connue en univers centralisé.

- Exemples : Rpc traditionnels
SUN-OS : Internet Protocole RPC
OSF-DCE : "Distributed Computing Environment"

- Exemples : Systèmes d'objets répartis
JAVA RMI :"Remote Method Invocation"
CORBA :"Common Object Request Broker Architecture"
WS-SOAP :"Web Services - Simple Object Access Protocol"

NIVEAU PRÉSENTATION

Traite du codage des données échangées: différents sites ayant des représentations différentes peuvent utiliser les données.

Les conversions

. Nécessaire pour tous les types de données
Types caractères, numériques, construits.

Syntaxe abstraite : Permet la définition d'une grande variété de structures de données (analogue de la syntaxe de définition de types dans un langage évolué).

Syntaxe de transfert : Représentation unique dans le réseau des valeurs échangées pour transférer les données.

Exemples de standards de conversion

- Réseaux publics

Syntaxe abstraite ASN1: **X208**

Syntaxe de transfert :**X209**

- SUN-OS/ Internet

XDR : "eXternal Data Representation".

- Systèmes d'objets répartis CORBA

IDL : "Interface Definition Language".

CDR : Common Data representation.

- Web services

WSDL : "Web Services Definition Language".

RPC/Encoded : "Syntaxe de transfert XML"

Les protocoles de sécurité

Objectif

Echanger des informations de façon sécuritaire en résolvant des problèmes de:

- **confidentialité**
- **intégrité**
- **authentification**

Problème central du développement des applications réseau par exemple de commerce électronique.

Exemples

Utilisation de techniques de cryptographie
DES, RSA, ...

Définition de protocoles de sécurité

Au niveau session SSL (Secure Socket Layer)

Aux autres niveaux (L2TP, IPSEC, SHTTP, Kerberos ….)

NIVEAU APPLICATION

Le niveau application est défini pour fournir à l'utilisateur des fonctions dont il a besoin couramment.

- **En termes d'un cadre** de développement d'une application informatique répartie (Exemple: structuration objet).
- **En termes de protocoles**

fonctions réseaux pré définies qui déchargent l'utilisateur de travaux répétitifs de programmation d'applications souvent utilisées.

On distingue aussi les applications qui concernent des échanges automatisés entre calculateurs de celles qui concernent le travail des personnels.

- **Travail Collectif,"Groupware"**

Échange de documents multimédia
Planification des activités ("Workflow")
Courrier électronique, Télé conférence
Gestion des agendas

Protocoles d'application

Le transfert de fichiers

Objectif

Déplacer des fichiers d'un site à un autre (plats en général).
Très nombreux protocoles proposés

Exemples

Internet FTP "File Transfer Protocol"
OSI/IEC FTAM "File Transfer Access and Management"

L'accès aux fichiers distants

Objectif

Accès unifié en univers réparti à différents fichiers (réalisation des requêtes d'accès).

Exemples

SUN-OS NFS "Network File System"
OSI/IEC FTAM "File Transfer Access and Management"

La gestion transactionnelle répartie

Objectif

La cohérence,
La persistance des données,
L'optimisation des performances.

Assurer le maintien cohérent d'un ensemble de données réparties en présence de nombreuses transactions concurrentes et de pannes.

Exemples

En univers Ouvert

Normes OSI/IEC TP et XOPEN/XA
"Transaction Processing"

Produits propriétaire:

Moniteurs IBM CICS, TUXEDO

L'accès aux bases de données distantes

Objectif

Permettre à un client d'accèder à une base de données distante (le plus souvent au moyen de requêtes SQL)

Exemples

Normalisation de facto Microsoft

ODBC

"Open Data Base Connectivity"

Proposition du groupe SAG "SQL Access Group" d'un ensemble de requêtes CLI "Call level Interface".

Autre proposition

Consortium Borland, IBM, Novell.
IDAPI : "Integrated Database Application Programming Interface"

Produits propriétaires

EDA-SQL de Information Builders Inc
Interface de communication API/SQL connectant de très nombreuses bases de données sur des plates formes clientes.

La désignation

Objectif

Gérer des **annuaires** permettant à un utilisateur de retrouver des attributs d'un nom symbolique (principalement l'adresse réseau).

Exemples

Internet **DNS** "Domain Name System"
OSI/IEC **DS** "Directory Services"
Annuaire **LDAP** "Lightweight Directory Access Protocol'

La messagerie

Objectif

Permettre d'échanger du courrier électronique entre usagers

Exemples

Internet **SMTP** "Simple Mail Transfer Protocol"
OSI **MHS** "Message Handling System"

L'échange de données informatisées

Objectif

Permettre d'échanger des documents administratifs standards sur des réseaux de transmission de données (commandes, factures, ...) entre agents économiques.

Exemple

Normes EDI "Electronic Data Interchange"

L'échange de documents électroniques

Objectif

Définir et échanger des documents généraux structurés de types particuliers (page, lettre, journal, livre, catalogue).

Exemples

HTML "HyperText Markup Language"
XML "eXtended Markup Language"
Très nombreuses propositions de formats.

L'accès aux informations distantes WWW "World Wide Web"

Objectif

Permettre l'accès à des données distantes pour des personnes.

- Systèmes de désignation **URL** ("Uniform Resource Locator"),
- Protocole de communication **HTT**P (Hyper Text Transfer Protocol"),
- Initialement format **HTML** + format **MIME** Extension format **XML**

Extension : permettre les communications entre programmes : notion de services web (**SOAP** ‘ Simple Object Access Protocol’, **WSDL** ‘Web Services Definition language’ **UDDI** 'Universal Description Discovery and Integration').

L'administration de réseaux

Objectif

Permettre l'accès à des variables gérées par des agents d'administration:

- Lecture de l'état d'un appareil, de statistiques de fonctionnement.
- Positionnement de variables.

Internet SNMP : 'Simple Network Management Protocol'.

WBEM: 'Web Based Enterprise Management'

Conclusion

Les réseaux et systèmes répartis

Un problème extrêmement complexe au centre de l'informatique actuelle et à venir.

Une évolution permanente des concepts des outils et des différentes propositions commerciales.

Un marché très concurrentiel qui induit une compétition souvent inutile dans l'offre, une opacité importante dans les fonctions offertes par les produits.

Premier Chapitre

L'INGÉNIERIE DES PROTOCOLES DE COMMUNICATION

Plan du cours

Première partie

I.1 Le mode message asynchrone.

I.2 L'interface socket

Seconde partie

II.1 Le mode appel de procédure distante.,

II.2 Exemple: CORBA

INTRODUCTION

- **Le contrôle réparti** c'est l'ensemble des moyens offerts à un programmeur pour **développer des applications dans un univers réparti** (réseau, système réparti,..)

Application utilisateur

Interface de Programmation d'Applications **(IPA - API)**

Mécanisme d'exécution
Mécanisme de communication
Mécanisme de mémorisation

Démarche descendante du développement logiciel

- La conception de l'application.

- Les étapes de raffinement.

- La programmation : l'adaptation finale aux outils disponibles (syntaxe et sémantique de l'IPA)

I Les moyens de développement disponibles

Introduction Aux IPA - Distribuées

"DAPI Distributed Application Programming Interface"

Syntaxe et sémantique des moyens de programmation **utilisables pour l'expression d'un comportement réparti**

Comportant donc nécessairement des fonctionnalités de **communication et de synchronisation.**

=> La description des interfaces de programmation d'application distribuées devrait comprendre:

. **Les comportements en mode normal** de fonctionnement.

. **Les comportements en présence de pannes.**

. **Le comportement temporel.**

Styles de description des interfaces

Approches services systèmes/réseaux

Les plus répandues:

Sémantiques plus simples

Accessible de tous les langages

Mais contrôles limités

Problèmes de désignation.

Exemples. Interface socket (**TCP** Internet).

Approches langages

Très nombreuses propositions

Vérifications de cohérence

Désignation plus facilement résolue

(pour une application complètement définie)

Syntaxes/Sémantiques complexes

LA COMMUNICATION DANS L'INTERFACE DE PROGRAMMATION

- La communication par messages : l'outil de base (Système RC4000 [Brinch Hansen 1970])

- Développement d'interfaces de programmation d'applications réparties de complexité croissante:

En termes de richesse du schéma de communication/synchronisation offert.

Impliquant un nombre de messages échangés de plus en plus élevé.

. Le mode message asynchrone.

. Le mode rendez-vous.

. Le mode appel de procédure distante.

. Le mode mémoire partagée répartie

=> Modes de communication encore le plus souvent définis en point à point.

=> Evolution en cours vers des interactions de groupes.

Critères de comparaison des outils

Facilité d'utilisation

Syntaxe des primitives de service

Compréhension de la sémantique

Simplicité de la désignation.

Facilité d'extension à l'univers réparti d'applications existantes

Efficacité (performances)

Richesse des mécanismes

de synchronisation (des propriétés d'ordre)

Interopérabilité (normalisation réelle)

Comparaison des interfaces

Chaque mode présente des **avantages et des inconvénients.**

Possède des **défenseurs et des détracteurs**

L'étude comparative des différents modes **est très délicate.**

La situation est très **évolutive.**

=> Pas d'inutiles débats.

=> Faire des essais.

=> Disposer de plusieurs modes.

Utilisation de modes différents.

Problèmes à résoudre :

Pour une utilisation simultanée de **plusieurs modes** de communication.

=> Développement de **méthodologies** de spécification, conception, programmation **supportant plusieurs sémantiques.**

I.1

LE MODE MESSAGE ASYNCHRONE

INTRODUCTION

- Basé directement sur le mode de communication par message (un message).

- Un service comprenant essentiellement deux primitives pour communiquer et se synchroniser.

TYPE COM_MESSAGE_ASYNCHRONE;
    METHOD envoyer (id_émetteur, id_récepteur, message, compléments);
    METHOD recevoir (id_émetteur, id_récepteur, message);
    METHOD ...;
END COM_MESSAGE_ASYNCHRONE.

identificateurs : ports de communication
messages : zones de données (typées).
compléments : selon les sémantiques multiples définissant les qualités de services

Nombreuses variantes sémantiques du mode message asynchrone

- Propriétés **d'ordre**.

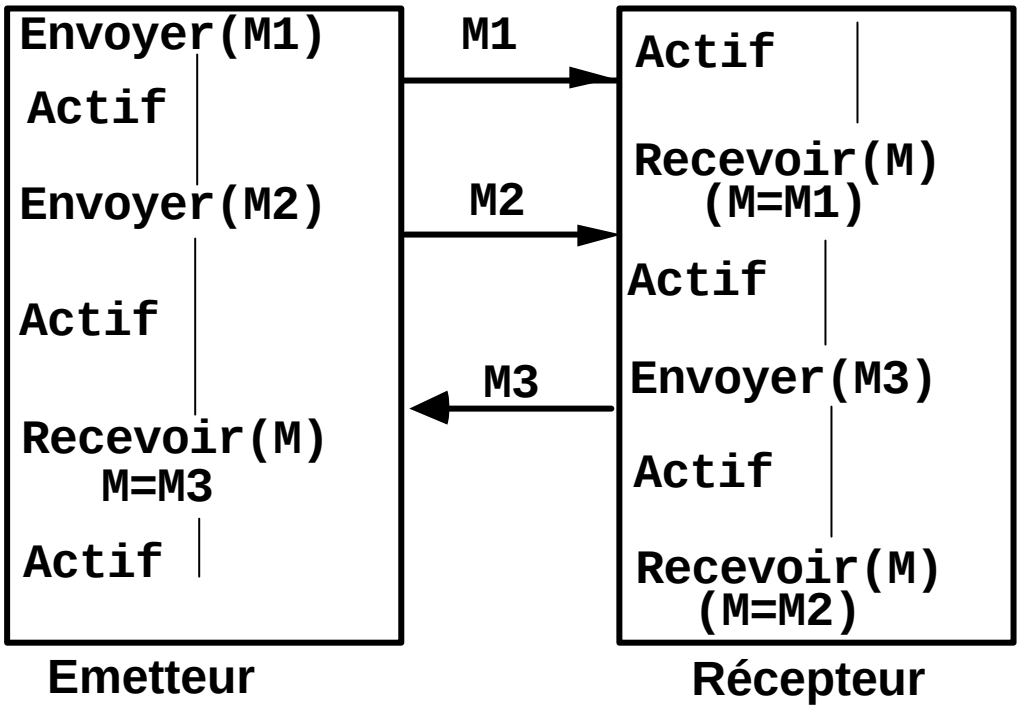
- Propriétés **de tolérance aux pannes**.

- Propriétés **temporelles**

- Autres **qualités de service.**

- Problème abordé ici: **la synchronisation et l'utilisation des primitives**.

Aspect principal pour la communication et la synchronisation: l'asynchronisme entre l'émetteur et le récepteur



- Le mode message asynchrone réalise un **"producteur-consommateur" réparti** entre un émetteur et un récepteur.

- Complet parallélisme autorisé entre l'émetteur et le récepteur.

Rien n'empêche aussi l'une des entités (l'émetteur) de suspendre son exécution.

Détails du comportement

L'émetteur

. Ayant demandé une émission, **reprend la main et continue son exécution** ***"immédiatement"*** **après** la prise en compte de sa demande.

. Le message est transmis (au rythme du transport d'informations par le réseau de communication) donc de façon **asynchrone** avec le comportement émetteur**.**

Le récepteur

Ayant décidé de prendre en compte un nouveau message il acquiert un message (le premier) en instance.

-celui qui se présente à après le recevoir

-un message qui se trouve dans une file d'attente de réception.

Utilisation du mode message asynchrone Aspect d'activation à distance

- Les messages sont le plus souvent typés:

Ils sont **une demande de traitement distant** associé au typage

Utilisant les données du message pour paramétrer l'exécution distante.

- La **nature du traitement déclenché** dépend du contexte dans lequel le message est pris en compte.

Sémantique de l'activation

a) De type **activation en parallèle ("fork")** puisque en mode normal l'émetteur et le récepteur continuent.

b) Interprétable également comme un **branchement inconditionnel ("goto")** à distance si l'émetteur se suspend définitivement après l'émission.

Aspect d'échange de données

- Le mode message permet une opération d'affectation de variable à distance:

envoyer (M=écrire(v) , id_dest)

recevoir (M'=lire(v) , id-emet)

. Avec possibilité de **gestion de cohérence des types** des variables v.

. Avec **une sémantique de consistance des données entre l'émetteur et le récepteur très faible.**

Repose sur les propriétés d'ordre, temporelles, de tolérance aux pannes spécifiées par la qualité de service.

Propriété minimale de cohérence des communications point à point: la causalité

envoyer (M=écrire) -> recevoir (M'=lire)

$ t1 : émetteur.écrire (v,t1)

=> $ t2>t1 : recepteur.lire (v, t2)

Propriétés supplémentaires: rendez-vous, mémoire répartie.

Spécification des applications utilisant le mode message asynchrone

Principes Généraux

Solution Des Automates Synchronisés

Comportement séquentiel des processus:

=> Ensemble de processus (entités réseaux) **séquentiels** communicants (très souvent en mode point à point deux entités seulement).

=> Les méthodes de spécification de chaque processus utilisent des **automates d'état fini**

Interaction entre processus par échange de message

=> Les processus communicants se synchronisent par messages asynchrones.

Représentation graphique des applications: automates

États

Il est défini par **un ensemble significatif de variables locales** de chaque processus.

Chaque état doit être **interprétable aisément** en terme d'évolution locale de l'application répartie.

Choix des états:

Étape préliminaire importante de la spécification

. Pas un niveau de détail trop fin (trop grand nombre d'états) mais suffisamment pour représenter l'évolution au niveau de détail souhaité.

Transitions.

Elles comportent deux mentions:
les conditions, et les actions.
Elles sont représentées par les arcs du graphe.

Condition

Les transitions sont franchissables lorsqu'une condition booléenne ou garde est satisfaite:

. condition booléenne portant sur les variables locales.

. condition booléenne portant sur les échanges (requête de service, message entrant).

. condition booléenne portant sur les signaux internes (horloge).

Action

- C'est l'opération réalisée lors du franchissement de la transition: les traitements à réaliser lors du passage à l'état suivant (manipulation de variable, envoi de message).
- Le détail des actions ne doit pas être important sinon le modèle est illisible (renvoyer à des textes).

Les commandes d'échanges

- Représentation des opérations sur les diagrammes (conditions et actions)

condition recevoir(M) ?M

action envoyer(M) !M

- Désignation

- La commande d'émission (d'une entité P1) doit désigner un destinataire (P2).

Ex :Dans le processus P1 :P2!M1

- La commande de réception (exécutée par un processus P2) doit évoquer une source P1.

Ex :Dans le processus P2 :P1?M1

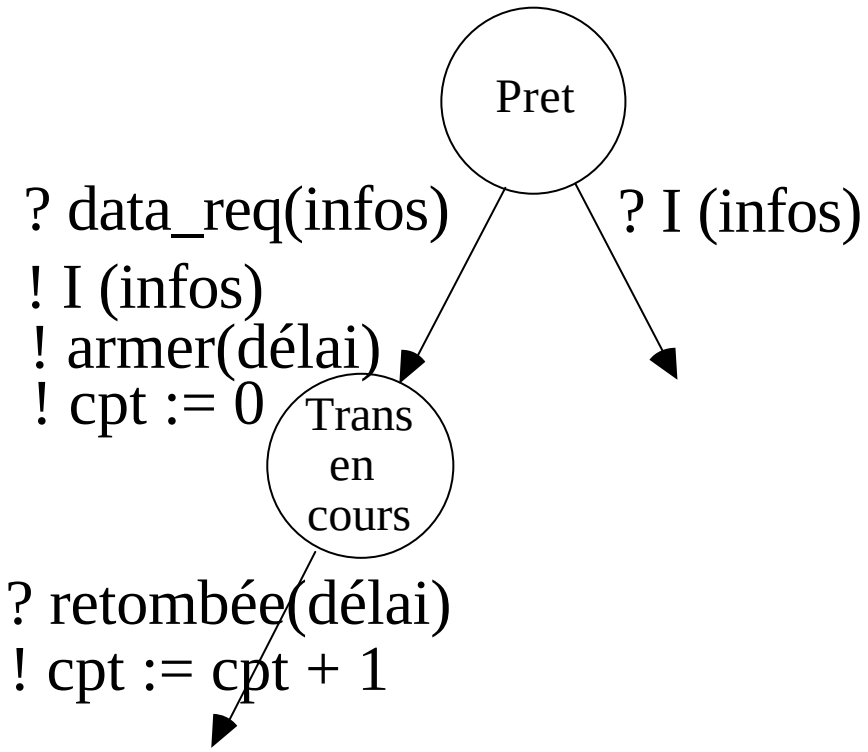
Typage

Il doit y avoir correspondance de type entre la variable de réception et la valeur émise (possibilité de variable article).

Type[message_émis] =

Type [message_reçu]

Exemple



Sémantique de l'alternative constituée par l'existence de plusieurs transitions en un état

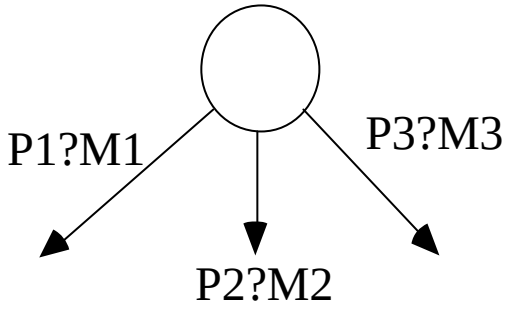
Évaluation des gardes (conditions booléénnes)

. Une garde est **franchissable** si elle est constituée par **une expression booléenne** portant sur des variables locales à **valeur vrai**.

. Une garde est **franchissable** s'il s'agit d'une **commande d'échange** satisfaite : en fait une réception dont le processus source à fait parvenir un message du type attendu qui se trouve en tête de la file d'attente.

. Une garde **combinaison des deux cas précédents.**

Alternative en un état



Indéterminisme de l'évaluation

- On choisit **l'une quelconque des transitions ayant sa condition booléenne satisfaite** (sa garde ouverte). Une garde franchissable quelconque est sélectionnée

=> **La liste des commandes** associée à la garde est **exécutée**.

- Aucune garde n'est franchissable mais il existe des gardes avec condition de réception pouvant être satisfaite par une émission

=> **Attente que l'une d'elles soit satisfaite.**

- Aucune garde n'est franchissable:

. elles sont toutes booléennes

. aucun message ne sera envoyé qui corresponde à une réception attendue

=> **Erreur d'exécution.**

Différents automates

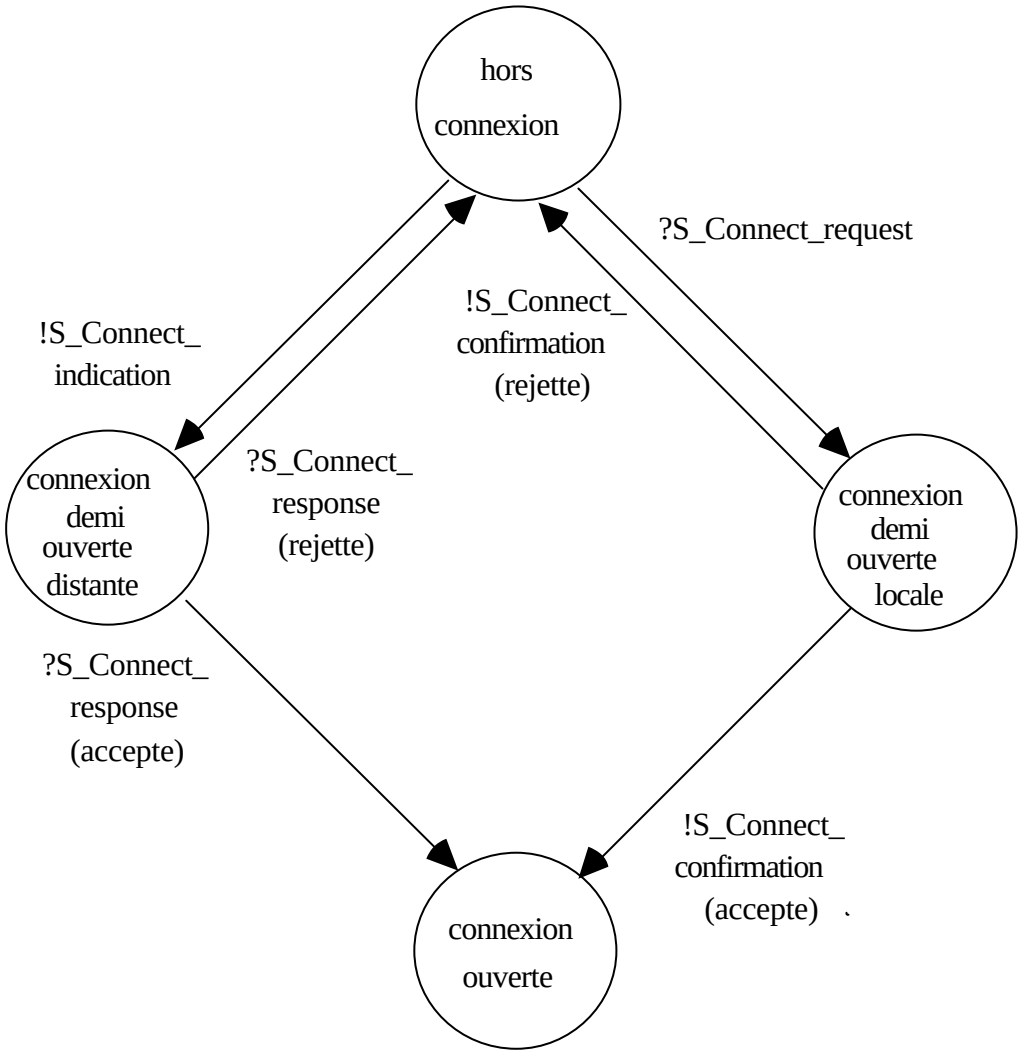
Automate du service

. Établissement de **la liste des unités de service** (primitives, SDU) de service échangeables entre le niveau n et le niveau n+1.

. **Définition des enchaînements autorisés** de primitives sous la forme d'un automate d'état.

=> Toutes les successions légales qu'un observateur de l'interface n n+1 peut observer.

Exemple de comportements autorisés du service de connexion de session OSI.



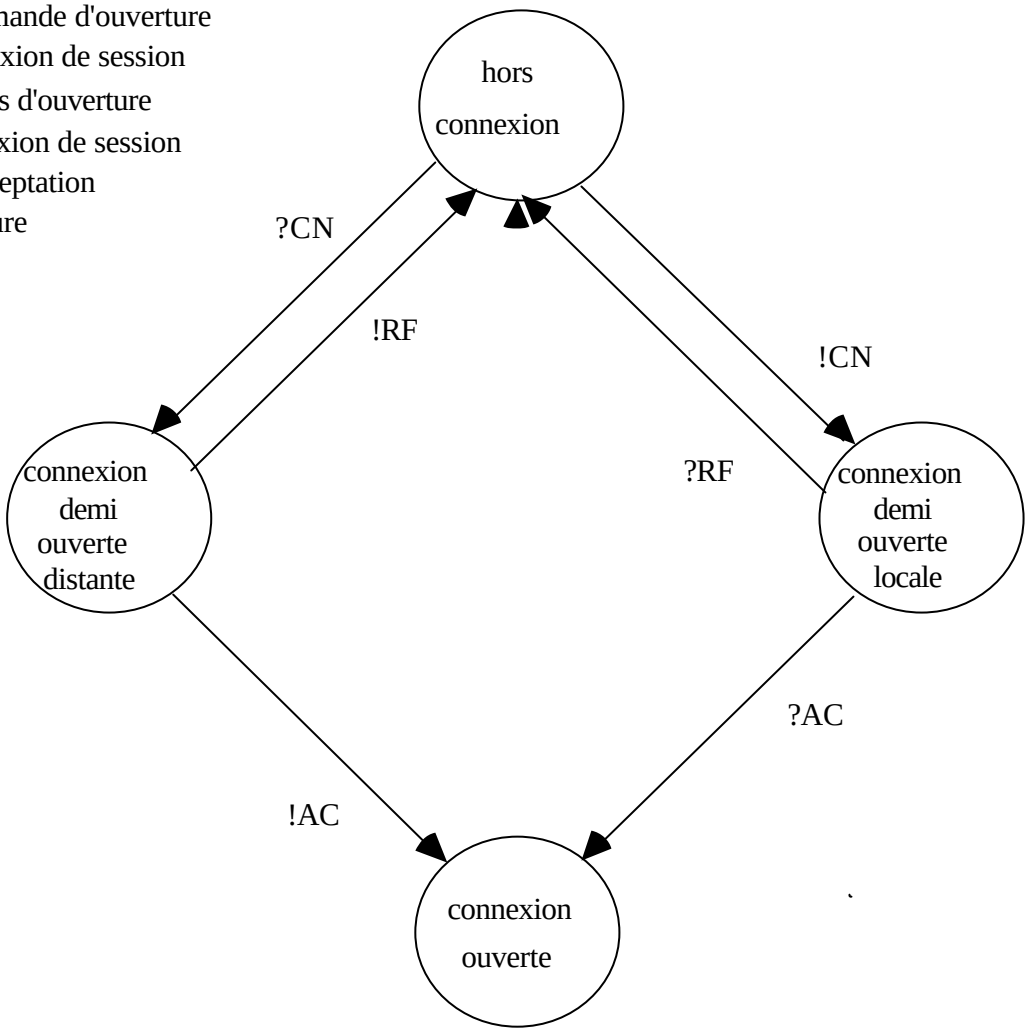
Automate du protocole

. Établissement de la liste des unités de protocole (messages, PDU) échangés entre deux niveaux n.

. Définition de tous les enchaînements de PDU qu'un observateur de la voie de communication peut observer).

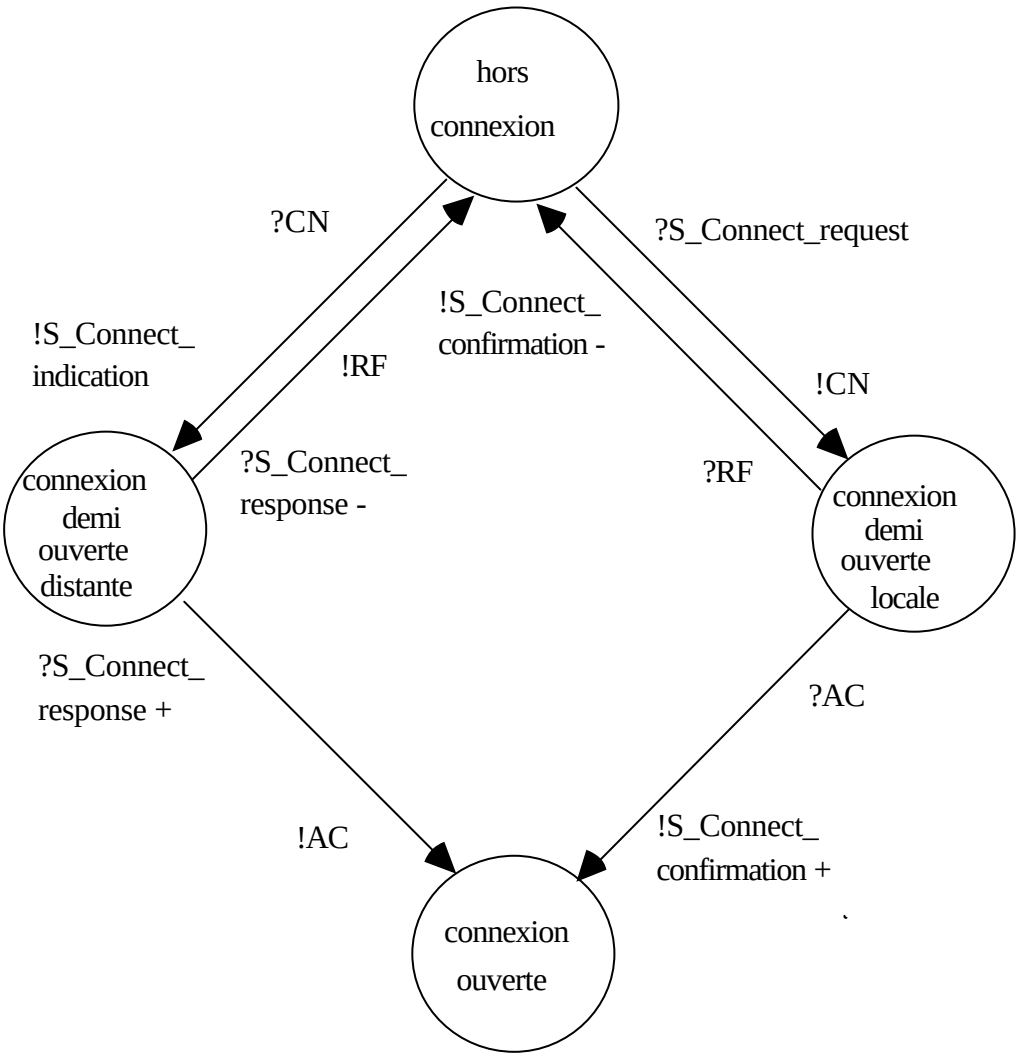
Ex: Une partie du protocole d'ouverture de connexion de session

CN: Demande d'ouverture de connexion de session
RF: Refus d'ouverture de connexion de session
AC: Acceptation d'ouverture



Automate complet

. Réunion des deux automates de service et de protocole



Validation des spécifications

Problèmes de constructions

Réceptions non spécifiées

Dans un état un message peut se présenter dont le cas n'a pas été prévu

Interblocages

Dans un état un message (ou une configuration) est attendu qui ne peut jamais se présenter.

Plus généralement

Définition d'assertions de bon fonctionnement sur le modèle à automates communicants.

- assertions portant sur des variables d'état (booléennes)

- assertions portant sur des trajectoires (logique temporelle)

Méthodes de validation systématiques employées

Simulation comportementale

Exécution en **simulation** du comportement décrit par les automates: une partie seulement des comportements sont couverts.

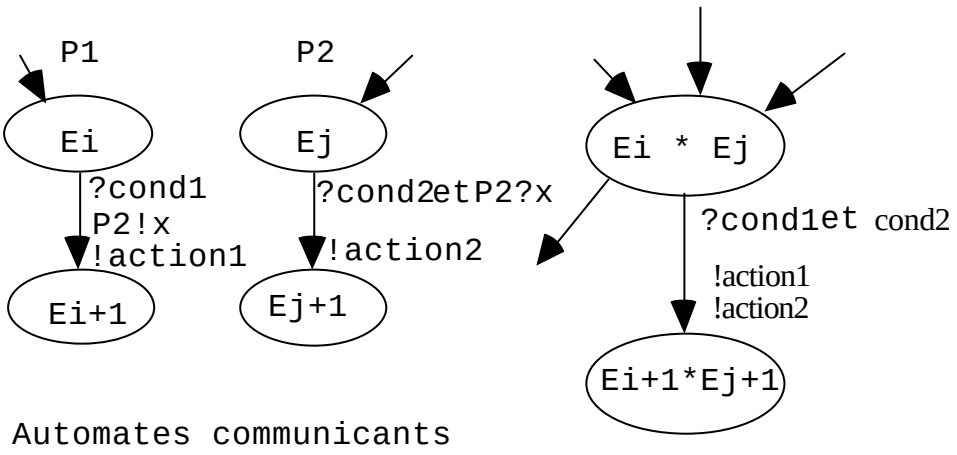
Détection d'erreurs et correction (technique s'apparentant au test du modèle).

Difficulté majeure : On ne connaît pas la couverture du test (le taux d'erreurs résiduelles après le test).

Analyse exhaustive.

- L'application se présente comme un ensemble d'automates communicants

- Réalisation du produit des automates (produit "synchronisé" tenant compte des communications).



Automates communicants

- Obtention du graphe d'accessibilité du système complet

- Vérification d'assertions sur le graphe complet.

Difficulté majeure

Explosion combinatoire de la taille des espaces d'état.

Exemples d'IPA distribuées en mode message asynchrone

Presque toutes les IPA des architectures "ouvertes" de réseaux

- **Internet, OSI, SNA**

Presque toutes les IPA des systèmes répartis classiques,

- **Chorus , Mach, Amoeba**

De nombreux langages de programmations

- **Langages de spécifications de protocoles** (Estelle, ...)

- **Langages "acteurs" de l'intelligence artificielle distribuée** (Act1, ...)

Z; Font exception : les propositions récentes qui abandonnent le mode message:

Les interfaces **orientées RPC** (**orientées objets**), orientées **transactionnel** (**partage de mémoire)**.

Approfondissement
Les interfaces de mode message dans les architectures de réseaux

Deux approches principales (beaucoup d'approches annexes)

Comme interface de service du niveau transport.

Comme interface de service du niveau application => offrant un mode unifié de communication par messages

Les interfaces de "transport" en mode message asynchrone

"**End to end communication**"
Communication de bout en bout
"**Peer to peer communication**"
Communication d'égal à égal

- Existence **de piles de protocoles** de couches basses résumées au niveau transport
TCP/IP, IPX/SPX, NetBIOS, ...

- Existence **d'interfaces logicielles pour l'accès à des piles** de protocoles : API utilisable de préférence pour plusieurs piles
Sockets, TLI, NetBEUI, APPC/CPI-C, ...

<table>
<tr><td colspan="2" rowspan="2">Transport</td><td>NetBEUI</td><td colspan="2">Sockets TLI</td><td>Tubes nommés</td><td>CPI-C APPC</td></tr>
<tr><td rowspan="4">NetBIOS</td><td>TCP</td><td>SPX</td><td rowspan="2">LU 6.2 APPN</td></tr>
<tr><td colspan="2">Réseau</td><td>IP</td><td>IPX</td></tr>
<tr><td rowspan="2">Liaison</td><td>LLC</td><td colspan="3">IEEE 802-2</td></tr>
<tr><td>MAC</td><td>MAC</td><td colspan="2">LLC / SNAP</td></tr>
<tr><td colspan="2">Physique</td><td>RNIS</td><td>Réseau Local</td><td colspan="2">Voies point à point ATM, ..</td></tr>
</table>

Etat actuel des API de transport

"Sockets" (prises)

Interface de programmation pour la suite TCP/IP. 1981. Système UNIX Berkeley BSD. Le standard UNIX de facto.

L'API sockets sous Windows est baptisée WinSock.

TLI ("Transport Layer Interface")

Proposition AT&T d'une interface TCP IP plus performante et plus indépendante du réseau sous-jacent (1986, 25 primitives).

NetBEUI "NetBios Extended Basic Input Output System"

Interface de programmation introduite en 1984 pour le PC Network (utilisé pour la pile NetBIOS") pile de protocoles pour les réseaux locaux (IBM et Microsoft).

Utilisé pour la pile de communication SPX/IPX de Novell ("Sequenced Packed Exchange/ Internet Packet Exchange").

Les tubes nommés ("Named pipes")

Interface d'échange entre processus IPC ("InterProcess Communication") introduite sous UNIX BSD pour étendre la notion de tube. Ils permettent de considérer les échanges réseaux comme des accès fichiers.

Disponibles sur TCP/IP, SPX/IPX.

APPC et CPI-C

("Advanced Program to Program Communication" et "Common Programming Interface for Communication")

Evolution de SNA vers une architecture réseau incorporant tout type de matériels grands, moyens et petits systèmes sous le nom d'APPN ("Advanced Peer to Peer Network").

APPC est une interface de programmation pour l'accès aux services SNA LU6.2 pour tous les types de produits (60 primitives pas tout à fait compatibles).

CPI-C (40 primitives) simplifie l'interface et masque les différences.

Les nouvelles interfaces d'application en mode message

Notion de MOM

"Message Oriented Middleware"
Une solution basée sur les files de messages ("message queues")

- Existence de propositions spécifiques permettant les communications de mode message asynchrone.
- Une proposition à volonté unificatrice: **les files d'attentes de messages** ("Message queue"). 1993. Consortium IBM etc avec normalisation OSI.
- MOM: une Interface universelle pour des échanges en mode message asynchrone via des files d'attente (**qui cache les différentes API transport**).
- Les files sont **persistantes** (sur disque) ou non persistantes. Un site qui n'est pas opérationnel sera atteint lors de son réveil (similaire à une messagerie).

Conclusion : mode message asynchrone

- C'est le mode le plus basique

Comparable à l'assembleur (affectation, branchement).

- Le moins contraignant il permet aux utilisateurs par des échanges successifs, la construction de tout type de protocole.

- L'utilisateur n'a pas en général envie d'être obligé de construire ses propres outils d'où:

L'enrichissement du mode message en termes de qualité de service par les couches successives des protocoles réseau.

Le besoin d'autres schémas prédéfinis plus complexes.

- Le mode message asynchrone est encore **le mode privilégié des interfaces**

On peut prévoir à terme son "**enfouissement**" dans les couches internes.

Communication en mode message asynchrone

Exemple de service TCP/UDP:

L'interface SOCKET Berkeley

Généralités interface "socket"

Définie en 1982 comme interface de programmation d'applications réseaux (API) pour la version UNIX Berkely (BSD).

Existence de plusieurs autres interfaces d'accès possibles (TLI, NETBIOS, ...)

Objectifs généraux

. Fournir des moyens de communications entre processus (IPC) **utilisables en toutes circonstances**: échanges locaux ou réseaux.

. Chercher à cacher au maximum **les détails d'implantation** des couches de transport aux usagers.

. Si possible chercher à cacher les **différences entre protocoles de transport hétérogènes** sous une même interface (TCP, Novell XNS, OSI, ...)

. Fournir une interface d'accès qui se rapproche **des accès fichiers pour simplifier la programmation**.

=> En fait des similitudes et des différences majeures entre sockets et fichiers.

Choix de conception des sockets

Une "socket" (prise) est un point d'accès de service pour des couches transport essentiellement TCP/UDP mais aussi d'autres protocoles (OSI, DECNET...).

- La caractéristique principale d'une socket est donc son **type**:

Pour quel protocole de transport est-elle un point d'accès de service?

Quelle est la sémantique de l'accès de service?

- Une socket possède un nom: un identifiant unique sur chaque site (en fait un entier sur 16 bits) appelé "**numéro de port**".
- Une socket est caractérisée par un **ensemble de primitives** de service pour l'accès aux fonctions de transport.
- Une socket encapsule des données:

un descriptif (pour sa désignation et sa gestion)

des files d'attente de messages en entrée et en sortie.

Désignation des sockets

- Pour identifier complètement une socket dans un réseau et pour une couche transport il faut un couple qui l'identifie de façon unique dans ce réseau pour ce transport:

Exemple Internet avec TCP:
(N° Port TCP , @IP)
(Numéro de port TCP , Adresse IP)

- Certains numéros sont réservés pour des services généraux et sont des ports **bien connus** ou "well-known ports".

Exemples: Sockets UDP
Echo server: 7,
DNS: 53,
TFTP: 69.

Exemples: Sockets TCP
FTP: 21,
Telnet: 23,
DNS: 53,
HTTP: 80

Choix de conception des sockets avec TCP

- TCP est un transport **fiable** en **connexion** et en mode **bidirectionnel point à point**.

- Une socket TCP peut être utilisée par **plusieurs connexions** TCP simultanément.

- Une **connexion est identifiée par le couple** d'adresses socket des deux extrémités.

- Un échange TCP est orienté **flot d'octets**.
Les zones de données qui correspondent à des envois successifs ne sont pas connues à la réception.

Pour optimiser TCP peut tamponner les données et les émettre ultérieurement.

L'option **"push"** permet de demander l'émission immédiate d'un message.

L'option **"urgent"** permet l'échange de données exceptionnelles avec signalement d'arrivée.

Choix de conception des sockets avec UDP

- UDP est une couche transport **non fiable, sans connexion,** en mode **bidirectionnel** et **point à point**.

- L'adresse UDP d'une socket sur l'Internet est identique à celle d'une socket TCP.
Rappel: les deux ensembles d'adresses sont indépendants:

(N° Port UDP , @IP)

(Numéro de port UDP , Adresse IP)

- Un échange UDP est sans connexion (échange de **datagrammes)**.

- Les zones de données qui correspondent à des envois successifs sont **respectées** à la réception.

Les primitives de l'interface socket

Exemples en langage C sous UNIX.

socket

- Permet la création d'un nouveau point d'accès de service transport:
  définition de son type.
  allocation de l'espace des données.
- Trois paramètres d'appel
  . "**Famille**" d'adresses réseaux utilisées
  locale, réseau IP, réseau OSI ...
  . **Type** de la socket (du service)
  sémantique de la communication.
  . **Protocole** de transport utilisé.
- Un paramètre résultat:
  le numéro de descripteur socket.

- Profil d'appel de la primitive en C

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
int socket (  int famille,
              int type,
              int protocole);
```

Aprofondissement des paramètres de la primitive socket

Paramètre Famille

AF_UNIX : Communication locale (i-node)
AF_INET : Communication Internet
AF_ISO : Communication ISO
....

Paramètre Type

- **SOCK_STREAM** : Flot d'octets en mode connecté (ne préserve pas les limites de l'enregistrement)
- **SOCK_DGRAM** : Datagramme en mode non connecté (préserve les limites de l'enregistrement)
- **SOCK_RAW** : Accès aux couches basses.
- **SOCK_SEQPACKET** : Format structuré ordonné (protocoles différents de l'Internet)

Paramètre Type de protocole

Valeur	Relation avec le paramètre type
IPPROTO_TCP	SOCK_STREAM
IPPROTO_UDP	SOCK_DGRAM
IPPROTO_ICMP	SOCK_RAW
IPPROTO_RAW	SOCK_RAW

bind

- Primitive pour l'attribution d'une adresse de socket à un descripteur de socket.

- Ceci n'est pas fait directement lors de la création du descriptif.

. Un serveur (qui accepte des connexions) doit définir sur quelle adresse.

. Un client (qui ouvre des connexions) n'est pas forcé de définir une adresse (qui est alors attribuée automatiquement).

- Profil d'appel de la primitive

```
#include <sys/types.h>
#include <sys/socket.h>

int bind (int s,
       struct sockaddr_in *mon_adresse,
       int longueur_mon_adresse)
```

- Trois paramètres d'appel

. Numéro du descriptif de Socket (s).

. Structure de donnée adresse de socket
Pour internet type sockaddr_in.

. Longueur de la structure d'adresse.

Aprofondissement concernant la primitive bind

- Descripteur d'adresse de socket pour les protocoles Internet.

```
#include <sys/socket.h>
struct sockaddr_in                    {
  short               sin_family;
  u_short             sin_port;
  struct in_addr      sin_addr;
  char                sin_zero[8];   };
```

- Un exemple d'exécution de "bind" pour les protocoles Internet.

```
struct servent      *sp
struct sockaddr_in sin
/* Pour connaître le numéro de port */
if((sp=getservbyname(service,"tcp")==NULL)
                    Cas d'erreur
/* Remplissage de la structure sockaddr */
/* htonl convertit dans le bon ordre */
/* INADDR_ANY adresse IP du site local */
sin.sin_family= AF_INET;
sin.sin_port = sp -> s_port;
sin.sin_addr.s_addr=htonl(INADDR_ANY):
/* Création d'une socket internet */
if ((s=socket(AF_INET,SOCK_STREAM,0))<0)
                    cas d'erreur
/* Attribution d'une adresse */
```

```
if (bind(s, &sin, sizeof(sin)) < 0)
                        cas d'erreur
```

listen

- Utilisé dans le mode connecté lorsque plusieurs clients sont susceptibles d'établir plusieurs connexions avec un serveur.

- Celui ci indique le nombre d'appel maximum attendu pour réserver l'espace nécessaire aux descriptifs des connexions.

- La primitive listen est immédiate (non bloquante).

- Profil d'appel

```
int listen(int s , int max_connexion)
```

s : Référence du descripteur de socket
max_connexion : Nombre maximum de connexions.

accept

- Dans le mode connecté la primitive **accept** permet de se bloquer en attente d'une nouvelle demande de connexion

- Après l'accept, la connexion est complète entre les deux processus.

- Le site qui émet accept exécute une ouverture passive.

- Pour chaque nouvelle connexion entrante la primitive fournit un pointeur sur une nouvelle socket qui est du même modèle que la socket précédemment créée.

- Profil d'appel

```
#include <sys/types.h>
#include <sys/socket.h>
int accept
   (int ns,
   struct sockaddr_in *addr_cl,
   int lg_addr_cl)
```

ns : Référence nouvelle socket

addr_cl : L'adresse du client.

lg_addr_cl: La longueur de l'adresse.

Aprofondissement concernant les primitives listen et accept

Exemple de code pour un serveur qui acepte des connexions successives et qui créé un processus pour traiter chaque client.

```
#include <sys/socket.h>

/* Adresse socket du client appelant */
struct sockaddr_in from;

quelen = ... ;
if (listen (s, quelen) <0 )
       Cas d'erreur

/* On accepte des appels successifs */
/* Pour un appel on créé un processus */

if((g=accept(f,&from,sizeof(from)))<0)
                  Cas d'erreur

if ( fork ...
/* Processus traitant de connexion*/
```

connect

- La primitive **connect** (bloquante) permet à un client de demander l'ouverture (**active**) de connexion à un serveur.

- L'adresse du serveur doit être fournie. La partie extrémité locale relative au client est renseignée automatiquement.

- Pour un échange connecté, le connect permet d'utiliser ensuite les primitives read, write, send, recv.

Le client ne doit plus fournir l'adresse du serveur pour chaque appel mais le descriptif de la socket..

- Profil d'appel

```
#include <sys/types.h>
#include <sys/socket.h>
int connect
       (int s,
       struct sockaddr_in *addr_serv,
       int lg_addr_serv)
```

s : La référence de la socket
addr_serv : L'adresse du serveur.
lg_addr_serv : La longueur de l'adresse.

send, recv

- Les primitives **send, recv** (bloquantes) permettent l'échange effectif des données.

- Le profil d'appel est identique à celui des primitives read et write sur fichiers avec un quatrième paramètre pour préciser des options de communications.

- Profil d'appel

```
#include <sys/types.h>
#include <sys/socket.h>

int send
       (int s, char *zone,
       int lg_zone,int options)

int recv
       (int s,char *zone,
       int lg_zone,int options_com)
```

s	: La référence de la socket
zone	: La zone à échanger.
lg_zone	: La longueur de la zone.
options_com	: Les options (données urgentes , ...)

sendto, recvfrom

- Les primitives **sendto, recvfrom** permettent l'échange des données plutôt dans le mode non connecté UDP.

- On doit préciser l'adresse destinataire dans toutes les primitives **sendto** et l'adresse émetteur dans les **recvfrom**.

- Profil d'appel

```
#include <sys/types.h>
#include <sys/socket.h>
int sendto (
       int s,
       char *zone,
       int lg_zone,
       int options_com,
       struct sockaddr_in *addr_dest,
       int lg_addr)
int recvfrom (
       int s,
       char *zone,
       int lg_zone,
       int options_com,
       struct sockaddr_in *addr_emet,
       int *lg_addr)
```

addr_dest : L'adresse du destinataire.
addr_emet : L'adresse de l'émetteur.

lg_addr : La longueur de l'adresse.

Shutdown

- Permet la purge des données en instance sur une socket avant la fermeture.

shutdown(s , h);

h = 0 l’usager ne veut plus recevoir de données
h = 1 l’usager ne veut plus envoyer de données
h = 2 l’usager ne veut plus ni recevoir, ni envoyer.

close

- Permet la fermeture d'une connexion et la destruction du descriptif.

- Profil d'appel

```
#include <sys/types.h>
#include <sys/socket.h>

int close ( int s )
```

Fonctionnement en TCP

- Serveur.

    socket
    bind
    listen
    accept
    recv, send
    close

- Client.

    socket
    connect
    recv, send
    close

Fonctionnement en UDP

    socket
    recvfrom, sendto

SYSTÈMES D'OBJETS

RÉPARTIS

G. Florin

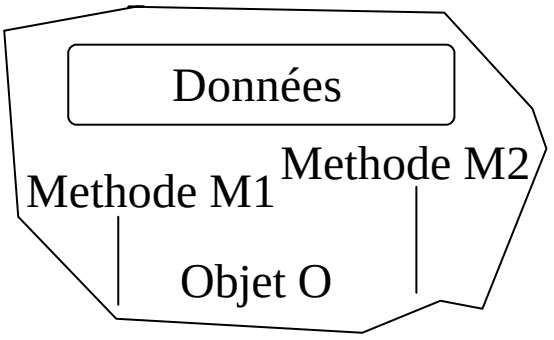
INTRODUCTION
Langages et systèmes d'objets répartis

• Issus de la convergence de deux approches

L'approche objet

Une structuration des applications qui place la notion d'objet au centre de la conception: performante et de plus en plus utilisée

Objet: **associe les traitements et les données** en rendant les codes moins coûteux à développer, à maintenir, à réutiliser.



Abstraction

Comportement communs => Typage.

Encapsulation

Association des actions aux données.

Héritage

Réutilisation du code.

Polymorphisme

Réutilisation des références.

La distribution

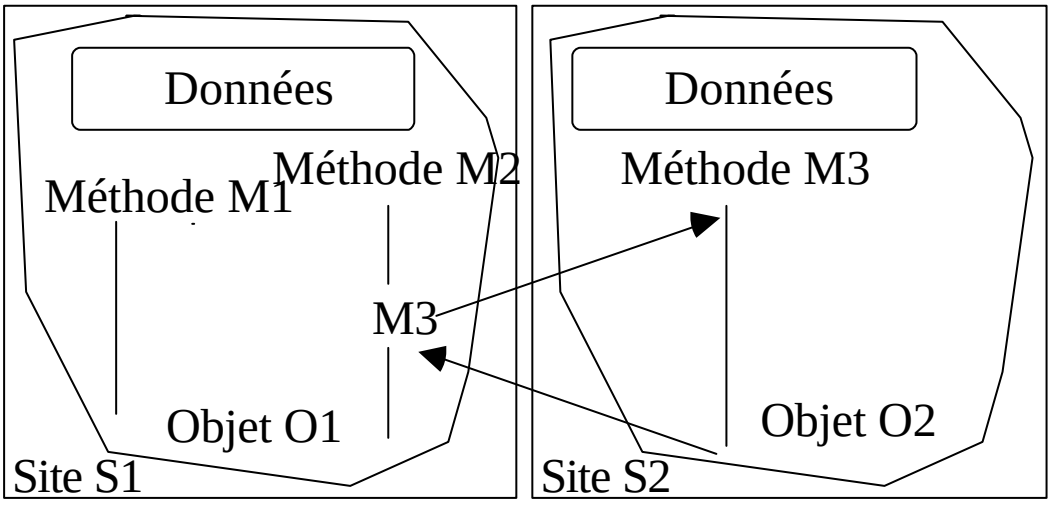
Intégration de la notion d'objet au système réparti

Les communications réseau offrent la possibilité d'implanter les objets de façon homogène sur les sites d'un système réparti.

L'interface du système d'objet réparti est un ensemble de primitives ou un langage objet concurrent et réparti

=> offrant la possibilité **d'implanter et d'exécuter à la demande** des objets sur des sites quelconques

- **Création à distance** d'instances
- **Exécution à distance** des méthodes.
- **Concurrence/synchronisation** .
- **Autres fonctions**: transactionnel, sécurité, optimisation du placement ...



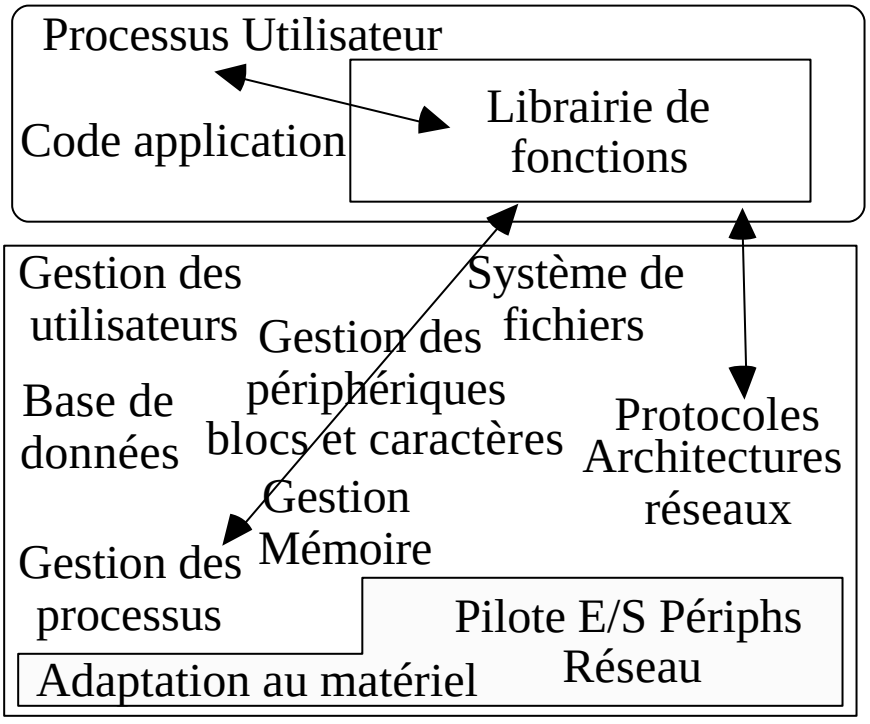
Un enjeu technologique majeur

Situation actuelle: l'hétérogénéité de désignation, de protection, d'interface

Objets manipulés par un utilisateur:

Fichiers, données en mémoires, processus, communications, objets spécifiques usagers ...

Les entités manipulées sont désignées, protégées, traitées de façon différente.

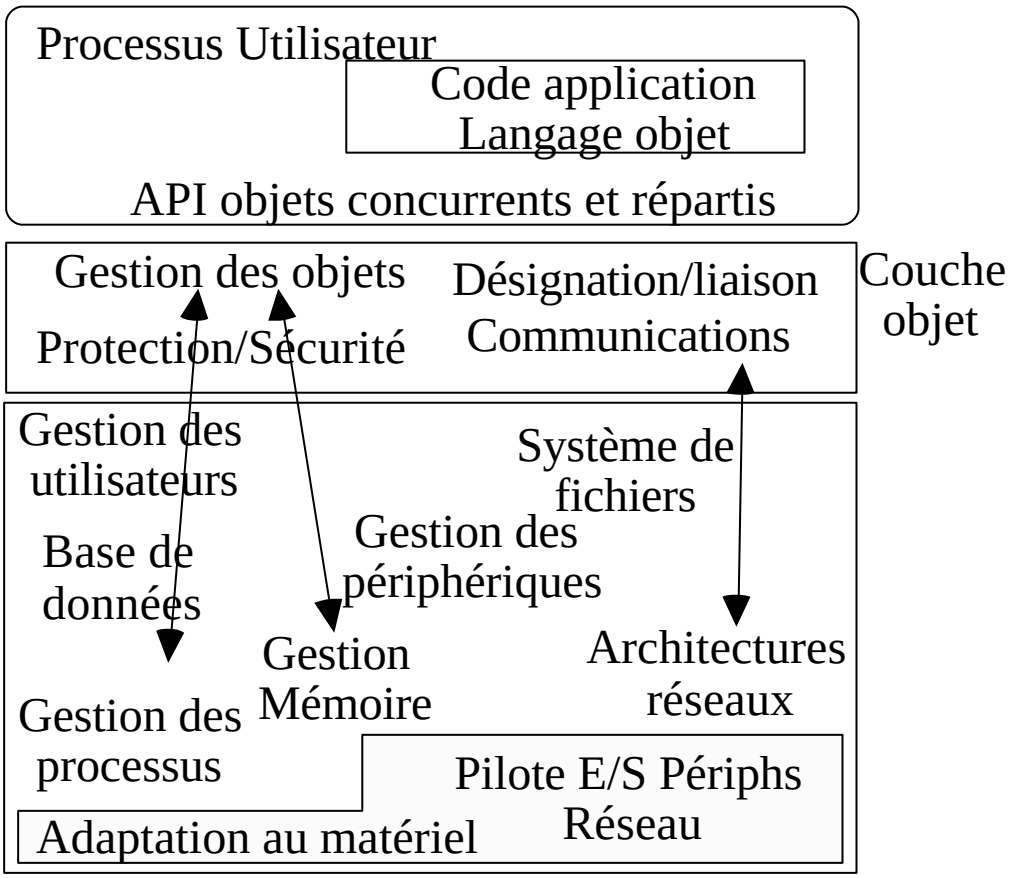


Situation actuelle: complexité de la programmation des applications particulièrement en univers réparti (hétérogène).

Projet : simplifier et uniformiser les accès et les communications offerts par les interfaces systèmes et les codes utilisateurs

Uniformisation des abstractions utilisées sous le concept objet

- Désignation universelle
- Interactions de communication simplifiées
- Protection et sécurité universelle



Réaliser effectivement le projet défini initialement pour les systèmes répartis

Nombreuses difficultés

- Performance des systèmes et applications développées en approche objets répartis.

Les solutions objets imposent le plus souvent un nombre de messages plus élevés et des délais d'interactions plus importants.

- Adaptation des développeurs aux techniques objets répartis.

Les habitudes changent difficilement.

- Difficulté d'une normalisation réelle permettant l'interopérabilité des applications.

Un enjeu pour la normalisation

Standardiser l'offre systèmes d'objets répartis pour intégrer des systèmes d'origine différentes dans une application:

La normalisation minimum habituelle par les protocoles (standardiser les interactions):

=> interopérabilité entre des implantations

- unifier les techniques d'appel distant
- accéder à des annuaires communs d'objets.

Puis standardiser les langages et techniques de développement ...

=> portabilité des applications

OMG - CORBA

"Object Management Group"

"Common Object Request Broker Architecture"

Microsoft OLE/DCOM

"Object Link Embedding"

"Distributed Component Object Model"

Beaucoup de travail reste à faire.

Des produits commerciaux

Un enjeu technologique pour tous les constructeurs et les consortiums.

Les plates-formes CORBA

IONA, **Visigenic**, **Chorus COOL**, **IBM DSOM** "Distributed System Object Model", ...

La plate-forme Microsoft OLE/DCOM/Active X

"Distributed Component Object Model"

Très nombreuses autres annonces (d'importance industrielle variable)

Plan

INTÉGRATION DE LA RÉPARTITION DANS L'APPROCHE OBJET: LE STANDARD CORBA

- Introduction: situation du standard
- La norme CORBA 2.0
- L'interopérabilité en CORBA 2.0

INTÉGRATION DE LA RÉPARTITION DANS L'APPROCHE OBJET: LE STANDARD CORBA

Introduction: situation du standard

L'OMG "Object Management Group"

Consortium d'entreprises, d'organismes
de systèmes/réseaux,
du logiciel,
d'utilisateurs finaux
Intéressés par les technologies objet.
13 membres en 1989 jusqu'à plus de 700 membres.

Objectif principal

Promouvoir la théorie, l'utilisation des objets dans les systèmes répartis (portabilité, réutilisation, interopérabilité dans les systèmes d'objets répartis ouverts).

Moyen principal

Définir des standards pour l'interopérabilité et la portabilité d'applications ‘objet réparti’
Définir les services dont elles ont besoin.

Fonctionnement

Émission de RFI et RFP

"Request For Information"

"Request For Proposals"

Expression d'un cahier des charges pour une fonction, un service par un groupe de travail de l'OMG

Réponses par différents organismes

Adoption d'une (ou plusieurs) propositions pour définir un standard.

Conséquence importante

L'OMG produit des normes (approche normative).

Les logiciels effectifs sont ensuite développés par qui le souhaite:

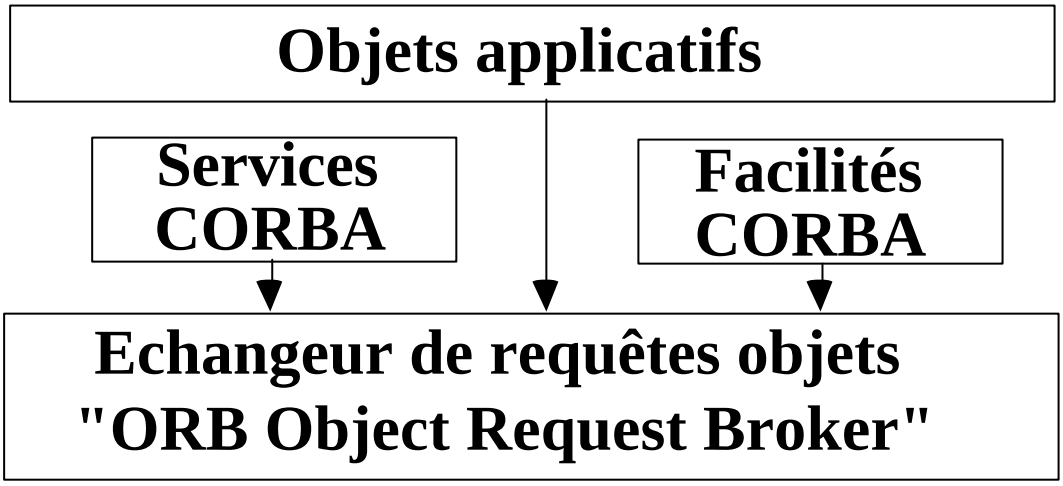
=> Problèmes d'interopérabilité

=> Problèmes de multiplicité des efforts de développement.

L'OMA
"Object Management Architecture"

Un modèle de référence pour une architecture dédiée aux objets répartis.

Environnement cooperatif de composants client-server



CORBA permet à des objets (des composants) d'interagir en univers ouvert

Découvrir la localisation, réaliser les invocations, pour de nombreux langages, environnements d'exécution, réseaux .

La norme définit en plus un ensemble de services

créer, supprimer, nommer, rendre persistants, gérer des transactions etc. entre objets

Les composants orientés systèmes

L'échangeur de requêtes objets "Object Request Broker" ORB

La partie centrale de l'ensemble qui réalise effectivement les interactions entre objets ("**bus à objets**").

Les services systèmes communs CORBA Services "Common Object Services"

Des objets développés en CORBA qui offrent des services systèmes complémentaires à l'ORB (utilisables par invocation ou par héritage).

Exemples: nommage, transactionnel,...

Les composants orientés applications

Les services objets communs
"CORBA facilities"
"Common facilities"

Boite à outils de services d'usage courant nécessaires aux objets d'applications.

Exemples: services d'impressions de documents composites, stockage de documents composites, messageries , ...

Les objets applicatifs
"Objets métiers"
"Applications objects"

Objets conçus pour des besoins particuliers.

Les objets applicatifs collaborent par échange de requêtes pour créer des applications "coopératives" client-serveur (applications réparties, multi-tiers, workflow ...)

L'échangeur de requêtes ORB "Object Request Broker"

Acheminement des requêtes et des réponses

L'ORB décharge une application objet de tout un ensemble de tâches pour l'exécution d'appels de méthodes à distance sur des objets en univers hétérogène:

- localisation des serveurs.
- transport des requêtes et des réponses.
- alignement, présentation des paramètres.
- activation à distance.

Réalisation d'invocations statiques et dynamiques à distance

Invocation statique: les objets appelants et appelés sont connus à la compilation (possibilité de vérification du typage fort).

Invocation dynamique: l'appelant et l'appelé peuvent être créés en exécution et néanmoins peuvent collaborer (possibilité de réaliser une liaison tardive).

Support d'une grande variété de langages

L'ORB permet à des applications en C, C++, JAVA, ADA, Cobol, Smalltalk, ... de coopérer en univers réparti. Les outils essentiels sont:

- Le langage de description d'interfaces IDL Corba ("Interface Definition Langage").

```
interface Personne              {
  readonly attribute unsigned short age;
  attribute string nom;     };
```

- La définition des correspondances ("mappings") entre les types des différents langages et les types IDL.

Exemple: IDL	Java
unsigned short	unsigned short
string	java.lang.string

- La définition d'un format commun de représentation des données CDR.

("**C**ommon **D**ata **R**epresentation")

Support d'une représentation des interfaces

L'ORB gère un référentiel d'interface: une base de données des définitions d'interfaces ("Interface Repository").

ORB et RPC

L'outil central de l'ORB est un protocole de RPC ("Remote Procedure Call").

L'ORB ajoute l'intégration du RPC à un univers d'objets répartis ouvert.

. Interconnexion d'univers objets répartis de types différents développés dans des langages différents.

. Détermination de la localisation d'un objet distant qui implante la méthode. Une requête d'exécution à distance comporte:

- Une référence de l'objet distant:
  On s'adresse à un objet particulier.
- Un sélecteur de la méthode appelée
  Les messages peuvent donc être "polymorphes" : pour un même message l'opération réalisée dépend de l'objet invoqué.
- Un contexte (optionnel)

Précisant des paramètres nécessaires au contexte de l'exécution (utilisateur pour la protection).

Les services systèmes communs "CORBA Services" "Common object services"

Toujours en cours de développements et d'amélioration.

Le service du cycle de vie "Life cycle service"

Création, déplacement, destruction d'objets (de composants) dans un système d'objets.

Le service de persistance "Persistance service"

Sauvegarde et rechargement d'objets sur des mémoires persistantes (bases de données).

Le service de nommage "Naming service

Politique de désignation et localisation des objets sur les sites d'un système d'objets CORBA.

Les services systèmes communs (suite)

Le service d'événements "Event service"

Service de génération et de distribution d'événements (asynchrones) à des objets qui s'abonnent ou se désabonnent.

Le service de contrôle de concurrence "Concurrency control service"

Gestionnaire de verrous pour réaliser un contrôle de concurrence entre activités ou transactions.

Le service transactionnel "Transaction service"

Service de validation à deux phases pour des unités transactionnelles.

Le service de relation "Relationship service"

Permet de créer des relations entre objets et d'utiliser les liens.

Les services systèmes communs (suite)

Le service d'externalisation "Externalization service"

Permet de transformer un objet en un format standard de type flot pour le sauvegarder ou le transmettre.

Le service d'interrogation "Query service"

Langage d'interrogation de données objets dérivé de SQL et OQL "Object Query Language"

Le service de licences "Licensing service"

Définition de fonctions de contrôle de l'usage d'objets (par création, par site, par session).

Le service de propriétés "Properties service"

Association de valeurs à des objets pour leur associer des propriétés (dates, attribut).

Les services systèmes communs (suite)

Le service de sécurité
"Security service"

Authentification d'accès, confidentialité, non répudiation, protection d'accès par listes de contrôle d'accès, journalisation.

Le service de temps
"Time service"

Obtenir l'heure d'un site distant, calculer un intervalle entre deux événements.

Le service de courtier"
"Trader service"

Service de pages jaunes d'annuaire.

Les utilitaires communs "CORBA Facilities" "Common facilities"

Ensemble d'outils définis sous forme d'objets.

Les utilitaires communs "verticaux"

Définis par métiers (segments de marchés):
domaines: électronique, santé, assurance, ...

Les utilitaires communs "horizontaux"

Définis par fonctions

1 Services d'interface utilisateur

2 Services d'archivage et de transfert de documents composites

Les services définis pour la manipulation des objets intégrés dans des documents composites: affichage, couper/coller, impression :**Opendoc**.

Services de stockage et de transmission.

3 Services d'administration système

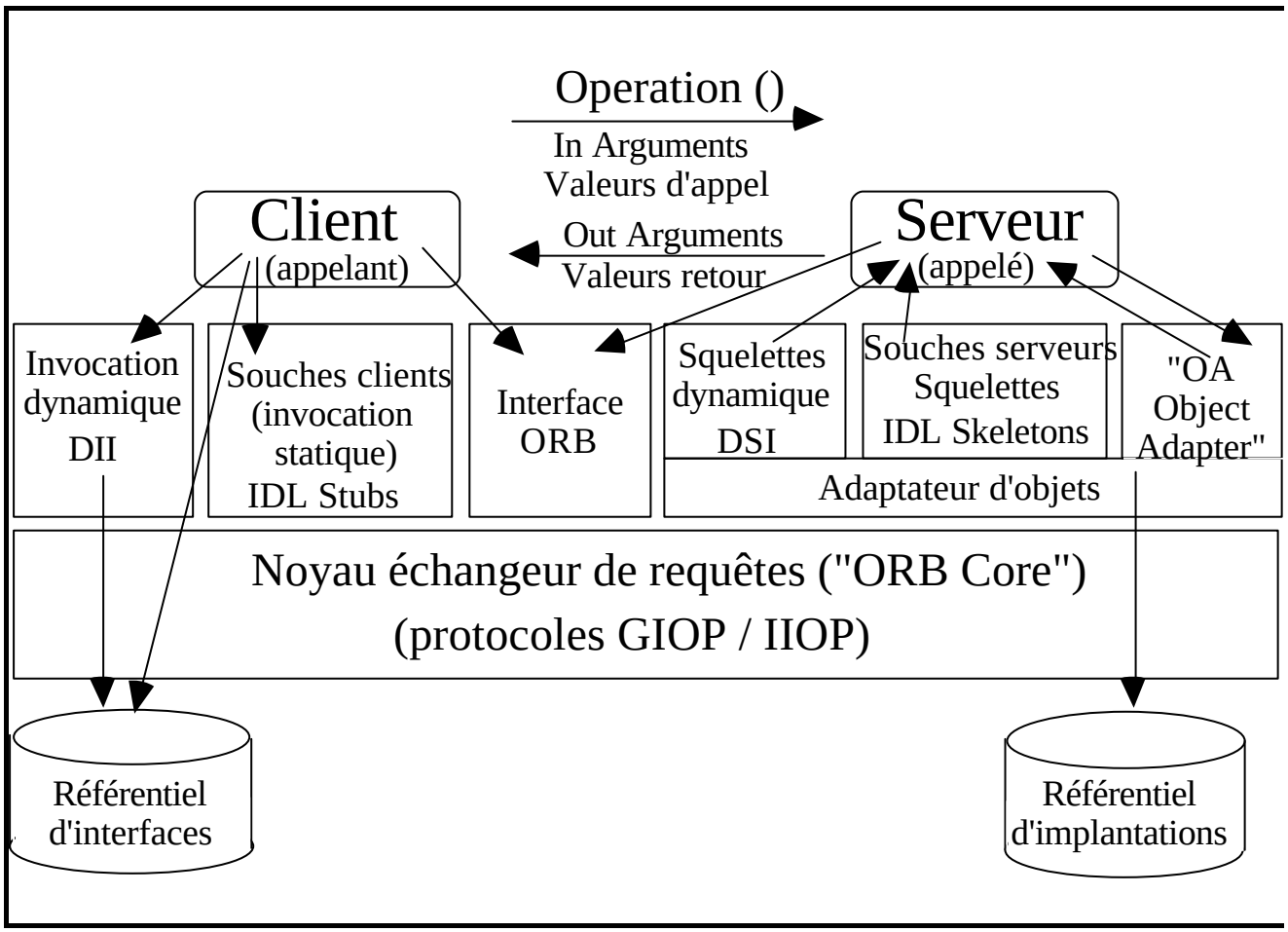
Fonctions habituelles d'administration: détection des pannes, installation, performances.

4 Services d'administration de tâches

Workflow, courrier électronique, ...

Les principaux éléments de la norme CORBA 2.0

La structure de CORBA 2.0



L'interface d'invocation statique
Les souches IDL coté client
"IDL stubs"

Lorsque le client connaît à la compilation toutes les procédures qu'il va utiliser il peut définir les interfaces d'appel et générer des souches clients. Idéalement l'appel distant peut ensuite être réalisé comme s'il s'agissait d'un appel

objet -> opération (arguments)

Rappel: utilisation de souches

Les souches clients sont des procédures d'interface statique pour accéder aux implantations distantes.

Ce sont les souches clients qui effectuent l'empaquetage puis le dépaquetage (marshalling unmarshalling) des paramètres de la méthode invoquée.

Langage l'IDL CORBA ("Interface Definition Language").

Les souches clients sont définies dans l'IDL CORBA puis traduites dans le langage du client par un compilateur.

Exemple de déclaration d'interface en langage IDL

```
module Voiture
{
    /* Définition d'une classe cabriolet */
    interface Cabriolet :
véhicule_thermique ;
    {
    attribute integer consommation ;
    exception Panne (string explication) ;
    void accélère ( in short durée)
        raises (Panne) ;
    void freine ( in short durée)
        raises (Panne) ;
    ....
    }
    interface Monospace :
véhicule_thermique;
    {
    attribute integer place ;
    void accélère ( in short durée)
    ....
    }
} /* Fin de module */
```

L'interface d'invocation dynamique "DII Dynamic Invocation Interface"

Une interface client-serveur générique capable d'envoyer tout type de requête à tout instant vers n'importe quel objet.

Permet en exécution la découverte d'un objet (non connu à la compilation)

- rechercher un objet,
- obtenir sa référence,
- générer un appel (paramètres),
- émettre l'appel
- récupérer les résultats.

Utilisation

En cas de liaison tardive. Exemple: création dynamique d'interface graphique (par clics) et utilisation à distance immédiate

Évaluation de performances

L'ensemble des opérations à réaliser rendent l'invocation dynamique assez coûteuse (plusieurs appels à l'API DII sont nécessaires impliquant d'éventuels échanges de messages). Dans la plupart des cas l'invocation statique est suffisante.

Quelques éléments de fonctionnement de l'interface d'invocation dynamique (1)

Liste des opérations à effectuer

Obtenir la référence de l'objet/méthode,
Créer une liste de paramètres,
Générer le premier paramètre,
...
Générer le dernier paramètre,
Envoyer l'invocation,
Récupérer les résultats.

Quelques requêtes

A) Connaître la méthode appelée get_interface, describe_interface

Pour obtenir la spécification de l'interface de l'objet. Pour obtenir des informations sur les opérations supportées par un objet du référentiel d'interface.

B) Créer une liste des arguments d'appel

create_list Créer la structure de donnée liste d'arguments.

add_arg Pour ajouter un à un chaque argument.

Quelques éléments de fonctionnement de l'interface d'invocation dynamique (2)

C) Créer une requête d'appel distant

create_request

Pour créer un objet requête pour l'opération considérée.
- référence de l'objet distant
- sélecteur de méthode
- liste des arguments

D) Effectuer l'invocation

invoke()

Pour effectuer la requête en RPC.

Le référentiel d'interface
IR "Interface Repository"

Un élément essentiel de l'ORB c'est la base de données d'interfaces des objets enregistrés.

Le référentiel d'interface est consulté pour connaître la description des opérations qu'une interface supporte et les liens d'héritage entre interfaces.

Usager : Découverte de composants réutilisables (en cas de liaison dynamique)

ORB : Vérification de conformité d'appel.

Des objets de type InterfaceDef (descripteurs d'interfaces IDL) sont stockés dans une librairie.

Quelques éléments de l'API

Object::get_interface

Retourne une référence sur un objet InterfaceDef qui décrit une interface d'objet.

InterfaceDef::describe_interface

Pour obtenir des informations sur les opérations supportées par un objet.

L'interface de l'ORB
"ORB Interface"

Différents styles d'implantations possibles des ORB:

- librairie de fonctions
- un ou plusieurs processus

CORBA spécifie l'interface de l'ORB de manière à être indépendante des détails d'implantations.

Exemple de types de fonctions:

Initialisation de l'ORB

Conversion des références d'objets en chaînes et vice versa.

Référentiel d'implantations "Implementation repository"

La base de données des classes (des implantations d'objets) supportées, des objets instanciés et de leurs références objets.

Autres informations gérées:
suivi, audit, sécurité, administration.

Squelette statique "IDL Skeleton"

En langage CORBA, c'est la procédure souche coté serveur lors d'une liaison statique.

Interface de liaison dynamique "DSI Dynamic Skeleton Interface"

C'est la partie souche coté serveur lors d'une invocation dynamique.

Il réalise la liaison avec l'objet et la méthode cible.

Les adaptateurs d'objets "OA Object Adapter"

Fonctions principales

Assister l'ORB dans la délivrance des requêtes d'accès aux objets coté serveur.

Permettre l'exécution d'une méthode d'un objet (constituer l'environnement) : instancier, assigner des références objets.

Enregistrer les objets dans le référentiel d'implantation.

Éviter la multiplication des variantes d'adaptateurs d'objets.

L'adaptateur de base BOA ("Basic Object Adapter")

La norme CORBA spécifie une version minimale d'adaptateur (le BOA).

a. Il gère le référentiel d'implantations (enregistrer et accéder aux implantations).

b. Il génère et interprète les références.

c. Il active et désactive les implantations.

d. Il reçoit les requêtes aux méthodes et les transmet via la souche serveur (‘skeleton’).

e. Il authentifie les clients.

Adaptateurs de base d'objets et gestion du parallélisme

1. Serveur partagé (BOA "Shared Server")

Un seul processus pour tous les objets.
Les requêtes sont traitées en séquence.

2. Serveur non partagé (BOA "Unshared Server")

Un processus créé par objet.
Possibilité de parallélisme entre objets.

3. Serveur par méthode (BOA "Server-Per-Method")

Un processus par requête.
Possibilité de parallélisme entre méthodes

4. Serveur persistant (BOA "Persistent Server")

Plusieurs processus sont créés au départ.
L'OA fonctionne en serveur partagé pour chaque processus.
Possibilité d'un niveau de parallélisme égal au nombre de processus serveurs.

Différents autres adaptateurs d'objets

Des adaptateurs d'objets spécialisés supportent certains styles d'implantation des objets.

Bases de données orientées objets

Utilisable pour la gestion des objets à grain fin et pour la persistance.

Disposent d'une interface spécifique de chaque base orienté objet.

Adaptateurs d'objets pour objets locaux

Pour éviter d'utiliser des techniques d'invocations distantes coûteuses pour des objets locaux.

Conclusion Objets Répartis
- CORBA -

Un outil très important dans les applications client serveur

Approche de plus en plus répandue de la programmation orientée objet: notion de programmation par "composants"

Idée de programmation indépendante des composants et de réutilisation.

Le système d'objets répartis est la clé de voûte de cette approche.

La normalisation des systèmes d'objets répartis: CORBA
Nombreuses implantations

Une bataille en cours entre l'approche CORBA et l'approche Microsoft

Bibliographie

Georges Gardarin, Olivier Gardarin "Le client-serveur" Eyrolles

Robert Orfali, Dan Harkey, Jeri Edwards, "The essential distributed object survival guide" Wiley.

Nombreuses pages WEB
http://www.omg.org/

Chapitre

La conversion

des données

Introduction
à la conversion des données

Tout système informatique (processeur + système d'exploitation + langage + ...) effectue de **nombreux choix relativement au codage des informations gérées**.

- Les choix possibles sont très **variés**.
- Ils concernent **tous les types** de données.
- Ils conduisent à des **différences majeures** concernant des notions qui ont en fait le même objectif (caractère, entier, ...)

Grave hétérogénéité syntaxique et **sémantique**
(d'un constructeur à un autre, d'un modèle à l'autre du même constructeur)

Très pénalisante pour des systèmes en réseau de machines hétérogènes
puisque les données circulent entre machines (systèmes "ouverts").

La présentation des données

- Fonction **essentielle de la répartition (**en environnement ouvert).

- Elle est à effectuer à **chaque transfert**
=> Performances élevées indispensables

Objectif assigné à la **couche présentation** du modèle **OSI**
Normes correspondantes.

Définition de syntaxe abstraite et de syntaxe de transfert ASN1
Service et protocole de présentation ISO 8822 et 8823

Objectif assigné à de nombreuses implantations propriétaires de logiciels de conversion non normalisés.

. Fournisseurs de logiciels réseaux.
. Fournisseurs d'architectures client-serveur.
. Fournisseurs de systèmes d'objets répartis.
. Fournisseurs de bases de données réparties

Problèmes liés aux choix matériels (représentation interne des données)

Codes caractères

Premier problème de conversion: choix de la représentation des caractères dans le cadre de l'interfonctionnement en mode texte.

=> Code EBDIC (IBM) et normes ASCII (Ex IA5 International ASCII n°5).

Position des octets ou des bits dans un mot mémoire

Deux visions pour la position des octets.

Machines "grand boutiste" (Sun Sparc) ("Big endian")

Les octets sont numérotés de gauche à droite

[0 1 2 3]

Machines "petit boutiste " (Pentium) ("Little endian")

Les octets sont numérotés de droite à gauche

[3 2 1 0]

Échanges en série de données contenues en mémoire (ex: une adresse IP sur 32 bits)
La sérialisation d'un mot mémoire donne selon les cas des résultats différents.

Représentation des types machine de base

Pour tous les types de données de base gérés par le matériel nombreuses variantes:
Représentation des entiers,

Longueur variable (8, 16, 32, 64 bits)
Codage (complément à 2 ou à 1)

Représentation des flottants,

Longueur variable et placement des zones mantisse et exposant.
Codages variables des mantisses et exposant (en base 2 ,ou 16)
Codage du bit signe.

Représentation des types complexes

. Pour la transmission des données définies à partir de types complexes (agrégats, ou types construits "articles").

Représentation des pointeurs

. Problème des adresses utilisées par le matériel: des **pointeurs** peuvent figurer dans les structures de données échangées.

Problèmes liés aux langages (représentation abstraite des données)

- Existence de très nombreux langages (C, C++, Cobol, Ada, Java) définis indépendamment des réseaux.
- Chaque langage effectue des choix concernant les types de données primitives et leur attribue un nom (**syntaxe abstraite**).
- Chaque langage effectue des choix relativement aux **objets** (sémantique des langages orientés objets très variées).
- Chaque langage effectue des choix concernant les **modes de transmission** des arguments dans les appels procéduraux.
- Chaque compilateur effectue une correspondance entre les **types** de données du **langage** et les **types du matériel**.

Faire en sorte que les différents choix effectués par les langages concernant les **types** et la **transmission** des arguments deviennent compatibles avec les réseaux

L'hétérogénéité: un problème habituel pour les programmes

Un programme écrit pour un ordinateur donné (en langage machine) ne peut pas être exécuté sur un autre ordinateur

La solution "compilée"

Écrire l'application dans **un langage de haut niveau**.
Convertir dans le langage d'une machine à l'aide **d'un compilateur** adapté à la machine. Compiler strictement le même langage sur toutes les machines.

La solution "interprétée"

Écrire l'application dans **un langage de haut niveau** (exemple pascal, java) puis compiler dans **un langage intermédiaire** (pcode, bytecode) d'une machine virtuelle.

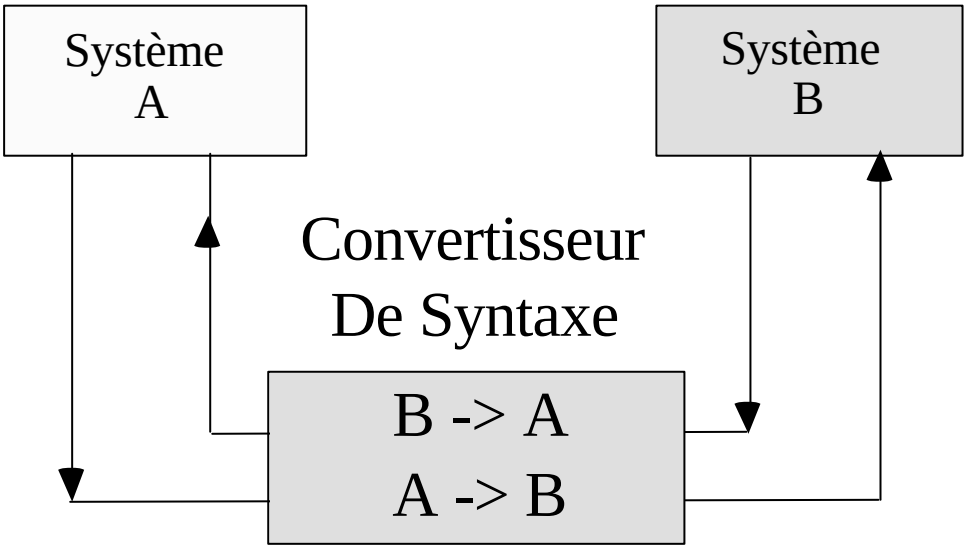
Utiliser sur chaque **machine réelle** un **interpréteur** du langage intermédiaire dans le langage particulier d'une machine cible.

L'hétérogénéité des données dans les réseaux

Résoudre les problèmes de représentation pour faire communiquer des systèmes hétérogènes

Solution directe "transcodage"

Écrire pour un couple de machines **deux traducteurs** des représentations de l'une à l'autre (approche analogue de la compilation).



Conversion directe de syntaxe

Approche impraticable s'il faut faire communiquer sans restrictions un grand nombre d'ordinateurs incompatibles:
Pour N systèmes: N*N-1 convertisseurs.

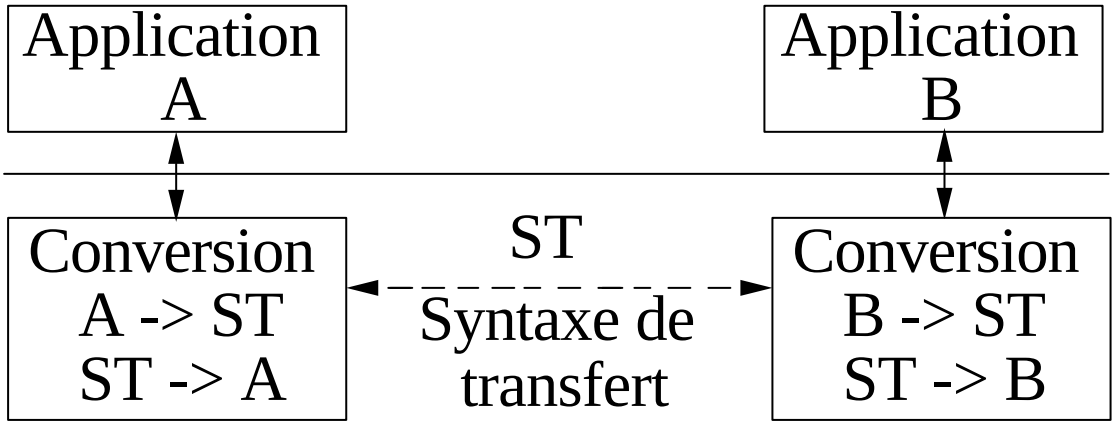
Solution d'un langage intermédiaire "pivot" commun à tout le réseau

Écrire pour chaque machine un traducteur (compilateur) de la représentation de cette machine vers une représentation intermédiaire commune au réseau (schéma analogue de l'interprétation).

Notion de "**syntaxe de transfert**".
(la façon dont les données sont codées dans les messages)

Chaque machine doit disposer d'un traducteur de la représentation réseau (syntaxe de transfert) vers la représentation interne de la machine.

Pour N machines le nombre de convertisseurs est réduit à 2*N



Syntaxe abstraite commune au réseau

Chaque interface de service doit offrir une définition **précise et lisible** des données nécessaires à son invocation.

Une syntaxe de transfert (plutôt de niveau langage machine) est **très peu pratique** pour définir et comprendre des structures de données => C'est le rôle d'une **syntaxe abstraite**

Chaque langage évolué définit sa propre syntaxe abstraite de définition des données.

=> Les langages sont plus ou moins **bons**

=> Les concepts et les langages **évoluent**.

=> Aucun langage ne **s'impose**.

=> Langages évolués définis en dehors des problèmes de la répartition : pas **tous les ingrédients nécessaires**.

Nécessité de définir une syntaxe abstraite commune au réseau différente des syntaxes abstraites des langages existants

Nombreuses propositions.

Message (ASN "Abstract Syntax Notation")

Appel distant (IDL "Interface Definition Language")

Fonctionnement d'une syntaxe abstraite

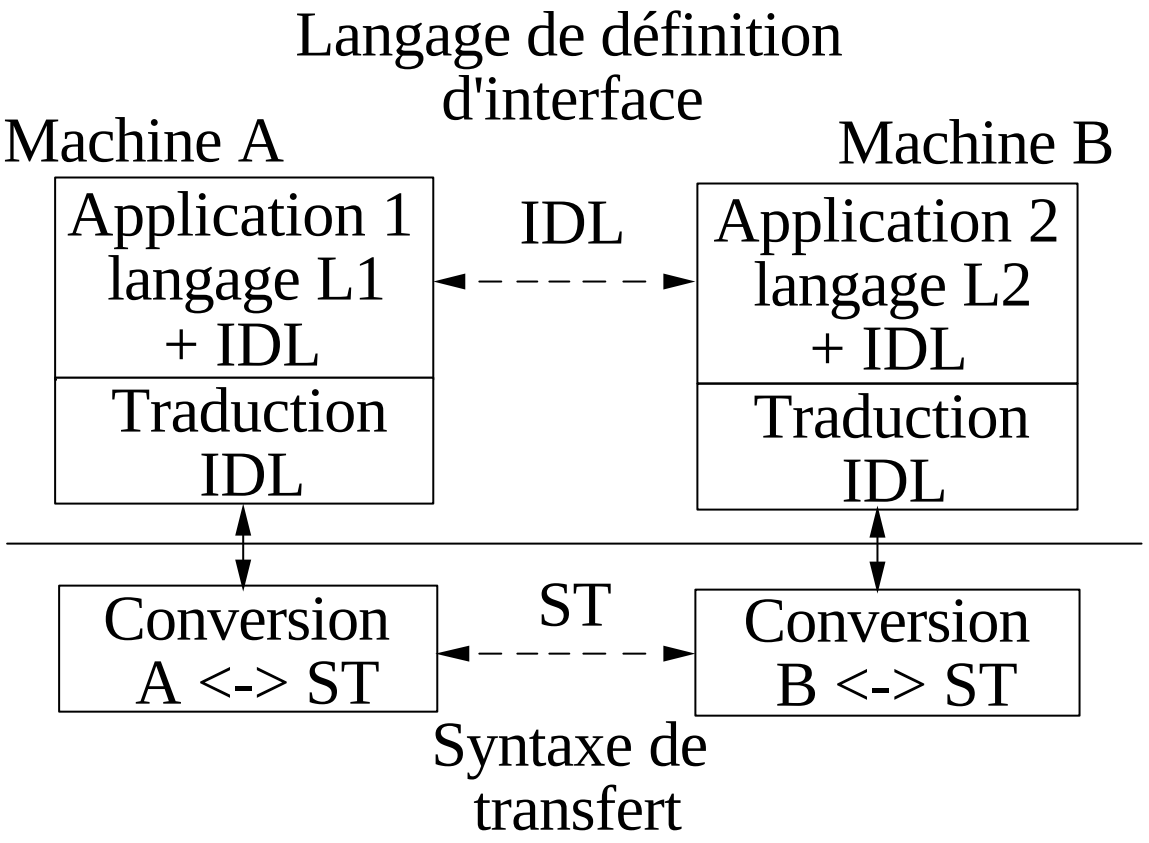
Le langage d'interface permet de définir de façon **universelle** les **services** d'un logiciel réseau et les **données** nécessaires.

Exemples: IDL DCE, DCOM, CORBA

Un programmeur utilisant un langage de développement (C++, Java) de son choix: il existe une **correspondance** (**"mapping"**) entre les données (objets) spécifiés dans l'IDL et les données (objets) du langage.

La traduction des directives IDL insérées dans le langage évolué est opérée par un **traducteur assez simple** (compilateur de syntaxe abstraite).

Schéma de fonctionnement



Conversion des données

L'approche ASN1

Introduction ASN1

Représentation normalisée des données échangées dans le cadre du modèle des systèmes ouverts OSI.

=> Définition d'une syntaxe abstraite et d'une syntaxe de transfert dédiée à l'acheminement des données par message.

Le codage et le décodage des structures de données selon la norme ASN1 est le principal objet du niveau présentation du modèle OSI.

Autres utilisations (hors modèle OSI)

Administration de réseaux **SNMP** ("Simple Network Management Protocol")

Commerce électronique SET ("Secure Electronic Transactions")

Plan du cours ASN1

Introduction

1. Syntaxe abstraite

2. Syntaxe de transfert

Conclusion

1 Notation de syntaxe abstraite ASN.1

Introduction
Syntaxe abstraite normalisée ASN1

Normalisation ITU-CCITT
CCITT X409 Norme de messagerie

Normalisation ISO dans le cadre:
Protocole de présentation avec connexion
ISO 8823
Protocole de présentation sans connexion
ISO 9576

Un pro

(**APDU** : "Application Protocol Data Unit")

- Chaque APDU = **Un type d'opération particulière** à faire réaliser à distance.
- Un APDU **comprend des données** dont le nombre et le type peuvent varier pour chaque utilisation particulière de l'application et qui nécessitent une conversion.

Pour cela il utilise les services du protocole de présentation

Exemple
Messagerie industrielle (MMS):

Objectif MMS
("Manufacturing Messaging Service"):

Lire à distance, **affecter à distance** des variables d'état d'une machine, ou **changer à distance** le programme de fabrication d'une machine outil.

. Les entités d'applications **doivent donner strictement** le même sens aux APDU et aux données contenues (suite de bits que le destinataire doit reconnaître).

Exemple d'utilisation:

Élément de protocole destinée à fixer deux paramètres d'une machine outil

`VITESBROCHE` : Un entier sur 8 bits

`LUBRIF` : Un booléen

Le destinataire doit être capable:

- de reconnaître le type d'opération (dans l'entête de l'APDU)
- d'interpréter les champs (les deux paramètres)

Solution: définir les données échangées dans un langage de haut niveau

- Une syntaxe abstraite décrit des données dans une **notation formelle** non ambiguë.

- Approche nécessairement **fortement typée** (chaque objet à un type).

- La description ASN.1 d'un type de données est appelée sa ***syntaxe abstraite*** (parce qu'aucune représentation particulière n'est imposée à ce niveau).

- A un type sont associées des **opérations** qui le caractérisent (lire, écrire).

- Premiers exemples de types de base

Types prédéfinis

"integer", "boolean" , ...

Types structurés

"sequence" Liste ordonnée de types,

"choice" Un type quelconque pris dans une liste,

"set" Collection non ordonnée de types variés.

Exemple1: Activation d'une machine outil

```
Demarre_MO ::= SEQUENCE
{  vitesbroche    INTEGER,
   lubrif         BOOLEAN      }
```

Exemple2: Descriptif au fichier du muséum des dinosaures

```
Dinosaure ::= SEQUENCE
{  nom            OCTET STRING,
   longueur   INTEGER,
   carnivore      BOOLEAN,
   os             INTEGER,
   decouverte     INTEGER       }
```

Exemple3: Description partielle d'un journal d'opérations messagerie industrielle

```
CreatejournalrequestPDU::=
              confirmes-RequestPDU[0]
IMPLICIT SEQUENCE
{  invokeID Unsigned32,
   listOfModifier SEQUENCE OF
                  Modifier OPTIONAL,
   createjournal [69] IMPLICIT SEQUENCE
   {journalName[0] ObjectName} }
Unsigned32 ::= INTEGER
ObjectName ::= CHOICE
{  vmdspecific   [0] IMPLICIT Identifier
   domainspecific[1] IMPLICIT SEQUENCE
   {  domainID  Identifier,
      itemID Identifier},
   aaspecific[2] IMPLICIT Identifier}
```

Les types primitifs ASN.1 (types de base)

Ces types servent de **briques de base** pour élaborer des types complexes.
Les mots réservés ASN.1 sont écrits en **majuscules**.
Les noms des types de base sont des mots **réservés**.

Tableau des types primitifs

Type primitif	Signification
INTEGER	Entier de longueur quelconque
BOOLEAN	Vrai ou faux
BIT STRING	Liste de 0 ou plusieurs bits
OCTET STRING	Liste de 0 ou plusieurs octets
NULL	Aucun type
ANY	Union de tous les types
OBJECT IDENTIFIER	Nom d'objet

Détail des principaux types primitifs

- **Integer** : entiers de taille arbitrairement grande, également utilisable pour les types énumérés par des définitions complémentaires (dimanche = 1, lundi = 2,)
- **Boolean** : vrai ou faux
- **Bit string** : décrit des chaînes binaires
  en binaire '010011101'B
  ou en héxadécimal '4D'H
- **Octet string** : définit des listes ordonnées d'octets.
- **Any** : permet de spécifier n'importe quel type (pour remplissage ultérieur de la zone par n'importe quel type).
- **Null** : définit un objet sans type (vide) qui n'a pas besoin d'être transmis.
- **Object identifier** : définit un moyen d'identifier au moyen de chaînes de symboles un objet dans un protocole:
Exemple : {iso standard 8571 part 4 ftam-pci(1)} référence un objet définit dans la partie 4 de la norme 8571 FTAM.

Les types constructeurs ASN.1

- **Sequence** : permet de construire des types combinaison quelconque (de façon analogue au "record" pascal, "struct" C) .

- **Sequence of** : permet de construire des tableaux d'un seul type.

- **Set** : permet de construire des ensembles (collections non ordonnées) d'objets de types quelconques

- **Set of** : permet de construire des ensembles d'objets du même type.

- **Choice** : définit une structure de donnée qui doit comprendre des types appartenant à un ensemble de types différents.

Type constructeur	Signification
SEQUENCE	Liste ordonnée de types divers
SEQUENCE OF	Liste ordonnée d'un seul type
SET	Collection non ordonnée de divers types
SET OF	Collection non ordonnée d'un seul type
CHOICE	Un type quelconque pris dans une liste

Exemple de l'emploi de "choice"

- Définition d'une application de transfert de travaux à distance (un travail comporte trois parties).

- La partie principale est constituée de commandes soit locales soit à distance

=> emploi de "choice".

- Les définitions manquantes sont supposées réalisées dans une bibliothèque annexe nommée joblib.

```
Job ::= SEQUENCE
{
   header              Account-PDU,
   body SEQUENCE OF Command-PDU,
   trailer             Termin-PDU
}

Command-PDU ::= CHOICE
{
   Local-command,
   Remote-command
}
Account-PDU   ::= Joblib.account
Local-command ::= Joblib.local
Remote-command::= Joblib.remote
Termin-PDU ::= Joblib.termination
```

Remarques complémentaires

En plus des types primitifs et des types construits, ASN.1 spécifie également des types **prédéfinis** couramment utilisés.

- Huit types de **chaînes** différents ont été définis qui sont des sous-ensemble différent de `OCTET STRING`.
- `NumericString` comporte les chiffres (de 0 à 9) et le caractère espace.
- `PrintableString` comprend les lettres minuscules, les lettres majuscules, les dix chiffres, le caractère espace ainsi que les 11 caractères suivants :
( ) ' + - . , / : = ?

- `GeneralizedTime` permet d'éviter toute ambiguïté sur une date pour déterminer si 5/12 représente le 5 décembre ou le 12 mai.

Exemple: la valeur `19980427210538.8` pour le type `GeneralizedTime` indique 21 heures 05 minutes 38.8 secondes, le 27 avril 1998.

Notion de module

On peut regrouper des définitions apparentées de type, de valeurs, de sous-types dans un **module.**

Un module est identifié par son nom.

La première lettre d'un nom de module commence toujours par une majuscule.

Exemple de module:

nom du module puis DEFINITIONS

```
Couleur DEFINITIONS ::=
    BEGIN
        CouleurPrimaire::= ENUMERATED
            {rouge(0),jaune(1),blanc(2)}
        couleurParDefaut
                CouleurPrimaire ::= jaune
    END
```

Notion de macro

La notation de syntaxe abstraite ASN.1 offre la possibilité de définir des types ou des valeurs au moyen de **macros définition.**

Exemple de macro définition

- X,Y abscisse et ordonnée dans un système de coordonnées rectangulaires. - On veut traiter le couple X, Y comme un type auquel on donne le nom de **complexe.**

- X et Y peuvent être de types différents et redéfinissables :X entier, Y réel (ou autres).

- On utilise une notation spéciale de façon à pouvoir préciser un type COMPLEXE particulier en invoquant:

```
COMPLEXE
    TYPEX=.un premier type
    TYPEY=.un second type
```

- Avec cette notation on peut définir des types COMPLEXE1, COMPLEXE2 etc

```
COMPLEXE1::= COMPLEXE
    TYPEX = INTEGER
    TYPEY = INTEGER
COMPLEXE2::= COMPLEXE
    TYPEX = REAL
    TYPEY = REAL
```

Macro définition de valeurs

- On veut que la notation de valeur pour une variable de type complexe soit de la forme: `(X=...;Y=....)`

- Avec par exemple pour une réalisation particulière de la variable de type COMPLEXE1: `( X=56;Y=3561)`

- Pour arriver à définir ceci on utilise une macro de la forme :

```
COMPLEXE
MACRO ::= BEGIN
    TYPE NOTATION::=
        "TYPEX"
        "="
        type (TypAbscisse)
        "TYPEY"
        "="
        type (TypOrdonnee)
    VALUE NOTATION::=
        "("
        "X"
        "="
        value (Abscisse TypAbscisse)
        ;
        "Y"
        "="
        value (Ordonnee TypOrdonnee)
        ")"
END
```

Attributs optionnel et défaut

"OPTIONAL"

En pratique, il est très utile de pouvoir définir des types de données complexes dont certains champs sont optionnels (renseignés ou pas).

En ASN.1 mot clé **OPTIONAL.**

"DEFAULT"

Autre possibilité: déclarer ces champs **DEFAULT**, suivis de la valeur par défaut qui devrait être utilisée par le récepteur s'ils ne sont pas transmis.

Problème

L'existence de types `OPTIONAL` et `DEFAULT` pose des problèmes pour identifier des données lorsque ces dernières sont reçues.

Supposons qu'une `SEQUENCE` possède 10 champs de même type et tous **optionnels**.

Seuls trois d'entre d'eux sont transmis.

Comment le récepteur détermine-t-il de quels champs il s'agit?

Notion d'étiquette ("tag")

- Le but de l'étiquetage est l'identification unique des champs pour lever les ambiguïtés.
- Quatre types d'étiquettes sont prévues

UNIVERSAL,

APPLICATION,

PRIVATE,

Étiquette spécifique au contexte.

- Notation des étiquettes entre crochets.

Exemple : `[APPLICATION 4]`.

Chaque étiquette est définie par un numéro unique (entier) qui identifie l'objet

Il est précédé de l'un des mots réservés `UNIVERSAL`, `APPLICATION` ou `PRIVATE`, ou d'aucun mot réservé, auquel cas l'étiquette est spécifique au contexte.

- Relativement à la syntaxe de transfert.

Chaque fois qu'un item est transmis son **type**, sa **longueur** et sa **valeur** sont transmis.

Lorsqu'un type ou un champ est étiqueté, **l'étiquette** est également transmise.

Redondance des informations de type et d'étiquette

- Si le récepteur peut dire de quel item il s'agit en décodant son étiquette, il est clair qu'il n'est plus nécessaire d'envoyer son type.

- ASN.1 offre la possibilité de supprimer le codage du type pour des champs étiquetés.

- La suppression de l'émission de l'information de type se fait par le mot réservé `IMPLICIT` après l'étiquette.

Exemple:`[PRIVATE 7] IMPLICIT`.

- Si `IMPLICIT` n'est pas mentionné, l'étiquette et le type sont envoyés.

Exemple du type dinosaure revisité

Il est ici étiqueté avec une option et une valeur par défaut.

```
Dinosaure ::=
           [PRIVATE 6] IMPLICIT SEQUENCE

{
nom        [0] IMPLICIT OCTET STRING,
longueur   [1] IMPLICIT INTEGER,
carnivore  [2] IMPLICIT BOOLEAN
                        DEFAULT TRUE,
os         [3] IMPLICIT INTEGER,
decouvert [4] IMPLICIT INTEGER OPTIONAL
}
```

ASN.1 permet de définir des types et des instances pour ces types.

Soit pour le type `Dinosaure` l'instance ayant pour valeur celle du stégosaure.

```
{"stegosaure",10,FALSE,300,1877}
```

Grammaire de ASN1

```
ModuleDefinition ::= Name "DEFINITIONS ::= BEGIN"
                                        ModuleBody "END"
ModuleBody ::= AssignmentList | Empty
AssignmentList::=Assignment|AssignmentList Assignment
Assignment ::= Name "::=" Type
Type ::= ExternalType | BuiltinType
ExternalType ::= Name "." Name
BuiltinType::=PrimitiveType|ConstructedType|TaggedType
PrimitiveType::=Integer|Boolean|BitSt|OctetSt|Any|Null

    |ObjId
ConstructedType::=Sequence|SequenceOf|Set|SetOf|Choice
TaggedType ::= Tag Type | Tag "IMPLICIT" Type
Integer ::= "INTEGER" | "INTEGER" "{"NamedNumberList"}"
Boolean ::= "BOOLEAN"
BitStr ::= "BIT STRING" | "BIT STRING"
"{"NamedBitList"}"
OctetStr ::= "OCTET STRING"
Any ::= "ANY"
Null ::= "NULL"
ObjId ::= "OBJECT IDENTIFIER"
Sequence::="SEQUENCE""{"ElementListType"}"|"SEQUENCE{}

    "
SequenceOf ::= "SEQUENCE OF" Type | "SEQUENCE"
Set ::= "SET" "{"ElementListType"}" | "SET {}"
SetOf ::= "SET OF" Type | "SET"
Choice ::= "CHOICE" "{"AlternativeTypeList"}"
Tag ::= "[ClassUnsignedNumber"]"
Class ::= "UNIVERSAL" | "APPLICATION"|"PRIVATE" |Empty
NamedNumberList ::= NamedNumber | NamedNumberList ","

    NamedNumber
NamedNumber ::= Name "(" UnsignedNumber ")" |
                        Name "(" "-" UnsignedNumber ")"
NamedBitList ::= NamedBit | NamedBitList "," NamedBit
NamedBit ::= Name "(" UnsignedNumber ")"
ElementListType ::= ElementType | ElementTypeList ","
                            ElementType
ElementType ::= NamedType | NamedType "OPTIONAL" |
                            NamedType "DEFAULT" Value
NamedType ::= Name Type | Type
AlternativeTypeList ::= NamedType |
AlternativeTypeList","NamedType
UnsignedNumber ::= Digit | UnsignedNumber Digit
Digit ::= "0"| "1"| "2"| "3"| "4"| "5"|"6"|"7"|"8"|"9"
Empty ::=
```

2 Syntaxe de transfert

Introduction à la syntaxe de transfert

- Définition des règles de codage des structures de données ASN.1 en une suite binaire pour la transmission.

- Chaque message transmis comprend une suite d'objets codés qui peuvent eux même comporter dans leur définition une suite d'objets codés ...

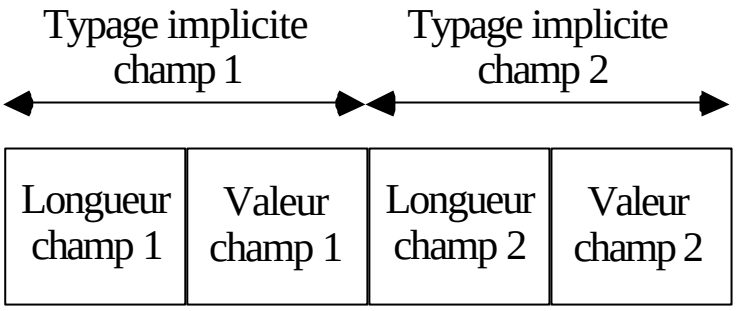
Approche récursive de l'encodage décodage.

Le codage d'un objet élémentaire est défini par des règles simples baptisées:

BER "Basic Encoding Rules"

BER est la représentation concrète de chaque objet primitif pour la communication entre entités distantes.

Première possibilité: syntaxe de transfert avec typage implicite



Option non retenue

Difficulté pour les valeurs optionnelles.

Solution retenue: syntaxe de transfert avec typage explicite

- Chaque champ est accompagné d'une information permettant de reconstruire son type à la réception.
- La zone valeur peut-être déterminée de deux façons.

Longueur connue avant l'émission

Aucune difficulté pour renseigner un champ longueur avant l'émission de la valeur.

Identificateur du message	Type champ 1	Longueur champ 1	Valeur champ 1	...

Longueur inconnue avant l'émission

La longueur n'est pas renseignée.
La valeur est déterminée par un délimiteur fin

Type	Longueur	Valeur	Délimiteur de fin

Structure des objets de la syntaxe de transfert

Chaque valeur transmise (d'un type primitif ou d'un type construit) comporte les champs suivants :

- L'identificateur du type

  codé comme un type

  ou codé comme une étiquette

  ou les deux,
- La longueur de la donnée en octets.
- La donnée,
- La marque de fin si la longueur des données est inconnue.

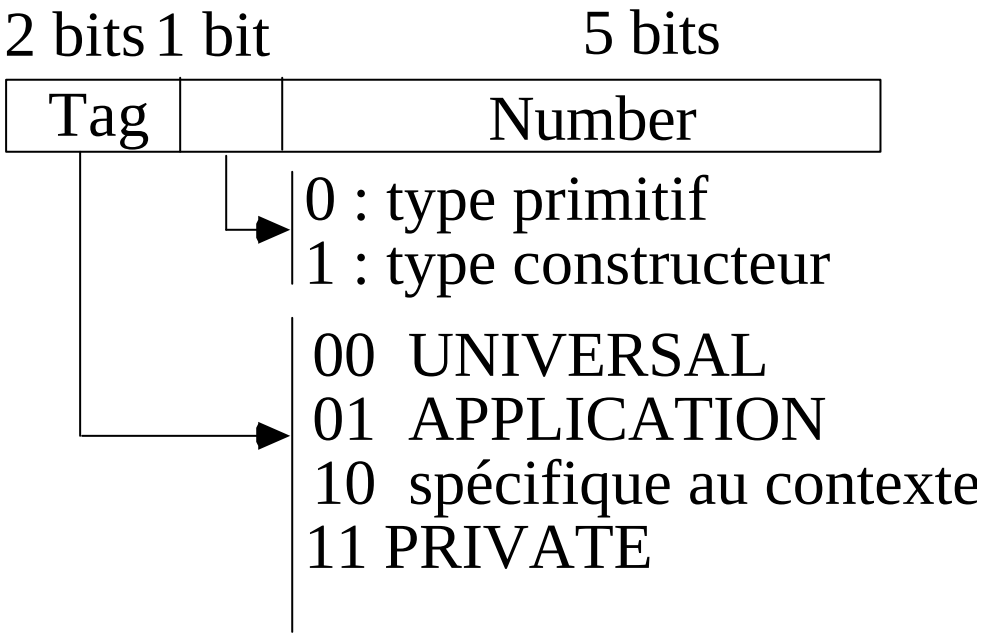
Les trois premiers champs sont obligatoires, alors que le dernier est optionnel.

L'identificateur (type ou étiquette) d'un objet

- Le premier octet d'un item transmis selon la syntaxe de transfert ASN.1

- Il possède lui-même trois sous-champs.

  Premier champ: 2 bits
  Type d'étiquette "tag"
  Second champ: 1 bit
  Type primitif ou construit
  Troisième champ: 5 bits
  Code du type universel
  ou valeur étiquette.



Classes de types et codage correspondant

Classe (français)	Classe (anglais)	Fonction	Valeur premiers bits
Universel	Universal	Type à usage général, indépendant de l'application	00
Application	Application wide	Type particulier à une application	01
Spécifique à un contexte	Context spécific	Type particulier à un contexte à l'intérieur d'une application	10
A usage privé	Private use	Type réservé à un usage privé	11

Affectation des codes de types universels et codage

Code ID	Type(français)	Type(anglais)	Bit 6: primitif:0 constructeur:1	Ident de type en hexadécimal
1	Booléen	BOOLEAN	0	01
2	Entier	INTEGER	0	02
3	Chaîne binaire	BIT STRING	0/1	03/23
4	Chaîne d'octets	OCTET STRING	0/1	04/24
5	Vide	NULL	0	05
6	Identificateur d'objet	OBJECT IDENTIFIER	0	06
7	Descripteur d'objet	Object descriptor	0	07
8	Externe	EXTERNAL	1	28
9	Réel	REAL	0	09
10	Enuméré	ENUMERATED	0	0A
11-15	Réservé			
16	Séquence, Séquence de	SEQUENCE, SEQUENCE OF	1	30
17	Ensemble, ensemble de	SET, SET OF	1	31
18	Chaine numérique	NumericString	0/1	12/32
19	Chaine imprimable	PrintableString	0/1	13/33
20	Chaîne T.61	TeletexString	0/1	14/34
21	Chaîne videotex	VideotexString	0/1	15/35
22	Chaîne IA5	IA5String	0/1	16/36
23	Heure UTC	UTCTime	0/1	17/37
24	Heure généralisée	GeneralizedTime	0/1	18/38
25	Chaîne graphique	GraphicString	0/1	19/39
26	Chaine ISO 646	VisibleString	0/1	1A/3A
27	Chaîne générale	GeneralString	0/1	1B/3B
28	Réservé			

Compléments concernant les codages

Entiers

Les entiers sont codés en complément à 2.

L'octet le plus significatif est transmis en tête.

Le codage du champ de données dépend du type de données présentes.

. Un entier positif inférieur à 128
nécessite un seul octet,
. Un entier positif inférieur à 32768
nécessite deux octets, etc ...

Booléens

Les booléens sont codés sur un octet.

0 pour FALSE
une valeur différente pour TRUE.

Chaînes de bits

Emises telles que.

Seul problème: indiquer leur longueur.

Le champ longueur indique le nombre d'octets et non le nombre de bits.

=> Problème du dernier octet.

On transmet un octet avant la chaîne de bits proprement dite qui indique combien de bits (entre 0 et 7) du dernier octet ne sont pas utilisés.

Exemple: Codage de '010011111'

En hexadécimal : 07 4F 80.

Chaînes d'octets

Les octets sont numérotés de gauche à droite ("big endian").

La valeur nulle est repérée par un champ de données inexistant (0 octet de longueur).

Sequence et Set

- On commence par transmettre le type ou l'étiquette pour la séquence ou l'ensemble, puis la longueur totale du codage de tous les champs, puis les valeurs.

- Les champs d'une séquence doivent être envoyés dans l'ordre. Les champs d'un ensemble peuvent être transmis dans un ordre quelconque.

- Si deux ou plusieurs champs d'un ensemble ont le même type, ils doivent être étiquetés afin que le récepteur puisse les distinguer les uns des autres.

Choice

Le codage d'une valeur de type `CHOICE` est celui de la structure de données qui est réellement envoyée (avec ce codage le récepteur peut interpréter un type indéfini).

Exemple: si on a le choix entre un `INTEGER` et un `BOOLEAN`, et si la structure de données manipulée est en fait un `INTEGER`, alors les règles de codage du type `INTEGER` s'appliquent.

Approfondissement: Réalisation des codeurs/décodeurs (compilateurs de protocoles)

- Le codeur/décodeur doit réaliser la conversion entre la représentation concrète de chaque machine et la syntaxe de transfert.

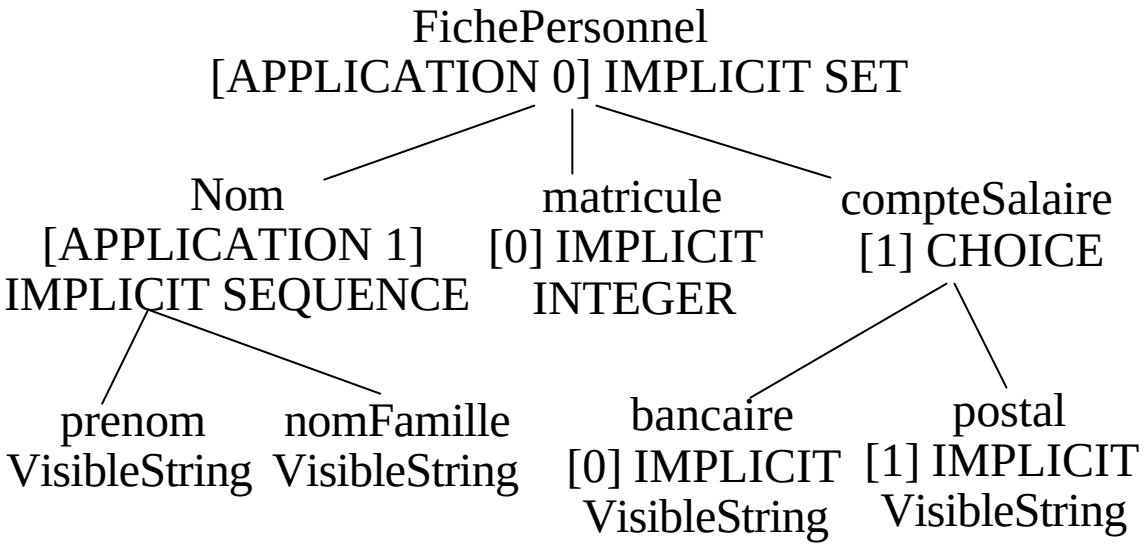
- Pour faciliter ces conversions de syntaxe, il est commode d'adopter une représentation de la syntaxe abstraite dont la structure reflète l'APDU

- L'analyseur syntaxique d'une définition de données en syntaxe abstraite génère une structure d'arbre (première étape habituelle d'un compilateur) sur lequel travaille ensuite les convertisseurs..

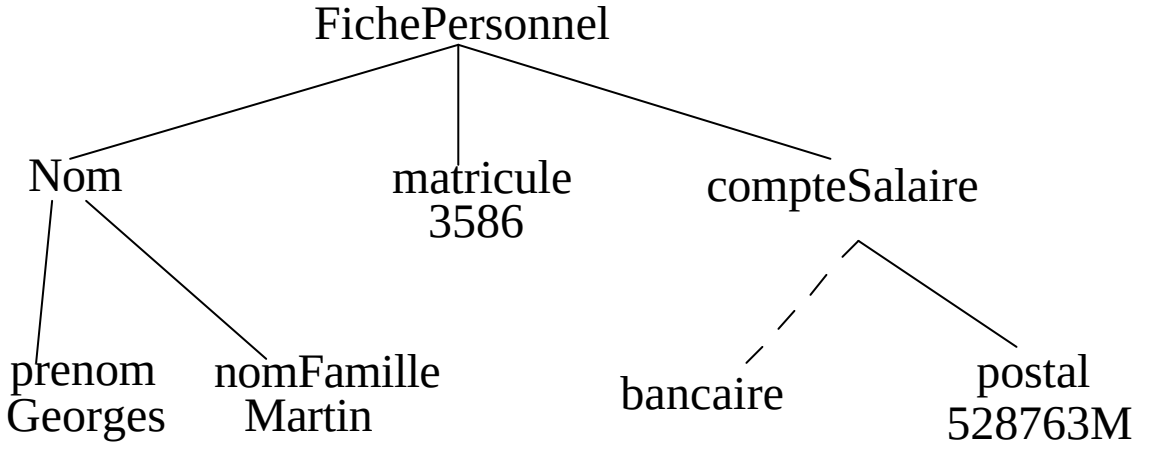
Exemple

```
FichePersonnel ::= [APPLICATION 0] IMPLICIT SET
{Nom,
    matricule    [0] IMPLICIT INTEGER,
    compteSalaire[1] CHOICE {
        bancaire     [0] IMPLICIT VisibleString,
        postal       [1] IMPLICIT VisibleString}}
Nom ::= [APPLICATION 1] IMPLICIT SEQUENCE {
    prenom       VisibleString,
    nomFamille   VisibleString}
```

Représentation par arbre de type



Arbre des valeurs pour la ficheMartin



Fonctionnement du codeur

Première passe

```
60   *                                          fiche perso
     61   *                                     nom
               1A   07   47656F72476573         prénom
               1A   06   4D417274494E           nomFamille
     80   02        0E02                       matricule
     A1   *                                     compte
               81   02   3532383736334D         postal
```

. Initialisation des zones valeurs qui correspondent aux symboles terminaux avec détermination des longueurs dans ce cas.

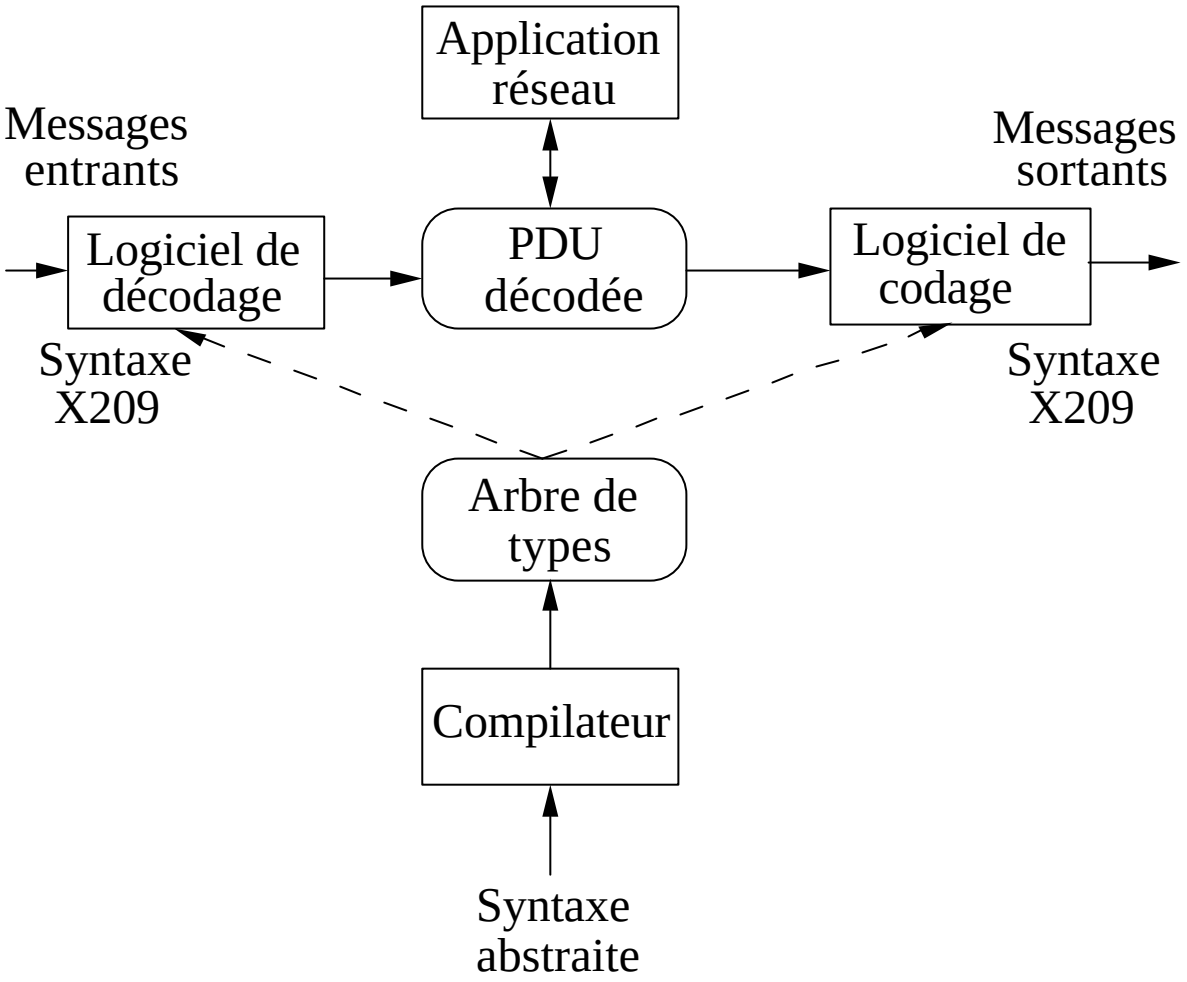
. Les zones longueurs des noeuds intermédiaires sont non calculées.

Deuxième passe

```
60   22
     61   11                                    nom
               1A   07   47656F72476573         prenom
               1A   06   4D417274494E           nomFamille
     80   02        0E02                       matricule
     A1   09                                    compte
               81   02   3532383736334D         postal
```

Détermination complète des longueurs

Ensemble des outils logiciels



Conclusion ASN1

- ASN1 est uniquement définie pour coder des données transportées dans le mode message.

- ASN1 achemine toutes les informations nécessaires pour interpréter de façon non ambigüe toute donnée à la réception => Un peu de perte de place

- ASN1 cherche en codant les informations au plus court à optimiser l'encombrement des messages .

=> Des temps d'exécution plus longs pour coder et décoder un message.

=> Des performances et des possibilités moyennes.

Améliorations

- Simplifier les possibilités offertes dans le cadre d'implantations particulières pour aller plus vite (ASN1 pour le temps réel).

- Introduire des fonctionnalités pour l'approche appel de procédure distante et l'approche objets répartis (les IDL).

Le langage IDL CORBA

"Interface Definition Language"

Généralités

- Le langage utilisé par CORBA pour spécifier les interfaces d'objets:
  les noms des méthodes,
  les paramètres ,
  les exceptions ...
syntaxe abstraite des interfaces d'objets CORBA

- L'IDL CORBA n'est pas un langage destiné a produire des programmes exécutables habituels (langage algorithmique).

- C'est un langage "déclaratif" qui permet d'établir les correspondances entre un client utilisateur (en langage du programme client) et un objet serveur (défini en langage du serveur).
- Sa compilation produit les souches clients et serveurs.
- Défini à partir de C++ il présente néanmoins des différences avec C++ (notion de langage "pivot").

Les types de base (primitifs)

Entiers

- **short** entiers 16 bits -2^{15} à $2^{15}-1$
- **long** entiers 32 bits -2^{31} à $2^{31}-1$
- **unsigned short** entiers non signés 16 bits 0 à $2^{16}-1$
- **unsigned long** entiers non signés 32 bits 0 à $2^{32}-1$

Flottants (à la norme IEEE)

- **float** flottants 32 bits (simple précision)
- **double** flottants 64 bits (double précision)

Booléens

- **boolean** Deux valeurs TRUE/FALSE

Caractères

- **char** Codé sur 8 bits.

Octets

- **octet** 8 bits quelconques

Type quelconque

- **any** Désigne n'importe quel type.

Les types complexes

- struct Type article

```
struct type_triplet {
    float x , y , z ;    };
```

- enum Type énuméré

```
enum type_rvb {
    rouge , vert , bleu };
```

- union Type variable avec discriminant

```
union montant switch (code_pays) {
    case FR : unsigned float francs ;
    case SP : unsigned float pesetas ;
    case IT : unsigned float lires ; };
```

- [] Type tableau

```
typedef long tableau[25]
```

- sequence Suite ordonnée de valeurs

```
typedef sequence<long> zone;
typedef sequence<sequence<short,5>> zone;
```

- string Type chaîne

```
typedef string<50> chaine ;
```

Les caractéristiques objets

Un exemple regroupant les principales caractéristiques

```
module Voiture
{
    struct chassis  {
            float longueur;
            float largeur;
                    }
    /* Définition d'une classe cabriolet */
    interface Cabriolet :
véhicule_thermique ;
    {
    attribute readonly integer puissance ;
    exception Panne {
        short code_exception ;
        string explication ; }

    long accélère ( in short durée);
    void réparer ( ....
```

Quelques éléments des caractéristiques objets

Modules

Permettent de structurer une application en regroupant des déclarations de types et d'interfaces logiquement associées.

Interfaces

Regroupent des attributs et des déclarations d'opérations.

Attributs

Variables d'un objet accessibles de l'extérieur de l'objet par des méthodes prédéfinies lire : _get et écrire: _set (cas particulier des attributs readonly).

Opérations

Associées à chaque méthode la spécification des opérations comporte:

. un nom, une liste de paramètres avec un atttribut directionnel **in**, **out**, **inout**.

. la spécification d'un type d'information attendue en retour (**void** si aucune information n'est retournée).

Exceptions

La spécification des exceptions permet de définir des types de problèmes prévus à l'avance.

Une exception comporte des informations permettant de préciser la nature du problème.

A une exception correspond un traitant d'exception qui est décrit dans le langage d'exécution.

On doit spécifier pour chaque opération lève quelles exceptions sont levées (mot clé **raises**).

Héritage

Les spécifications d'interfaces peuvent être réutilisées. L'héritage peut être multiple.

interface Cabriolet : véhicule_thermique;

L'interopérabilité en CORBA

- Dans ses premières versions CORBA spécifiait de façon insuffisante les ORB:
messages échanges, référence d'objets.
=> Des implantations de CORBA pouvaient être conformes sans être interopérables.

Avec CORBA 2, l'OMG a défini des protocoles normalisés d'interactions entre ORB.

GIOP 'General Inter ORB Protocol'
Un standard de communication générique entre ORB.
- Ensemble des messages à utiliser
- Syntaxe de transfert des données CDR "Common data representation"

IIOP 'Internet Inter ORB Protocol'
Une implantation de IIOP sur TCP/IP.
ESIOP 'Environment Specific Inter ORB Protocol'
Implantation utilisant le RPC DCE.

GIOP 'General Inter ORB Protocol'

But principal: fournir un cadre pour permettre l'invocation d'objets distants quelquesoit l'ORB utilisé pour les gérer.

Référence Interopérable d'objets
IOR Interoperable Object Reference

Définit une structure de donnée normalisée utilisable pour identifier de manière unique les objets.

- L'adresse de la machine ou se trouve l'ORB
- Le moyen d'accéder à cet ORB
  (Ex: son adresse de transport).
- Une référence locale de l'objet laissée au libre choix de chaque ORB.

Un exemple d'IOR

1.0:sunrpc_2_0x61a79_153029598/rm=tcp_163.173.128.208_3503:IDL%3AMath%3A/Math/Calcul

Protocole GIOP (1)

7 types de messages différents.

Request : émis par le client pour invoquer des opérations sur les objets ou serveurs CORBA.

- identificateur de requête.
- valeurs des paramètres en entrée (mode in et inout).

Reply : émis par le serveur en réponse à un message **Request**.

- identificateur de la requête.
- valeur des résultats de l'opération ainsi que les paramètres en sortie de celle-ci (mode inout et out).
- valeur de l'exception s'il y en a.

CancelRequest: un client d'informe un serveur qu'il n'attend plus la réponse à une requête.

LocateRequest : localisation d'une référence IOR (si le serveur interrogé est capable de recevoir des requêtes pour l'objet donné)

Sinon détermination de la nouvelle adresse IOR permettant d'accéder à l'objet (localisation migration des objets).

Protocole GIOP (2)

LocateReply : Réponse à un message de type LocateRequest.

- Un booléen indique si la référence IOR désigne un objet local.
- Si ce booléen est à faux, alors le message contient en plus une référence IOR indiquant la nouvelle localisation de l'objet.

CloseConnection : Le serveur interrompt son service.

Toutes les requêtes en attente de réponses ne seront jamais traitées.

MessageError : Erreur de protocole

En réponse à tout message GIOP qui ne peut pas être traité car il est erroné.

numéro de version inconnu,
type de message inconnu,
en-tête erroné...;

La syntaxe de transfert de CORBA "CDR Common Data Representation"

Le format dans lequel CORBA représente les données décrites en IDL transférées de machine à machine.

Les types primitifs

- Notion de flot d'octets ("octets streams") qui encodent les données: 0 .. n-1
- Contraintes de remplissage des flots d'octets par les différents types (alignement).

Type	Alignement
char	1
octet	1
short	2
unsigned short	2
long	4
unsigned long	4
float	4
double	8
boolean	1
enum	4

Types entiers

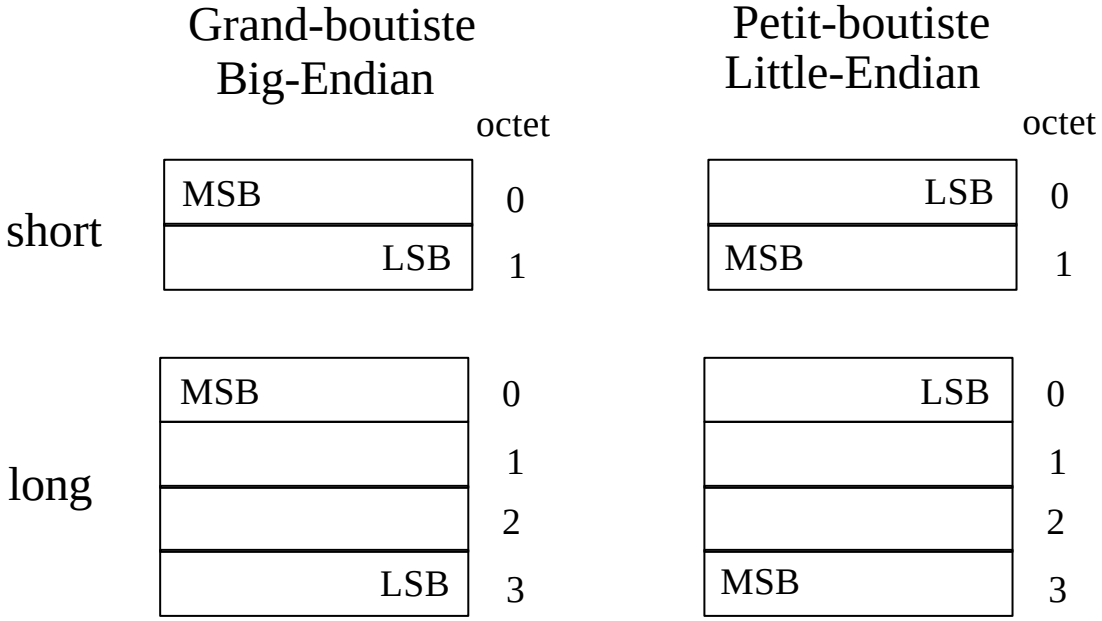
short:
entiers signés en complément à 2 sur 16 bits

unsigned short :
entiers non signés sur 16 bits

long:
entiers signés en complément à 2 sur 32 bits

unsigned long :
entiers non signés sur 32 bits

Deux versions de codage
(grand boutiste et petit boutiste)



Types flottants

Selon la normalisation IEEE

signe s, exposant e, mantisse ("fraction part") f

float:

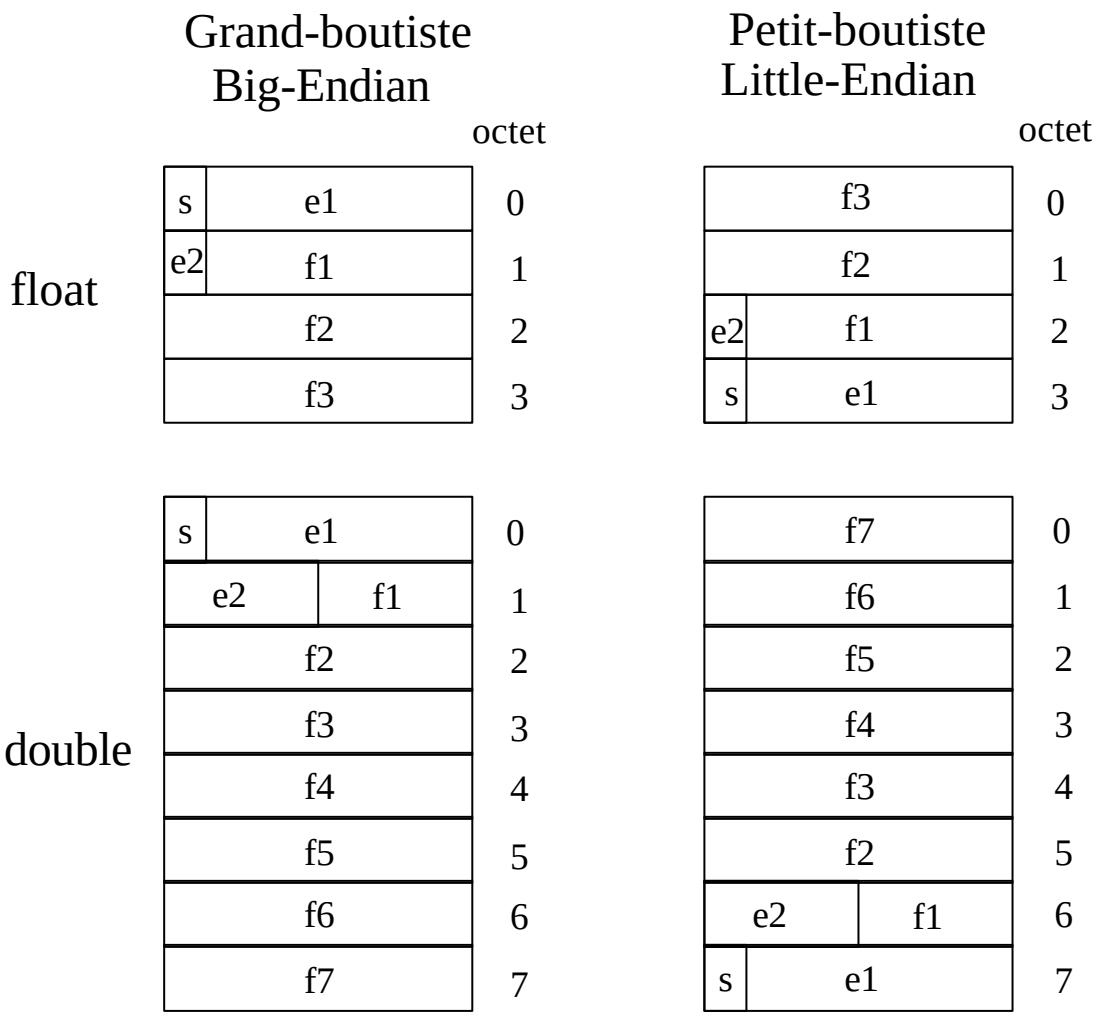
flottants sur 32 bits s:1bit, e:8 bits, f:23 bits

Valeur := $(-1)^s * 2^{(e-127)} * (1+f)$

double:

flottants sur 64 bits s:1bit, e:10 bits, f:52 bits

Valeur := $(-1)^s * 2^{(e-1023)} * (1+f)$



Type octet

Valeurs sur 8 bits non interprétées.

Aucune conversion n'est appliquée pendant la transmission.

On peut par exemple considérer qu'il s'agit d'entiers sur 8 bit non signés.

Type booléen

Encodés sur un octet.

TRUE à la valeur 1
FALSE à la valeur 0.

Type caractère

Encodés sur un octet.

Selon l'alphabet ISO 8859.1 (Latin-1).

Les types construits

Principe général: réalisation de l'alignement "marshalling" par compilation d'une liste d'appels à des routines de conversion. Pas de contraintes particulières de frontières de mots ou d'octets.

Type struct

Les composants d'une structure sont rangées dans un message dans l'ordre des déclarations.

Type union

On code d'abord le discriminant du type union puis on code la représentation de la variable à transmettre selon le type associé au discriminant.

Type array

On code le tableau par une suite d'éléments de même type. Pour les tableaux multidimensionnels on utilise l'ordre des indices.

Type sequence

On code d'abord le nombre d'éléments dans la séquence par un entier non signé long.

Type chaîne

On code d'abord la longueur de la chaîne par un entier non signé long. Puis la suite des caractères en ISO latin 1

Type enum

On code les types énumérés sur entier non signé long. les valeurs des entiers associés sont déterminés par l'ordre dans lequel apparaît chaque valeur énumérée en commençant par 0.