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.

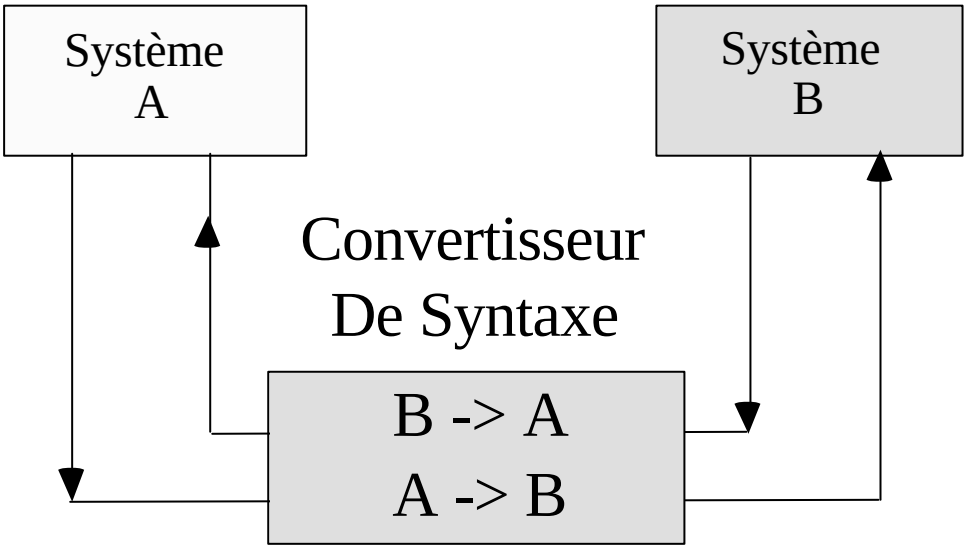
Utiliser sur chaque **machine réelle** un **interpréteur** du langage intermédiaire dans le langage particulier d'une machine cible.

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.

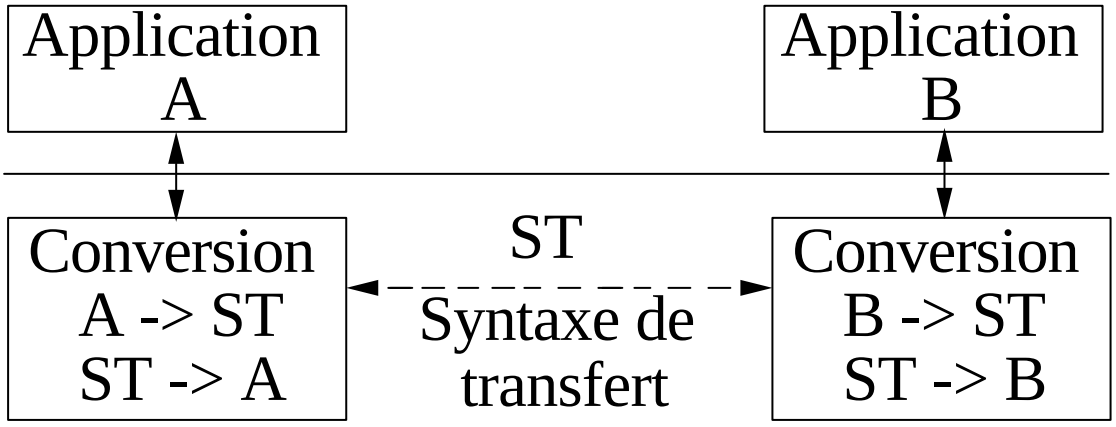
Solution d'un langage intermédiaire "pivot" commun à tout le réseau

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

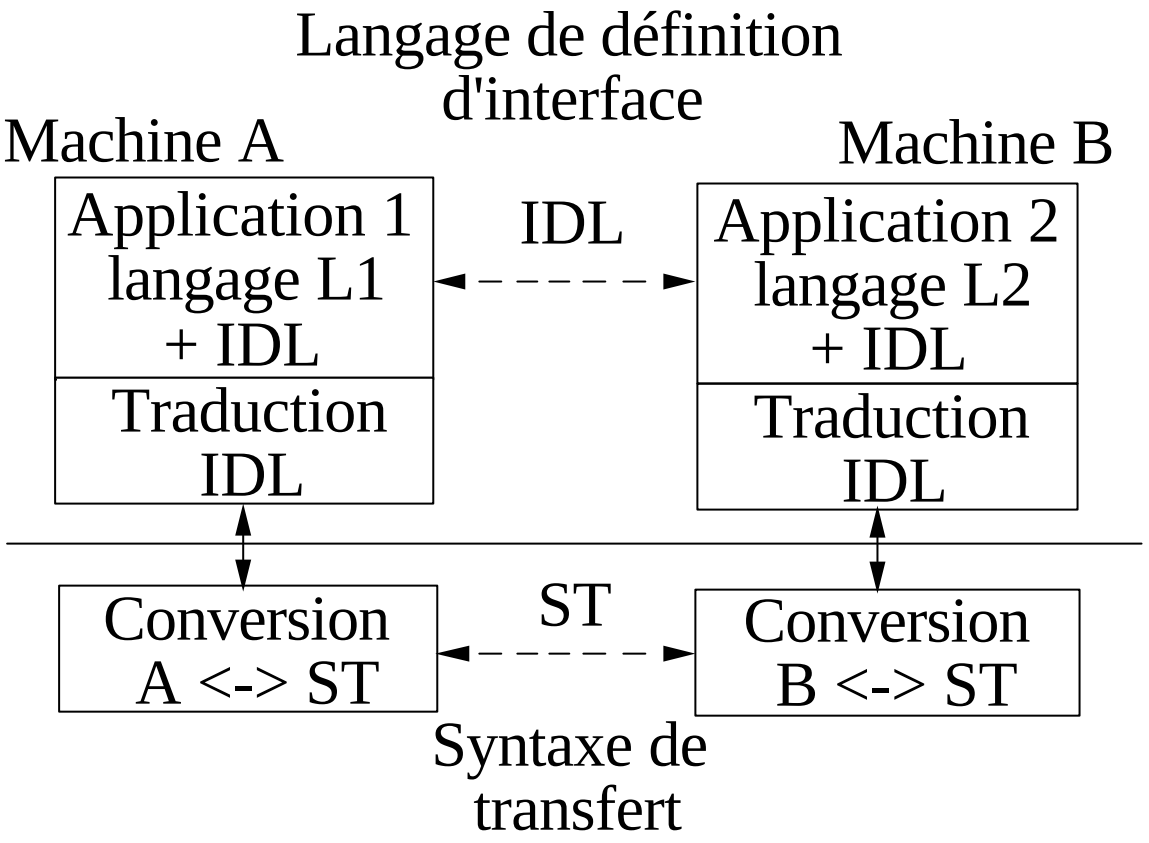
Le langage d'interface permet de définir de façon **universelle** les **services** d'un logiciel réseau et les **données** nécessaires.

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 ...

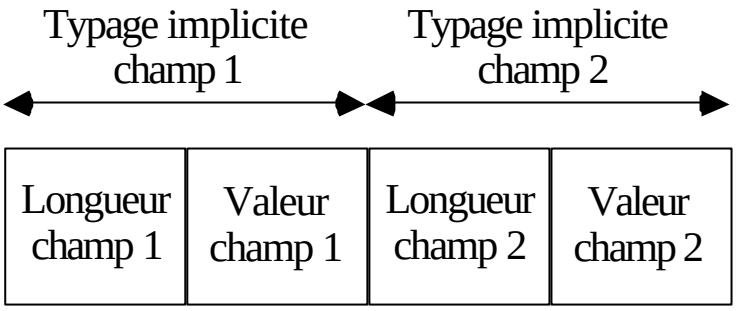
Approche récursive de l'encodage décodage.

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.

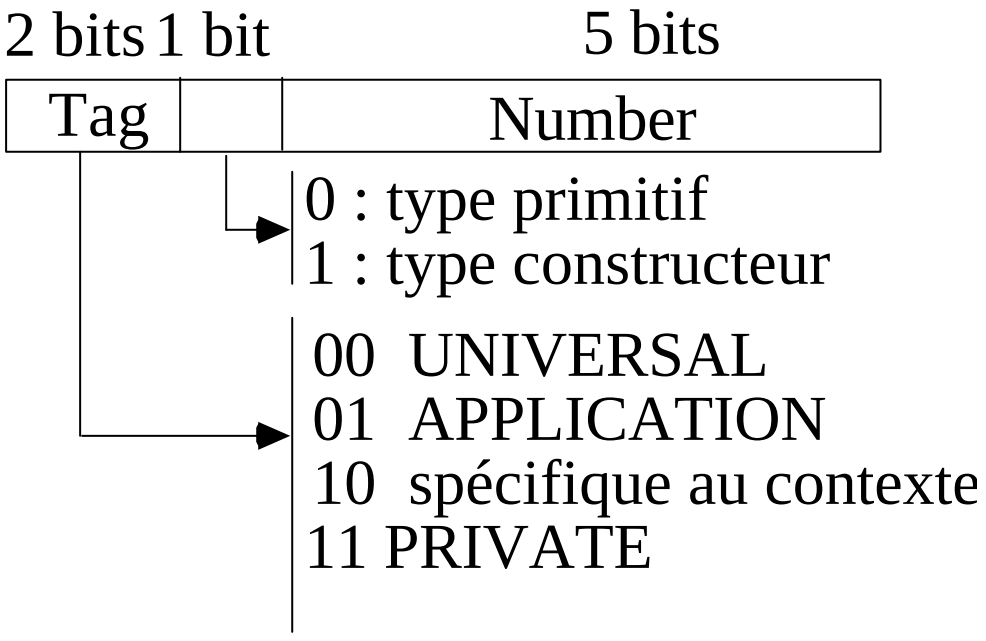
Les trois premiers champs sont obligatoires, alors que le dernier est optionnel.

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.

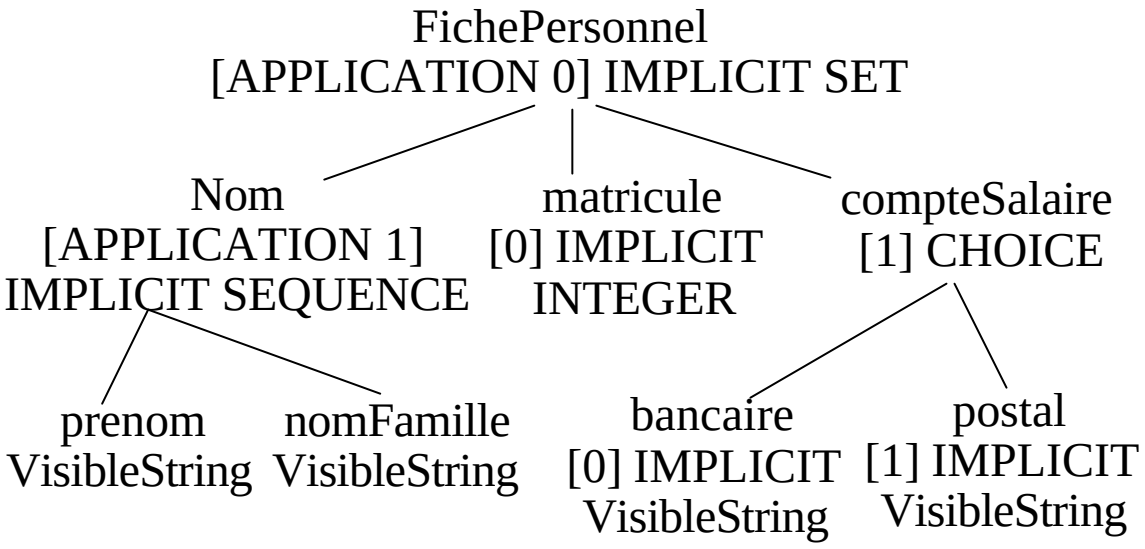
- Pour faciliter ces conversions de syntaxe, il est commode d'adopter une représentation de la syntaxe abstraite dont la structure reflète l'APDU

- 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..

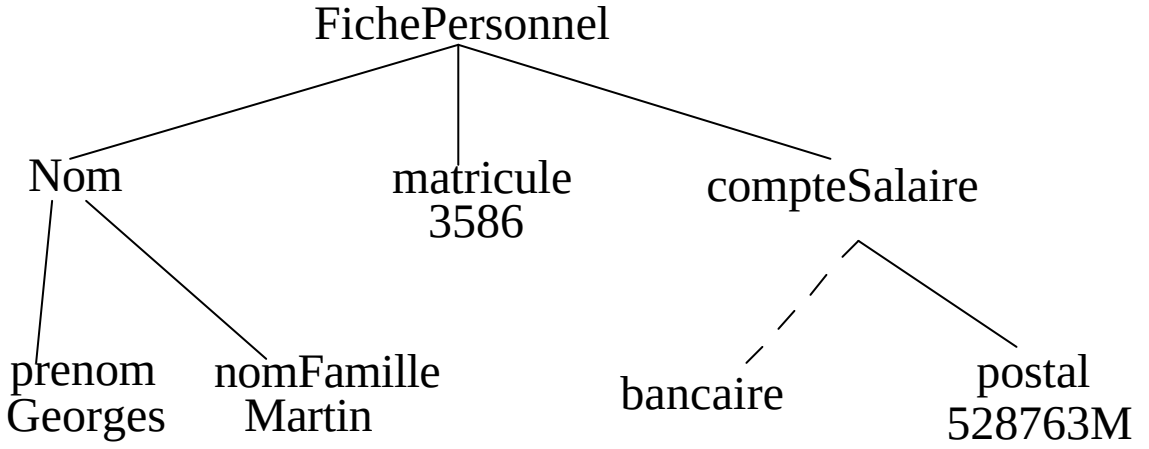
Exemple

```
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.

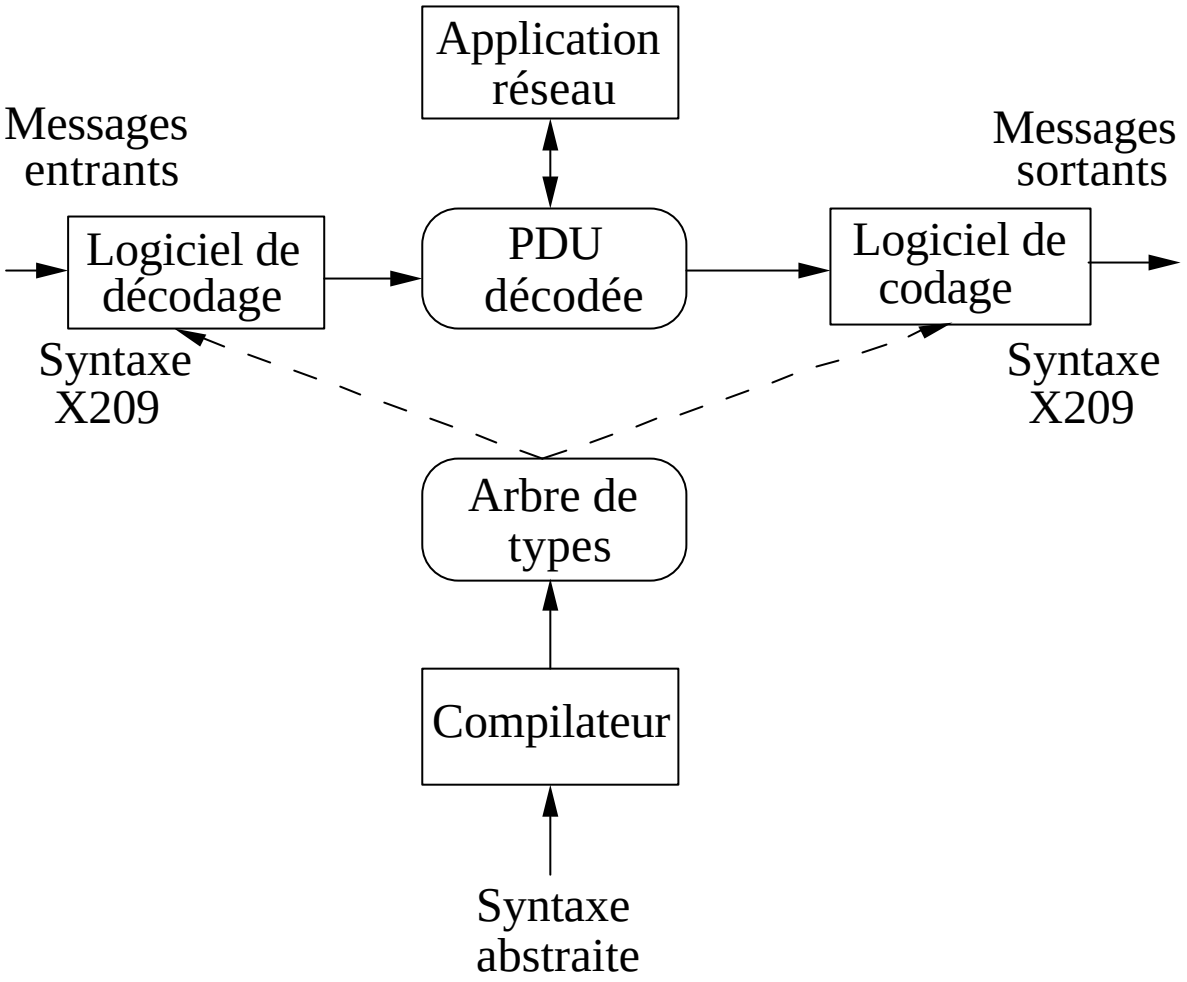
. Les zones longueurs des noeuds intermédiaires sont non calculées.

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

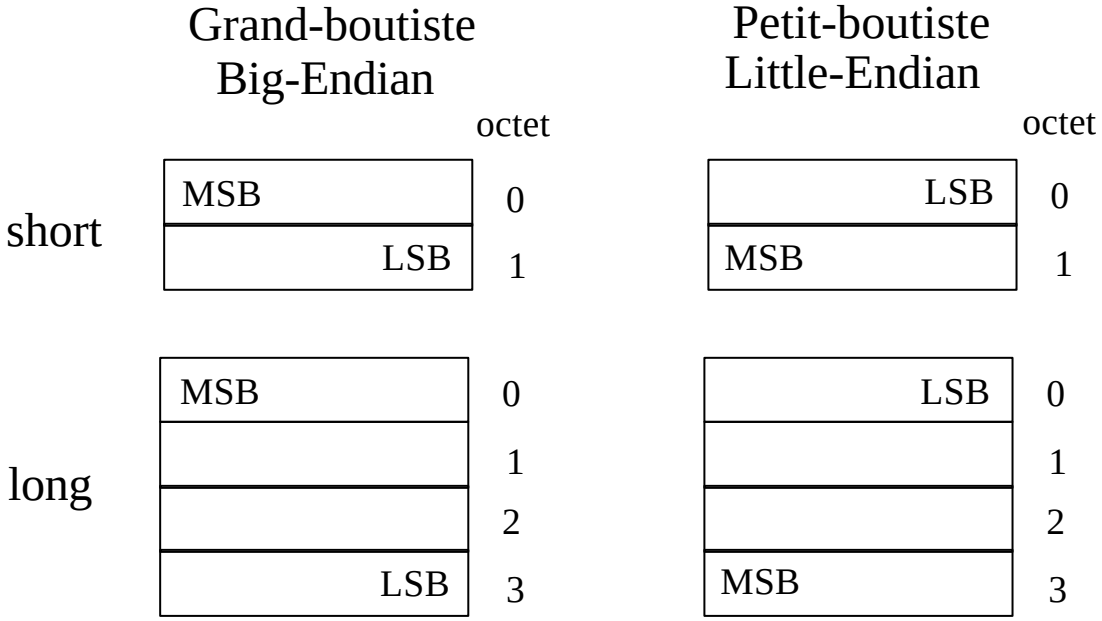
- Notion de flot d'octets ("octets streams") qui encodent les données: 0 .. n-1
- Contraintes de remplissage des flots d'octets par les différents types (alignement).

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:

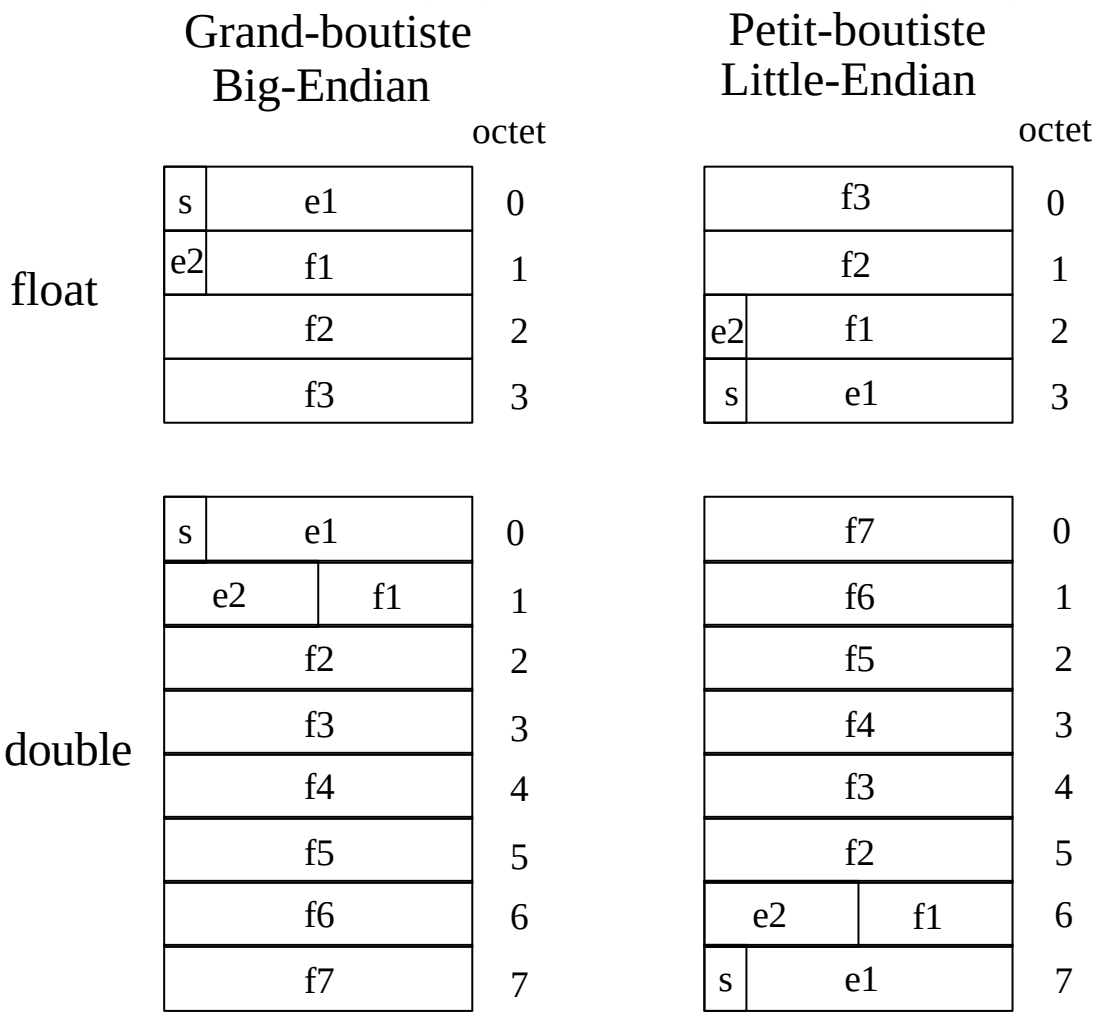
flottants sur 32 bits s:1bit, e:8 bits, f:23 bits

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.