La norme STEP
(Standard ISO 10303)
Rapport technique
Equipe *Architectures et Systèmes*

Alain Plantec

2000

Table des matières

Table des figures

Chapitre 1

Introduction

Le standard STEP[1] est la norme ISO 10303 élaborée par le sous-comité ISO TC 184/SC4. L'objectif de cette norme est de permettre aux applications informatiques industrielles d'échanger et de partager des données indépendamment des spécificités des différents systèmes informatiques échangeant ou partageant des informations [ISO94a].

Les travaux de normalisation comprennent d'une part le développement d'une technologie fournissant des méthodes et outils informatiques neutres pour la description, la validation et la manipulation des informations de définitions de produits, et d'autre part, la spécification de modèles de données standards, appelés protocoles d'application et organisés par métiers industriels. Ces protocoles d'application sont développés dans le cadre strict de la technologie STEP. Ils sont spécifiés dans le langage de modélisation EXPRESS [ISO94b] et constituent une librairie de concepts réutilisables pour développer des modèles de données dans différents secteurs industriels [Bou95a]. Les protocoles d'échanges et de partages des données définissent un format neutre d'échange de données [ISO94c] et une interface standard d'accès aux données [ISO94d].

La nature de STEP est double. En tant que norme ISO dont la finalité est la standardisation de la représentation des produits, elle s'adresse principalement aux groupes industriels dont le processus utilise les techniques de CAO ou CFAO. En tant que technologie, STEP présente un champ d'utilisation beaucoup plus large puisqu'elle peut s'appliquer à d'autres informations que les données de définition de produits [Aa94].

Les travaux concernant la norme *STEP* sont menés depuis 1983 par l'*ISO*, le secrétariat étant assuré par le *NIST*[2]. Cet effort de standardisation s'est rapidement internationalisé avec notamment la participation de *PROSTEP* en Allemagne et de l'association *GOSET* en France. L'activité américaine, (celle du *PDES*[3] et du *NIST*) demeure prépondérante. Les travaux de normalisation sont toujours très actifs.

La suite de ce chapitre s'intéresse plus particulièrement à la technologie STEP. La première partie décrit l'architecture de la norme, la deuxième partie décrit les principales caractéristiques du langage EXPRESS [ISO94b], la troisième partie explicite la méthode de mise en œuvre d'un protocole d'application, les quatrième et cinquième parties décrivent les méthodes de mise en œuvre

1. STandard for the Exchange of Product model data
2. National Institute of Standards and Technology
3. Product Data Exchange using STEP

des schémas EXPRESS, respectivement le format d'échange des données [ISO94c], et l'interface d'accès aux données, la SDAI[4] [ISO94d].

4. Standard Data Access Interface

Chapitre 2

L'architecture de la norme

STEP consiste en un ensemble de standards internationaux dont l'objectif est soit la spécification d'une méthode de description, de validation ou de mise en œuvre, soit la définition de schémas de données standards. Chaque standard compose une partie de la norme et est développée au sein d'une série. Les sept séries actuellement mises en œuvre sont décrites dans [ISO94a], la numérotation des parties reflète la structure de la norme :

- la partie 1 décrit la norme ISO 10303 et ses principes fondamentaux,
- les parties 11 à 19 décrivent les méthodes de description; il s'agit de la spécification du langage de modélisation textuel EXPRESS, du langage de modélisation graphique EXPRESS-G et du langage d'instanciation de modèles conceptuels EXPRESS-I,
- les parties 21 à 29 décrivent les méthodes de mise en œuvre; elles consistent en la spécification du format d'échange des données, d'une spécification neutre de l'interface d'accès aux données (SDAI) et des spécifications de la mise en œuvre de la SDAI dans des langages cibles tel que C, C++ [Str92] ou IDL [Obj95],
- les parties 31 à 39 décrivent les méthodes de validation et de tests de conformité des mise en œuvre des modèles de données standards,
- les parties 41 à 99 décrivent les ressources intégrées génériques qui sont des schémas de données de portée très générale puisque indépendants de tout contexte d'utilisation particulier,
- les parties 101 à 199 décrivent les ressources intégrées d'application qui regroupent des schémas de données plus spécifiques à un ensemble de domaines d'application,
- les parties 201 à 232 décrivent les protocoles d'application qui exploitent les ressources intégrées pour la spécification de modèles de données spécifiques à des domaines d'application particuliers.

La figure 2.1, extraite de [War94] schématise l'architecture technique de STEP. De par sa finalité, STEP est exploitée au travers des protocoles d'application. Ces modèles conceptuels standards s'appuient sur les ressources intégrées pour la spécification des concepts. Les protocoles d'application sont mis en œuvre pour l'échange ou le partage des données à l'aide du format d'échange et de la SDAI. Toutes les ressources standards et les méthodes de mise en œuvre sont décrites en partie avec les langages EXPRESS et EXPRESS-G et sont validées suivant les méthodes de test de conformité.

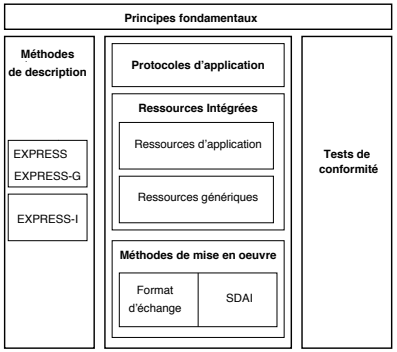


FIG. 2.1 – *L'architecture de STEP*

Chapitre 3

Le langage EXPRESS

La norme STEP est composée de plusieurs standards parmi lesquels la spécification du langage de description de modèles de données, EXPRESS [ISO94b]. Ce langage joue un rôle central pour la description et la mise en œuvre de la norme [Bou95b] : les outils de la norme et les protocoles d'application sont décrits en grande partie avec EXPRESS. Ce langage est le résultat d'une synthèse entre les langages de modélisation de données tels que NIAM et IDEF1X, les constructions impératives des langages de programmation tels que Pascal, ADA et C++ et certains éléments du langage SQL. Le manuel de référence du langage EXPRESS [ISO94b] contient aussi la spécification d'EXPRESS-G qui est la forme graphique d'EXPRESS.

La première version d'EXPRESS est, depuis 1994 le standard ISO 10303-11 [ISO94b]. Cette première version est présentée dans le chapitre 3.1. Depuis, de nombreuses extensions du langage ont été étudiées et mises en œuvre. Ces extensions ont principalement comme objectifs de permettre l'expression du comportement des données [ISO94e, ISO95d] et l'expression de vues logiques dans la description de modèles de données [ISO96]. Ces différents travaux sont actuellement exploités pour la spécification d'une nouvelle version du langage. Les évolutions principales du langage sont présentées dans le chapitre 3.3.

3.1 La première version d'EXPRESS

EXPRESS est défini comme un langage de modélisation de concepts et des contraintes appliquées à ces concepts. Ce n'est pas un langage de programmation, un modèle EXPRESS n'est pas exécutable. L'objectif est de disposer d'un langage pour la description des données et l'exploitation automatique de cette description par des outils informatiques. Cet objectif justifie le choix d'un langage textuel, spécifié par une grammaire exploitable par un analyseur syntaxique.

3.1.1 Le schéma

L'unité de modélisation des données est le *schéma*. Un schéma constitue un contexte de déclaration de constantes, de types, de classes d'objets et de contraintes.

3.1.2 Les entités

La description d'une classe d'objets ou *entité* associe un nom d'*entité* à un ensemble de propriétés. Les cardinalités attribuées à ces propriétés peuvent être spécifiées par des aggrégations. De plus,

une propriété peut être définie comme obligatoire ou facultative. Les propriétés se décrivent sous la forme d'attributs *explicites*, *dérivés* ou *inverses* :

- un attribut *explicite* représente une propriété dont la valeur doit être définie lors de la création d'une instance de l'entité,
- un attribut *dérivé* représente une propriété dont la valeur est calculée par l'évaluation d'une expression,
- un attribut *inverse* peut être spécifié dans le cas ou une autre entité est associée à l'entité courante par un attribut explicite; un attribut *inverse* définit la cardinalité de l'association dans le contexte de l'entité courante.

3.1.3 Les contraintes

EXPRESS permet la spécification de contraintes. Il est ainsi possible de contraindre individuellement ou globalement le domaine de valeur des attributs, et des instances d'entités :

- une *contrainte d'unicité* est spécifiée dans le contexte de la définition d'une entité; elle spécifie un ensemble d'attributs pour lesquels il est invalide de créer deux instances ayant les mêmes ensembles de valeurs,
- une *contrainte de domaine* est spécifiée dans le contexte de la définition d'un type ou d'une entité; elle contraint le domaine de valeur d'un type ou d'un attribut d'une instance; elle est vérifiable individuellement sur chaque entité,
- une *contrainte globale* est spécifiée dans le contexte d'un schéma; elle exprime une contrainte portant sur l'ensemble des instances d'une ou de plusieurs entités du schéma.

Pour la spécification des contraintes et du calcul des attributs dérivés, EXPRESS dispose d'un langage procédural qui autorise l'utilisation des opérateurs classiques de construction d'expression arithmétiques et logiques, des instructions classiques telles que l'affectation, une instruction conditionnelle et plusieurs instructions itératives. Les traitements procéduraux peuvent être décrits dans des procédures ou des fonctions.

3.1.4 L'héritage

Une entité peut hériter d'une ou de plusieurs autres entités. Un sous-type hérite de toutes les propriétés et contraintes locales de ses super-types. De plus, les contraintes globales qui s'appliquent aux super-types, s'appliquent aussi aux sous-types. Un sous-type peut redéfinir un attribut hérité. Il est notamment possible de redéfinir un attribut explicite en attribut dérivé, sa valeur pouvant être calculée dans le contexte du sous-type. Cette redéfinition doit cependant toujours respecter les contraintes définies dans le super-type.

Une spécificité du langage EXPRESS est la possibilité de contraindre la relation d'héritage dans la définition du super-type, par la spécification d'une expression pouvant utiliser trois opérateurs d'association entre le super-type et les sous-types : *AND* pour le *et*, *ONEOF* pour le *ou inclusif* et *ANDOR* pour le *ou exclusif*. L'algorithme de composition est défini dans [ISO94b]. Ces opérateurs contraignent la composition des entités au sein de la hiérarchie et permet la spécification implicite de nouvelles entités, compositions booléennes des sous-types. Les entités décrites dans une telle hiérarchie sont dites *entité complexes*. L'exemple 3.1 montre la spécification d'une hiérarchie d'entités complexes.

```
1  ENTITY flotteur ABSTRACT SUPERTYPE
2    OF (ONEOF((ONEOF(bathy, syscan) AND (diag ANDOR tx)), (surdrif AND tx)));
3      message_len : integer;
4  END_ENTITY;
5
6  ENTITY bathy SUBTYPE OF (flotteur);
7      balises : array[1:4] of integer;
8  END_ENTITY;
9
10 ENTITY syscan SUBTYPE OF (flotteur);
11     balise : integer;
12 END_ENTITY;
13
14 ENTITY surdrif SUBTYPE OF (flotteur);
15     balises : array[1:4] of OPTIONAL integer;
16 END_ENTITY;
17
18 ENTITY diag SUBTYPE OF (flotteur);
19 END_ENTITY;
20
21 ENTITY tx SUBTYPE OF (flotteur);
22 END_ENTITY;
```

Cet ensemble d'entités est extrait d'un schéma qui décrit différents types de flotteurs effectuant des mesures physiques en milieu marin. L'entité flotteur spécifie le concept abstrait de flotteur. Cette classification montre une hiérarchie d'entités complexes, spécifiée par la définition des sous-types de flotteur. Un flotteur bathy est aussi soit un diag, soit un tx, soit les deux. Un flotteur syscan est aussi soit un diag, soit un tx, soit les deux. Un flotteur surdrif est aussi obligatoirement un tx.

FIG. 3.1 – *Une hiérarchie complexe*

3.1.5 Les fonctions prédéfinies

EXPRESS dispose d'un ensemble de fonctions prédéfinies. Ces fonctions permettent d'effectuer les calculs arithmétiques et trigonométriques classiques sur des données mais aussi de requérir des informations sur le type des attributs et des variables. Dans cette seconde catégorie de fonctions est notamment définie la fonction **TypeOf**, qui retourne l'ensemble comprenant le nom du type de la variable qui lui est passée en argument, ainsi que ceux de tous ses super-types.

3.1.6 La réutilisation entre schémas

Un *schéma* peut utiliser d'autres *schémas*, de sorte qu'il est possible de décrire des *schémas* généraux, destinés à être référencés ou spécialisés par d'autres *schémas* plus spécifiques.

Comme le montre l'exemple 3.2, la réutilisation des types, fonctions et entités peut s'exprimer de deux manières conceptuellement distinctes, soit par *utilisation*, soit par *référence*. L'utilisation s'exprime par le mot clé *USE* et la référence par *REFERENCE*. Ces deux possibilités de réutilisation se distinguent par la manière dont sont considérées les entités définies dans le schéma utilisé ou


```
1  SCHEMA coreWidget;
2
3    ENTITY graphic ABSTRACT SUPERTYPE;
4      name              : STRING;
5      x,y,w,h           : INTEGER;
6    END_ENTITY;
7
8  END_SCHEMA; -- coreWidget
9
10 SCHEMA widgetContainer;
11   REFERENCE FROM coreWidget;
12
13   ENTITY container ABSTRACT SUPERTYPE;
14     name            : STRING;
15     owned_graphics  : LIST [1:?] OF graphic;
16   END_ENTITY; ...
17
18 END_SCHEMA; -- widgetContainer
19
20 SCHEMA tk_widgetContainer;
21   USE FROM widgetContainer(container);
22
23   TYPE tk_border = ENUMERATION OF (tk_top, tk_bottom, tk_left, tk_right);
       END_TYPE;
24
25   ENTITY tk_container SUBTYPE OF (container);
26     packing_border : tk_border;
27   END_ENTITY;
28
29   ENTITY tk_inputs;
30     input_graphic_list : list [0:?] of graphic;
31   END_ENTITY; ...
32
33 END_SCHEMA; -- tk_widgetContainer
```

Cet exemple présente trois extraits de schémas utilisés pour la modélisation des concepts d'objets graphiques et de conteneurs d'objets graphiques classiquement mis en œuvre par les outils de construction d'interfaces homme-machine. Les objets graphiques élémentaires sont spécifiés par le schéma *coreWidget*. Le schéma *widgetContainer* décrit les entités conteneurs d'objets graphiques. Ce schéma référence *coreWidget*. En effet, un objet graphique élémentaire ne peut exister que dans le contexte d'un conteneur. Le schéma *tk_widgetContainer* utilise l'entité *container* du schéma *widgetContainer*, l'objectif étant de spécialiser cette entité pour l'adapter à la spécification des conteneurs mis en œuvre par un outil particulier de construction d'interfaces graphiques.

FIG. 3.2 – *La réutilisation entre chémas*

référencé par le schéma qui utilise ou référence :

– dans le cas de l'*utilisation* les entités spécifiées dans le schéma utilisé peuvent être exploitées

indépendamment, soit directement, soit par héritage, soit comme type d'un attribut; le statut des éléments (types, fonctions, contraintes ou entités) utilisés est identique à celui des éléments spécifiés dans le schéma qui utilise,

– dans le cas de la *référence*, les entités spécifiées dans le schéma référencé ne peuvent être exploitées que dans le contexte d'une autre entité, comme type d'un attribut.

L'*utilisation* est introduite pour faciliter la redéfinition et l'enrichissement des schémas. De ce fait, *l'utilisation* est souvent liée à l'héritage. La *référence* est plus particulièrement liée à l'association d'entités : les entités et les types référencés sont les types des attributs des entités construites par association.

3.2 Le langage EXPRESS-G

EXPRESS-G est un formalisme graphique utilisé pour la représentation d'un sous-ensemble des constructions du langage EXPRESS. Un schéma peut être spécifié en EXPRESS-G indépendamment d'une définition textuelle en EXPRESS. Cependant, la spécification des contraintes et du calcul des attributs dérivés est impossible en EXPRESS-G. Le langage comprend trois catégories de symboles graphiques :

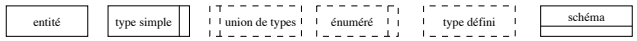
1. les *définitions* sont utilisées pour représenter les types simples, les types nommés, les types construits et la déclaration des schémas,
2. les *relations* sont utilisées pour représenter les relations entre *définitions*,
3. les *compositions* permettent aux schémas d'être représentés sur plusieurs pages.

Ces symboles peuvent être utilisés pour représenter les éléments constitutifs d'un schéma. Ces spécifications sont alors dites *de niveau entité*. Les relations de dépendance et de réutilisation entre schémas sont elles représentées par des spécifications dites *de niveau schéma*. La suite de cette partie présente les caractéristiques principales de ce langage. La spécification complète est donnée dans [ISO94b].

3.2.1 Les définitions

Une définition est représentée par une boite. Le nom du type ou du schéma représenté est indiqué dans la boite. Comme le montre la figure 3.3, le style de cette boite diffère suivant le type représenté, notamment par la nature du trait (pointillé ou plein).

Les définitions de fonctions, de procédures ou de contraintes ne peuvent pas être représentées en EXPRESS-G.



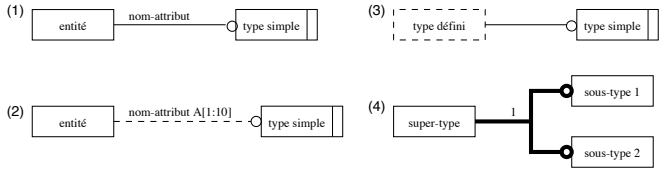
Pour un type simple, le trait est plein et le trait du coté droit est doublé. Pour un type construit, le trait est en pointillés. Un des traits de coté est doublé, à gauche pour une union de types et à droite pour un énuméré. Un type défini est représenté par une boite en pointillés. Un schéma est représenté par une boite à trait plein, coupé horizontalement. Une entité est représentée par une boite simple à traits pleins.

FIG. 3.3 – *La représentation des définitions en EXPRESS-G*

3.2.2 Les relations

Une relation est spécifiée par une ligne. Le style de la ligne varie suivant la nature de la relation. Une ligne discontinue est utilisée pour relier un attribut facultatif à son entité. Une ligne à trait épais est utilisée pour la relation d'héritage. Toutes les autres relations sont spécifiées par une trait continu simple. La direction de la relation est indiquée par un cercle. Par exemple, pour un trait reliant une entité à un de ses attributs, un cercle est dessiné du coté de la boite qui représente l'attribut. Pour la relation d'héritage, le cercle est spécifié du coté des sous-types.

Comme le montre la figure 3.4, une relation peut être annotée pour notamment préciser les cardinalités dans le cas d'aggrégations ou d'attributs inverses, pour qualifier la relation d'héritage ou pour indiquer le nom d'un attribut.



Voici quatre exemples de relations représentées en EXPRESS-G. (1) représente le cas d'un attribut de type simple relié à son entité. Le trait est annoté avec le nom de l'attribut. (2) représente un attribut, facultatif relié à son entité. L'annotation précise le nom de l'attribut, l'aggrégation utilisée (en l'occurrence A pour $ARRAY$) et les cardinalités. (3) représente la relation entre un type défini et son domaine. (5) représente une relation d'héritage entre un super-type et deux sous-types. L'annotation précise qu'il s'agit d'un héritage $ONEOF$.

FIG. 3.4 – *La représentation des relations en EXPRESS-G*

3.3 Vers une deuxième version du langage

Une première proposition d'évolution d'EXPRESS vers une deuxième version standard est décrite dans [ISO97]. Les évolutions remarquables sont principalement, la possibilité d'associer un comportement aux entités d'un schéma et une proposition de normalisation d'un schéma EXPRESS qui décrit les langages EXPRESS et EXPRESS-G.

3.3.1 La spécification du comportement

La spécification du langage introduit les concepts d'*action*, d'*événement* et de *réaction*. Un schéma ne consiste plus exclusivement en la spécification de données et de contraintes. La deuxième version d'EXPRESS considère un schéma comme la spécification d'un ensemble d'objets interréactifs.

Une action spécifie un ensemble de modifications apportées à la base de données. Une action est une suite d'instructions EXPRESS éventuellement associée à des pré ou post conditions. Une *action* peut créer, supprimer ou modifier les instances des entités d'un schéma.

Le comportement des entités d'un schéma s'exprime sous la forme d'*événements* et de *réactions* à ces *événements*. Un évènement peut être déclenché explicitement ou implicitement lorsque qu'une condition spécifiée dans la déclaration de l'évènement est vérifiée. On distingue les *événements* globaux des *événements* locaux :

- un *évènement* global est défini relativement à un schéma; une occurrence d'*événement* global est émise à toutes les instances de toutes les entités du schéma; du point de vue d'un schéma, la provenance d'un *événement* global est soit interne soit externe suivant que la description de son déclenchement (dans la spécification d'une action ou d'une réaction) appartienne ou non au schéma,
- un *événement* local est défini relativement à une entité; une occurrence d'*événement* local est émise par une instance et vers une ou plusieurs autres instances de l'entité; il s'agit alors d'un évènement interne; une occurrence d'*événement* local peut aussi être émise par une instance vers le schéma, il 's'agit alors d'un évènement externe.

Une *réaction* à un événement est définie relativement à une entité et consiste en la référence aux *événements* déclencheurs et en la spécification du comportement. Une *réaction* peut soit exécuter un ensemble d'instructions, soit émettre des événements.

3.3.2 Un schéma EXPRESS pour décrire EXPRESS

La mise en œuvre d'outils informatiques qui analysent et exploitent des schémas EXPRESS nécessite la consultation et la mise à jour d'une représentation interne des schémas analysés. Cette représentation interne consiste en des données décrites par un modèle de données communément appelé *méta-modèle*. Pour faciliter la mise en œuvre de ce méta-modèle et surtout favoriser une bonne interopérabilité entre outils, il est nécessaire de disposer d'une spécification commune du méta-modèle. Pour ces raisons, la prochaine version standard du manuel de référence du langage intègre la spécification d'un ensemble de schémas dont l'objectif est de décrire les données nécessaires au stockage et à l'échange des informations contenues dans un schéma EXPRESS.

Chapitre 4

Mise en œuvre des schémas EXPRESS

La norme STEP définit non seulement une méthode de description essentiellement basée sur le langage EXPRESS mais aussi des méthodes de mise en œuvre pour l'exploitation informatique des schémas EXPRESS. Les objectifs principaux de ces méthodes de mise en œuvre sont de décrire les protocoles d'échange et de partage des instances des schémas entre systèmes hétérogènes. L'échange de données s'effectue par fichiers d'échanges standards et le partage de l'information par une interface logicielle standard dite *Interface Standard d'Accès aux Données* ou *SDAI*.

Ces deux méthodes de mise en œuvre ne sont à l'heure actuelle décrites que pour la première version standard du langage EXPRESS. La suite de ce chapitre est donc uniquement corrélée à cette version du langage.

4.1 L'échange des données

Pour la plupart des applications informatiques, une des méthodes les plus couramment employées pour interagir entre systèmes est l'échange d'information par fichier. Cette méthode est en effet souvent très satisfaisante car les données sont persistantes et peuvent être manipulées par des systèmes et langages hétérogènes.

Principalement deux approches sont exploitées pour la mise en œuvre de l'interopérativité entre deux systèmes [Weg96, Bou95a] : par une liaison spécifique ou par une interface standardisée :

- une interface spécifique est directement adaptée au besoins des systèmes communicants; pour n systèmes communicants, $n * n$ interfaces doivent être développées,
- une interface standardisée est basée sur la réutilisation d'un protocole normalisé; pour n systèmes communicants, $2 * n$ mises en œuvre du protocole normalisé sont nécessaires.

Bien que permettant la mise en œuvre de composants d'échanges très efficaces en terme de rapidité d'exécution, la première solution est souvent problématique : la sémantique des données stockées dans les fichiers est dans la plupart des cas implicite et spécifiquement reconnu par les applications. Les algorithmes de lecture/écriture des données sont difficilement réutilisés et maintenus.

La norme STEP met en œuvre la seconde approche : un format neutre et standard de représentation des informations est spécifié, le langage EXPRESS est utilisé pour décrire les informations transportées. Chaque fichier d'échange constitue une instanciation d'un schéma EXPRESS. Les avantages principaux de cette méthode sont de favoriser l'intelligibilité des informations contenues

dans les fichiers, de diminuer les problèmes de maintenance des systèmes communicants et de rendre la méthode d'échange indépendante des spécificités des applications et des systèmes.

4.1.1 Les processus d'échange

Deux systèmes communicants mettent chacun en œuvre un composant d'import/export de données. Le schéma EXPRESS qui décrit les données échangées est partagé par les deux systèmes communicants. Il représente un consensus sur la définition des données échangées.

Etant donné le schéma EXPRESS, l'encodage des données est explicitement décrit par le document standard [ISO94c]. De plus, les contraintes décrites dans le schéma EXPRESS peuvent être vérifiées par les composants d'import/export.

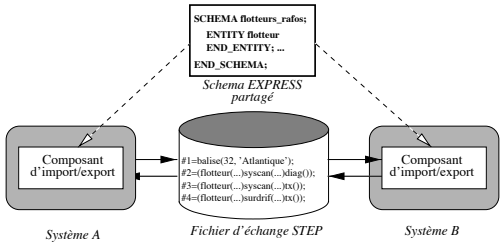


FIG. 4.1 – *Les processus d'échange par fichier STEP*

4.1.2 Le format des fichiers STEP

La spécification du format d'échange STEP est le standard ISO 10303-21 [ISO94c]. Un fichier STEP contient les instances d'un schéma EXPRESS auquel il se réfère explicitement. Un fichier est structuré en deux sections. La première section identifie l'origine du fichier et du schéma EXPRESS. La seconde section contient la liste des instances.

Chaque instance possède un identifiant numérique entier unique dans le contexte d'un fichier, le nom de l'entité correspondante et une liste de valeurs d'attributs. Seules les valeurs des attributs explicites sont échangées. Les valeurs des attributs dérivés, les contraintes et les algorithmes décrits dans le schéma EXPRESS ne sont pas contenus dans un fichier d'échange STEP. La table 4.1 présente des valeurs valides par type d'attribut. La liste de valeurs d'attributs est construite dans l'ordre défini par celui des attributs de l'entité EXPRESS correspondante. Le type des valeurs est implicitement exprimé par le format textuel de la valeur. Dans le cas d'une valeur dont l'attribut correspondant est une union de types (en EXPRESS un type *select*), il est possible de qualifier explicitement le type de la valeur. Ceci permet d'assurer dans tous les cas un typage fort.

La syntaxe du langage d'instanciation prévoit deux algorithmes pour la création de la représentation d'une instance dont l'entité correspondante est un sous-type, la *correspondance interne* et la *correspondance externe* :

– la correspondance interne est utilisée s'il n'y a qu'une seule possibilité pour l'ordre des valeurs d'attributs; dans ce cas, la liste des valeurs de l'entité sous-type est concaténée à celle de ses super-types; l'ordre est donné par la relation d'héritage [ISO94c].

– la correspondance externe est utilisée s'il existe plusieurs possibilités pour l'ordre d'insertion des valeurs d'attributs; c'est le cas lorsque la hiérarchie d'entités comprend un ou des super-types spécifiant leurs sous-types avec les opérateurs *AND* ou *ANDOR*; la liste des valeurs est alors produite sous la forme d'une liste principale contenant des sous-listes de valeurs; chaque sous-liste est associée à un nom d'entité et représente une correspondance interne pour cette entité; l'ordre d'insertion des sous-listes dans la liste principale peut varier d'une instance à l'autre. Une telle entité est dite une *entité complexe*.

Types d'attribut	Valeurs possibles
Integer	4523
Real	5.456 ou 2.3E2
String	'des caracteres'
Binary	"12A5E4"
Boolean	.T. ou .F.
Logical	.T., .F. ou .U.

Types d'attribut	Valeurs possibles
Enumeration	.deg_celsius.
Aggrégation	('ab','dhf')
Identifiant d'instance	#945823
Valeur qualifiée	*un_type*(6)
Valeur inexistante	$
Attribut redéfini	*

TAB. 4.1 – *Des exemples de valeurs d'attributs contenues dans un fichier STEP*

```
1 #1=(flotteur(32) bathy((56,345,2,8)) diag());
2 #2=(flotteur(22) syscan(7) diag());
3 #3=(flotteur(22) syscan(4) diag() tx());
4 #4=(flotteur(32) bathy((76,45,1,16)) tx());
5 #5=(flotteur(26) surdrif((14,$,$,$)) tx());
```

Cette liste contient des instances des entités présentées dans l'exemple 3.1. Etant donné la relation d'héritage entre l'entité *flotteur* et ses sous-types, la correspondance externe est utilisée pour l'instanciation des flotteurs. Par exemple, l'instance #1 est une combinaison entre les valeurs d'attributs de *flotteur*, *bathy* et *diag*. Il s'agit d'un flotteur *bathy&diag*.

FIG. 4.2 – *Instanciation de deux entités complexes*

Une instance peut constituer le contexte de définition d'instances dépendantes. La durée de vie d'une instance définit dans un contexte est dépendante de la durée de vie de l'instance qui définit le contexte. De plus, une instance déclarée dans un contexte n'est visible à l'extérieur du contexte que si elle est explicitement exportée.

La possibilité de créer des contextes peut être utilisée pour traduire des relations de dépendance entre entités telle que celle de la réutilisation d'entités entre schéma par la référence (cf. le chapitre 3.1.6 page 7). La norme [ISO94c] précise que la création de contexte est à la discrétion de l'outil qui crée le fichier d'échange. Il n'y a donc pas de règle de correspondance explicite entre la description des entités et la création de contextes. En pratique, cette possibilité est peu utilisée et devrait disparaître lors des prochaines révisions de la norme STEP.

4.2 Une interface logicielle pour le partage des données

L'objectif de la norme STEP est de pouvoir décrire et exploiter les données de produits pendant tout le cycle de vie. A chaque phase du cycle de vie correspond un ensemble de données, manipulées

```
1 #1=&scope
2 #2=button('ok', 150, 5, 30, 15);
3 #3=entry('e112', 100, 5, 60, 15);
4 #4=frame('f156', 2, 2, 70, 50);
5 endscope /#2,#3/
6 tk_container('ask_path', (#2,#3,#4), .tk_top.);
7 #5=tk_inputs((#2,#3));
```

#1 est une instance de l'entité *tk_container*, présentée dans l'exemple 3.2. Elle contient un contexte dans lequel sont déclarées une instance de l'entité *button*, une instance de l'entité *entry* et une instance de l'entité *frame* qui sont des sous-types de *graphic*. Les instances #2 et #3 sont explicitement exportées. Elles peuvent ainsi être référencées à l'extérieur du contexte, ce qui est le cas dans l'instance #5.

FIG. 4.3 – *Une instance définissant un contexte*

par des applications et/ou des composants logiciels spécifiques. Les informations mises à jour au cours d'une phase du cycle de vie peuvent être réutilisées directement ou indirectement au cours d'une autre étape. Il existe essentiellement deux protocoles pour le passage des informations entre différentes étapes du cycle de vie [Che94] :

- selon le protocole séquentiel, les informations circulent d'une étape à l'étape suivante de façon unidirectionnelle,
- selon le protocole de partage, toutes les étapes du cycle de vie sont gérées de façon concurrente par des applications ou composants logiciels qui partagent une base de données commune.

Le second protocole est préférable. En effet, le partage de l'information favorise l'adaptation et l'interopérabilité des applications. La principale difficulté à une telle organisation est de disposer d'une interface standard pour mettre en œuvre le partage de l'information au sein d'une base de données intégrée pour toutes les phases du cycle de vie [Che94].

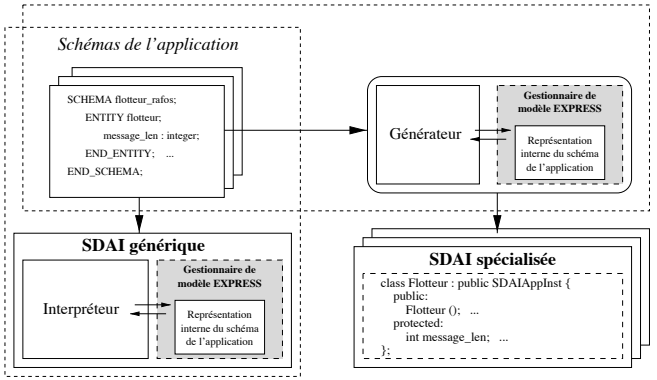
La norme STEP prévoit un protocole de partage de données entre les différentes phases du cycle de vie. Ce protocole est décrit par la partie 22 [ISO94d] qui spécifie une interface et le protocole d'accès au données stockées dans une base de données. Cette interface logicielle est communément appelée SDAI[1]. Cette spécification est donnée indépendamment des langages et des systèmes : le langage EXPRESS est utilisé pour la description des entités structurelles de la SDAI et le comportement de ces entités est décrit informellement en langue anglaise.

Les mises en œuvre de cette spécification dans des langages de programmation particuliers tel que le C++ [ISO95a], C [ISO95b] ou IDL [ISO95c], sont aussi décrites. Cependant, la SDAI est dite neutre car, quelque-soit le langage et le système physique de stockage, les données manipulées par les applications sont accédées au travers de la SDAI, telle qu'elles sont décrites par les schémas EXPRESS qui spécifient ces données. La SDAI définit une interface fonctionnelle entre les applications et l'environnement de stockage des données de sorte qu'elle permet l'indépendance des applications vis à vis des spécificités liées au stockage des données.

4.2.1 La mise en œuvre

La norme STEP prévoit deux possibilités pour la mise en œuvre d'une SDAI. Elle peut être soit spécialisée, soit indépendante du schéma EXPRESS qui décrit les données manipulées (ou schéma

1. Standard Data Acces Interface



La mise en œuvre d'une SDAI générique contient une représentation interne du schéma EXPRESS source. Les fonctions de la SDAI utilisent les informations contenues dans cette représentation interne. La mise en œuvre d'une SDAI spécialisée est générée à partir du schéma source. Une représentation interne du schéma EXPRESS source est, dans ce cas, exploitée par le générateur qui produit la SDAI. Par contre, les fonctions de la SDAI n'exploitent pas cette représentation.

FIG. 4.4 – *Représentation d'une SDAI générique et d'une SDAI spécialisée*

```
1 // Construction d'une instance de l'entité "syscan"
2 SDAIAppInstanceH appInstH;
3 appInstH = sdaiCreateInstanceBN(modelH,"syscan");
4
5 // Affectation de la valeur 5 a l'attribut "syscan.message_len"
6 const SDAIEntityH itypeH = appInstH->getInstanceType();
7 const SDAIAttrH atypeH = itypeH->getAttrType("message_len");
8 appInstH->PutAttr("message_len", atypeH, 5);
```

FIG. 4.5 – *Création d'une instance avec une SDAI générique*

EXPRESS source) :

- une mise en œuvre indépendante est dite SDAI générique (ou tardive), car elle permet l'accès aux données, quelque-soit le schéma EXPRESS source; comme le montre l'exemple 4.5, la référence aux constructions particulières du schéma EXPRESS source s'effectue par passage de paramètres aux fonctions de la SDAI,
- une mise en œuvre spécialisée (ou précoce) est dépendante du schéma EXPRESS source et peut être automatiquement produite à partir du schéma EXPRESS source; comme le montre l'exemple 4.6, la référence aux constructions particulières du schéma EXPRESS source est explicitement spécifiée par le nom des fonctions.

```
1   // Construction d'une instance de l'entité "syscan"
2   SDAISyscanH syscanH = new SDAISyscan(modelH);
3   // Affectation de la valeur 5 a l'attribut "syscan.message_len"
4   syscanH->set_message_len(5);
```

FIG. 4.6 – *Création d'une instance avec une SDAI spécialisée*

```
1  int main(int argc, char * argv[]) {
2    /* Ouverture d'une session, d'un repository et d'un modele */
3    SdaiSession session = sdaiOpenSession ();
4    SdaiRep repo = sdaiOpenRepositoryBN (session, "p21_step_file");
5    SdaiModel model = sdaiAccessModelBN (repo, "vitac", sdaiRW);
6    /* Récupération des balises */
7    SdaiSet extent = sdaiGetEntityExtentBN (model, "balise");
8    SdaitIterator itor = sdaiCreateIterator (extent);
9    while (sdaiNext(itor)) { /* Pour toutes les balises */
10     SdaiString aBaliseName = NULL;
11     SdaiInstance aBalise;
12     sdaiGetAggrByIterator (itor, sdaiINSTANCE, &aBalise);
13     sdaiGetAttrBN (aBalise, "name", sdaiSTRING, &aBaliseName);
14     fprintf (stdout, "%s\n", aBaliseName);
15   }
16   sdaiCloseSession (session);
17 }
```

FIG. 4.7 – *Traitement sur un ensemble d'instances avec une SDAI générique en C*

4.2.2 La gestion des instances

Du point de vue de l'application qui utilise les services d'une SDAI, les instances manipulées sont accédées et créées dans un *modèle* décrit par l'entité *sdai_modèle* de la partie 22. A un *sdai_modèle* correspond un schéma EXPRESS qui décrit les données manipulées.

Un *sdai_modèle* est créé dans un *repository*. Un *repository* établit une correspondance entre des *sdai_modèles* et une ressource physique de stockage des données tels qu'une base de données, un fichier ou encore la mémoire.

Un *sdai_modèle* peut partager des données avec d'autres *sdai_modèles* et ce, quelque soient leurs *repository* : une instance peut référencer d'autres instances appartenant à des *sdai_modèles* différents et une instance peut se dupliquer d'un *sdai_modèle* à un autre *sdai_modèle*. Une application peut ainsi utiliser les services de plusieurs bases de données au cours d'une même session. Les exemples 4.7 et 4.8 montrent la lecture d'un ensemble d'instances stockées dans un *sdai_modèle*.

4.2.3 La description des instances

La description des instances manipulées par une SDAI est contenue dans un dictionnaire décrit par le schéma *SDAI_dictionary_schema*. Son rôle est de décrire la sémantique statique d'un schéma

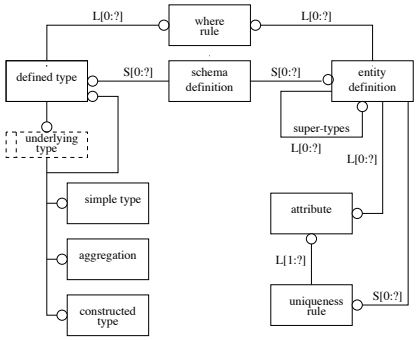

```
1  #include <flotteur_rafos.h>
2
3  int main(int argc, char * argv[]) {
4    // Ouverture d'une session, d'un repository et d'un modele
5    SessionH sessionH = Session:: OpenSession();
6    RepoH repoH = sessionH->OpenRepoRW("p21_step_file");
7    ModelH modelH = repoH->getModelRW("vitac");
8    // Récupération des balises
9    SDAIAGGRH(Set,Balise) balisesH;
10   balisesH = Balise::GetEntityExtents(modelH);
11   Iterator<Balise> baliseItor(*balisesH);
12   while (baliseItor.Next()) { // Pour toutes les balises
13     Balise * aBaliseH = baliseItor.GetCurrentMember();
14     cout « ( const char *) aBaliseH->getName() « endl;
15   }
16   sessionH->close();
17 }
```

FIG. 4.8 – *Traitement sur un ensemble d'instances avec une SDAI spécialisée en C++*

EXPRESS. Il contient la définition des concepts de *schéma*, de *type*, d'*entité* et d'*attribut* EXPRESS. Sa mise en œuvre permet à la SDAI de disposer de la description des objets qu'elle manipule. Ainsi, toute instance manipulée par une SDAI peut accéder à sa propre description et indirectement à la description du schéma EXPRESS qui la spécifie. La figure 4.9 montre une représentation simplifiée en EXPRESS-G du schéma *SDAI_dictionary_schema*.

4.2.4 L'architecture d'une application

Comme le montre la figure 4.10, la SDAI est classiquement exploitée par une application comme une couche horizontale dédiée à la gestion des lectures et des écritures des données dans ou depuis un ou plusieurs *repository*. Une application qui utilise les services de la SDAI accède aux données stockées dans les *repository* ouverts par la session courante. Les données de l'application peuvent être lues ou modifiées alors que les données du dictionnaire ne sont accessibles qu'en lecture. Les données de session sont mises à jour par les fonctions de la SDAI pour le maintien de sa cohérence interne.



Un schéma est associé à un identifiant et contient la spécification de *types définis* et d'*entités*:

- Un *type défini* se caractérise par un nom associé à un type sous-jacent ou *domaine* et à une liste de contraintes locales. Le domaine du type est soit un *type simple* (*integer*, *real*, *string*, *binary*, *logical* ou *boolean*), soit un type *aggregation* (*list*, *set*, *bag* ou *array*), soit un type *construit* (*enumeration* ou *select*), ou soit un autre *type défini*.
- Une *entité* se caractérise par un nom, la liste de ses contraintes locales (*where rule*), la liste de ses super-types, la liste de ses attributs et la liste des règles d'unicité (*uniqueness rule*) appliquées aux attributs de l'entité ou de ses super-types.

FIG. 4.9 – *Représentation en EXPRESS-G du schéma* SDAI_dictionary_schema

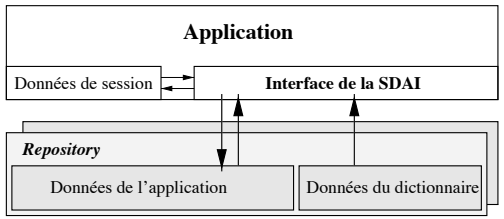


FIG. 4.10 – *L'architecture d'une application qui utilise la SDAI*

Chapitre 5

L'environnement de modélisation

La norme STEP constitue principalement un environnement comprenant les méthodes et les outils pour la spécification et la mise en œuvre de schémas de données. L'objectif est de fournir des schémas de données standards suffisamment consensuels pour recouvrir le besoin de tous les utilisateurs potentiels. La description d'un schéma de données standard est un *protocole d'application*. Chaque *protocole d'application* constitue une partie de la norme et comprend des spécifications formelles dans le langage EXPRESS et informelles pour l'explication du sens et le rôle des modèles de données.

L'environnement de modélisation de STEP autorise la spécification des données par des modèles fonctionnels et des schémas conceptuels [Dan93]. Du point de vue fonctionnel, les données sont décrites par rapport aux fonctionnalités qui les exploitent. Les définitions de données sont ainsi exprimées dans leur contexte d'utilisation. Les notations utilisées à ce niveau sont semi-formelles telles que IDEF0 [Fed93a]. La signification standard des données nécessaires aux fonctionnalités est définie au niveau conceptuel sous la forme de schémas EXPRESS [ISO94b].

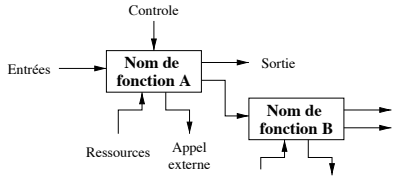
5.1 Le point de vue fonctionnel

La représentation de la signification de types de données nécessite la compréhension de leurs contextes d'utilisation. La définition de ce contexte comprend la définition des fonctions ou activités et des flots de données exploitées ou produites par ces fonctions.

L'information exploitée par chaque activité constitue un *groupe de données fonctionnel* qui identifie les données qui sont transformées, les données qui sont produites par ces transformations et les autres données nécessaires à l'exécution des fonctions. Le formalisme utilisé est le langage graphique IDEF0 [Fed93a]. Un exemple de schéma IDEF0 est présenté par la figure 5.1.

5.2 Le point de vue conceptuel

Un protocole d'application représente un schéma de données dont la signification doit être consensuelle et correctement comprise par tout utilisateur potentiel. Ceci implique un accord sur la signification des données, sur l'utilisation qui est faite du langage de modélisation et sur les schémas conceptuels construits pour expliciter la signification des données avec le langage de modélisation choisi [Dan93].



Cet exemple montre deux représentations de fonctions en IDEF0. Chaque représentation comprend le nom de la fonction encadrée. Les données en entrée sont transformées alors que les données en sortie sont produites par la fonction. Les données de contrôle sont nécessaires à l'exécution de la fonction. Les ressources représentent des outils utilisés et les appels externes sont des références à des fonctions décrites dans un autre modèle fonctionnel.

FIG. 5.1 – *Une présentation du formalisme IDEF0*

Les éléments conceptuels définis dans STEP pour la description de la signification des informations sont spécifiés dans un *schéma conceptuel*. Un *schéma conceptuel* est un réseau de *concepts* et de *relations* entre ces *concepts*. L'ensemble des éléments conceptuels utilisés dans un *schéma conceptuel* comprend le *concept*, la *relation* et la *propriété* :

- *un concept* est une catégorie d'objets, d'événements ou d'états qui peut être clairement spécifiée par une identifiant unique; un *concept* peut être soit indépendant soit dépendant d'un autre *concept*; un concept peut être construit par la *composition* d'autres concepts,
- *une relation* est représentée dans un *concept* par un lien vers un autre *concept*; le *rôle* décrit les occurrences d'un concept qui participe à une *relation*; une *association* est un *concept* présentant au moins deux *relations*,
- *une propriété* représente un aspect caractéristique d'un *concept* décrit qualitativement et quantitativement, *un domaine* est la spécification des valeurs autorisées pour une *propriété*; des *contraintes* peuvent être utilisées pour cette spécification,

Le langage de conception utilisé par STEP pour représenter les éléments conceptuels est le langage EXPRESS et son sous-ensemble graphique EXPRESS-G. Ces deux langages sont décrits par une norme ISO [ISO94b].

La correspondance entre les éléments conceptuels et les constructions d'EXPRESS est établie pour spécifier l'utilisation de ce langage pour la représentation des éléments conceptuels. L'application stricte de cette correspondance est fondamentale dans le cadre de la standardisation de schémas de données. Cette correspondance est décrite formellement par la norme, en EXPRESS-G dans [Dan93] et en EXPRESS dans [Spi94]. La table 5.1 montre les éléments conceptuels pour lesquels une construction syntaxique du langage EXPRESS permet une correspondance explicite.

Un *schéma conceptuel* est représenté par un ou plusieurs *schémas* EXPRESS intégrés par la déclaration de leur interface.

Lorsque la relation inter-schémas *USE FROM* est utilisée, l'intégration consiste en la copie des concepts dans le schéma qui utilise. Avant la copie, le schéma qui utilise est dit en *forme courte* alors qu'après la copie, le schéma est dit en *forme longue*. Cette copie peut être automatisée. Elle

Eléments conceptuels	EXPRESS
schéma conceptuel	*schema*
concept et association	*entity*
composition de concepts	héritage ou type *select*
rôle et propriété	attribut d'une *entity*
contraintes	contraintes locales ou globales

TAB. 5.1 – *Correspondance entre éléments conceptuels et EXPRESS*

est spécifiée par l'algorithme *ShortToLongForm*. Après l'exécution de cet algorithme il ne subsiste aucune relation *USE FROM*.

Lorsque la relation inter-schémas *REFERENCE FROM* est utilisée, les schémas qui participent à la relation ne sont pas modifiés par l'intégration.

Une *entity* EXPRESS est utilisée pour la représentation d'un *concept*. Un attribut d'une *entity* peut être utilisé soit pour la spécification d'une *propriété*, soit pour la spécification d'un *rôle* d'une *relation*.

Une *propriété* est implicitement spécifiée par un attribut dont le type est un type simple, ou une aggrégation d'éléments dont le type est un type simple.

Un *rôle* est implicitement déclaré par un attribut dont le type est une *entity* ou une aggrégation d'éléments dont le type est une *entity*.

Un attribut peut représenter à la fois un *rôle* et une *propriété* si le type de l'attribut est un type *select* dont la liste des types référence à la fois un type simple et un type *entity*.

L'extrait du schéma EXPRESS décrit dans [Spi94] et présenté dans la figure 5.2 montre la spécification formelle de la définition d'un *attribut* EXPRESS.

5.3 Les modèles mis en œuvre

La conception d'un protocole d'application standard procède de la mise en œuvre de plusieurs catégories de schémas : les *ressources intégrées*, les *constructions interprétées* par domaine d'application et les schémas constitutifs du protocole d'application.

Les *ressources intégrées* et les *constructions interprétées* sont des parties normalisées de STEP. Ils sont le résultat d'un processus d'intégration de plusieurs schémas reflétant le besoin dans différents domaines d'application. Ils constituent la source fondamentale de réutilisation et d'intégration des modèles de données standard.

L'exploitation des *ressources intégrées* et des *constructions interprétées* est un exemple de réutilisation au niveau conceptuel. Leur mise en œuvre par STEP permet de minimiser le problème du recouvrement possible entre plusieurs modèles standards et favorise l'intelligibilité, la maintenabilité et l'homogénéité des différents *protocoles d'application* de la norme.

```
1  ENTITY attribute ABSTRACT SUPERTYPE
2      OF (ONEOF((ONEOF(explicit_attribute, derived_attribute) AND (relationship
           ANDOR property)),
3                 (inverse_attribute AND relationship)));
4      attribute_name : name;
5      defining_entity : EXPRESS_entity;
6      representation : base_type;
7  END_ENTITY;
8
9  ENTITY explicit_attribute SUBTYPE OF (attribute); END_ENTITY;
10
11 ENTITY inverse_attribute SUBTYPE OF (attribute); END_ENTITY;
12
13 ENTITY derived_attribute SUBTYPE OF (attribute);
14       derivation : expression;
15 END_ENTITY;
16
17 ENTITY property SUBTYPE OF (attribute);
18     WHERE non_entity_type_in_representation : not_entity_in_base_type(representation);
19 END_ENTITY;
20
21 ENTITY relationship SUBTYPE OF (attribute);
22     WHERE entity_type_in_representation : entity_in_base_type(representation);
23 END_ENTITY;
24
25 FUNCTION entity_in_base_type (input : base_type) : boolean;
26     if 'ATTRIBUTE.EXPRESS_ENTITY' in typeof(input) then
27       return (true);
28     else if 'ATTRIBUTE.EXPRESS_TYPE' in typeof(input) then
29       return (entity_in_base_type(input\express_type.based_on));
30     else if ('ATTRIBUTE.AGGREGATION_TYPE' in typeof(input) then
31       return (entity_in_base_type(input\aggregation_type.element_type));
32     else ...
33     end_if; end_if; end_if;
34     return (false);
35 END_FUNCTION; -- entity_in_base_type
```

Dans ce schéma, le *rôle* est représenté par l'entité *relationship*. Un attribut est soit un attribut explicite, soit un attribut dérivé, soit un attribut inverse. Un attribut explicite ou dérivé est aussi soit une propriété, soit un rôle, soit les deux. Un attribut inverse est aussi obligatoirement un rôle.

FIG. 5.2 – *Une spécification formelle d'un* attribut *EXPRESS*

5.3.1 Les ressources intégrées

Les ressources intégrées ou IR[1] sont des schémas EXPRESS constituant des librairies de descriptions de concepts réutilisables pour plusieurs applications. STEP distingue les *ressources intégrées totalement génériques* des *ressources génériques par application*.

Les *ressources intégrées totalement génériques* sont des descriptions de concepts indépendants de toute catégorie d'applications alors que les *ressources génériques par application* sont des descriptions de concepts classées par catégorie d'application.

Les entités EXPRESS décrites dans les *ressources génériques par application* complètent et spécialisent les entités décrites dans les *ressources intégrées totalement génériques* pour améliorer leur adéquation à des catégories d'applications. Le grain de réutilisation permi par les IR est principalement celui de l'entité.

1. Integrated Resources

5.3.2 Les constructions interprétées

Les *constructions interprétées* ou AIC[2] sont spécifiées par catégories d'applications et sont des schémas EXPRESS qui décrivent des utilisations couramment mises en œuvre des mêmes ressources intégrées. L'utilisation des AIC permet d'éviter le recouvrement entre les *protocoles d'application* d'une même catégorie d'applications. Le grain de réutilisation est moins fin que celui permis par les IR puisqu'il s'agit de réutiliser des ensembles d'entités inter-dépendants, cette inter-dépendance étant spécifique d'une catégorie d'applications.

5.3.3 Les constituants du protocole d'application

Un AP est principalement développé sur la base des documents suivants : un modèle fonctionnel, appelé *modèle d'activité*, un *modèle de référence*, un *modèle interprété* et une table de correspondances entre les informations contenues dans le *modèle de référence* et les entités équivalentes spécifiées dans le *modèle interprété*.

Le modèle d'activités ou AAM[3] décrit le contexte d'utilisation des données d'un point de vue fonctionnel. Ce modèle contient la spécification des processus de transformations de données et des flots de données en IDEF0 [Fed93a].

Le modèle de référence ou ARM[4] contient la description semi-formelle ou formelle et la documentation des *groupes de données fonctionnels*. L'ensemble de ces descriptions compose la spécification des données qui doivent être échangées ou partagées. Pour la spécification d'un ARM, trois langages de modélisation sont autorisés : EXPRESS, IDEF1X [Fed93b] et NIAM [NH89]. La terminologie employée est celle du domaine du protocole d'application développé.

Le modèle interprété ou AIM[5] constitue le principal résultat du processus de mise en œuvre d'un protocole d'application. Il procède d'une intégration entre l'ARM, les *ressources intégrées* et les *constructions interprétées*. Un AIM est spécifié avec le langage EXPRESS. Il constitue le schéma de référence normalisé pour les échanges ou le partage des données relatif à un domaine d'application.

La table de correspondance établit formellement l'origine des composants d'un AIM. Chaque entité et attribut d'entité de l'AIM est spécifié dans cette table avec son correspondant de l'ARM. Cette table de correspondance conserve la trace du processus d'intégration dont l'AIM est issu.

2. Application Interpreted Constructs
3. Application Activity Model
4. Application Reference Model
5. Application Interpreted Model

Chapitre 6

Le processus de construction d'un protocole d'application

Un protocole d'application réalise le lien entre un besoin exprimé par des utilisateurs d'une catégorie d'applications et les mises en œuvre informatiques des systèmes d'information. L'utilisateur, considéré comme l'expert du domaine, est le principal intervenant au début du processus de développement d'un AP. L'utilisateur établit les exigences et le vocabulaire liés au domaine. Les intervenants issus des groupes de travail de STEP interviennent ensuite pour établir les modèles conceptuels cohérents d'un point de vue définition d'un système d'information.

Le processus de développement d'un AP commence par la spécification de l'AAM qui définit les activités et les flots de données caractéristiques du domaine d'applications. Sur la base des groupes de données fonctionnels mis en évidence dans l'AAM, l'ARM est développé pour décrire plus précisément les données et les contraintes appliquées aux données. L'AIM est ensuite construit par une intégration entre l'ARM, les ressources génériques et les constructions interprétées.

6.1 La spécification de l'AAM et de l'ARM

La première partie du développement d'un AP consiste en la spécification d'un cahier des charges en langue anglaise. Ce cahier des charges constitue une description informelle pour laquelle une connaissance approfondie du métier est fondamentale. Une première classification des concepts décrits est effectuée par la mise en évidence des *groupes de données fonctionnels*. Les processus de traitement de l'information et les groupes de données fonctionnelles spécifiques au domaine sont ensuite formalisés tout d'abord en IDEF0 dans l'AAM puis en IDEF1X, NIAM ou EXPRESS dans l'ARM.

6.2 La spécification de l'AIM

Comme le montre la figure 6.1 le développement de l'AIM procède d'une réutilisation des *ressources intégrées* et des *constructions interprétées* de STEP pour répondre aux exigences des utilisateurs formulées dans l'ARM. La spécification de l'AIM s'accompagne de l'écriture d'une table de correspondance entre les entités de l'ARM et les ressources et constructions utilisées pour les représenter dans l'AIM. Les constructions des AIC utilisées sont intégrées sans modification alors que les IR peuvent être intégrées par spécification de sous-types, ajout de nouvelles contraintes et

de nouvelles relations. L'intégration des IR est considérée par STEP comme un processus d'interprétation.

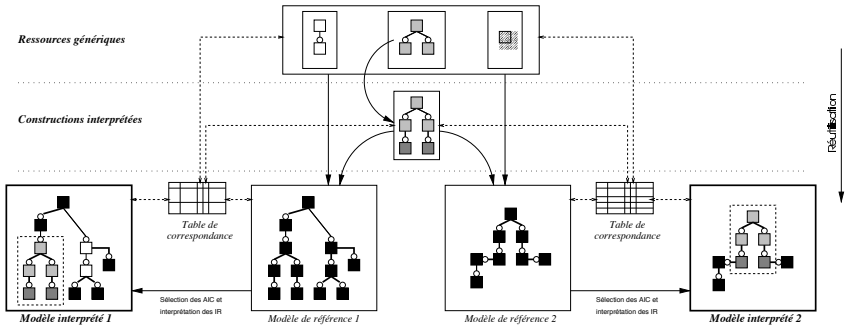


FIG. 6.1 – *La réutilisation de ressources conceptuelles pour le développement d'un AP*

6.2.1 Les processus d'intégration et d'interprétation

Comme le montre la figure 6.2, la conception d'un AIM intervient après une analyse informelle de l'AAM et le développement du modèle de l'ARM.

La première activité est un processus d'intégration des AIC. L'objectif est de sélectionner parmi les AIC disponibles, les constructions adéquates pour représenter le besoin de l'application.

La deuxième activité consiste en l'analyse de l'ensemble des concepts de l'ARM non intégrés lors de la première activité. L'objectif est de définir de nouvelles AIC. Une nouvelle AIC est définie si elle représente un besoin conceptuel réutilisable en l'état dans plusieurs AIM.

La troisième activité est le processus d'interprétation des IR. Elle consiste en l'analyse de l'ensemble des concepts de l'ARM non intégrés lors de la deuxième activité et en l'interprétation des IR pour la représentation de ces concepts.

L'interprétation s'effectue par une adaptation des entités spécifiées dans les IR. Cette adaptation peut consister en :

- la création de sous-types spécifiques à l'application; un sous-type peut contenir de nouvelles propriétés ou de nouveaux rôles, des attributs hérités peuvent être redéfinis en attributs dérivés,
- la spécification de nouvelles contraintes locales ou globales pour limiter le domaine de validité des valeurs d'attribut,
- la spécification de contraintes globales pour le contrôle de règles d'intégrité de références.

L'AIM est spécifié pendant la quatrième activité tout d'abord sous la forme d'un schéma dit au *format court* qui référence les schémas contenant les AIC intégrés et les IR utilisés. Ce schéma au *format court* ne contient donc dans un premier temps que ce qui est spécifique à l'application. Les AIC et les IR sont ensuite copiés dans ce schéma pour la spécification d'une version de l'AIM dite

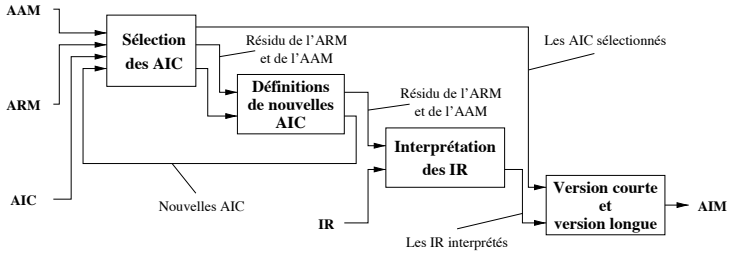


FIG. 6.2 – *Représentation du processus d'interprétation d'un AIM en IDEF0*

au *format long*. Le passage de la forme courte à la forme longue peut être effectué automatiquement par la mise en œuvre de l'algorithme *ShortToLongForm* [Pal95].

Chapitre 7

La validation d'un protocole d'application

La spécification d'un protocole d'application est une activité d'analyse et de conception très complexe et très peu automatisée. Les possibilités d'erreur sont donc nombreuses. Ce problème est traité par STEP. La norme définit les procédures pour la vérification et la validation des modèles. Ces procédures se constituent des méthodes de test de conformité spécifiées dans les parties 31 à 34 de la norme [ISO92].

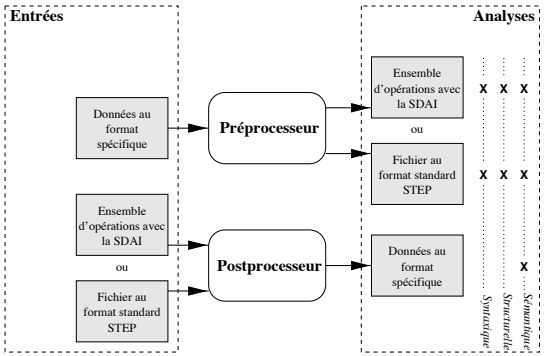
Les tests de conformité font partie intégrante d'un protocole d'application et doivent être définis dès le début de sa conception. Chaque constituant d'un protocole d'application doit être validé par des experts différents de ceux qui l'ont conçu. Les experts du domaine, de STEP et de la mise en œuvre des systèmes d'information interviennent au cours de la validation.

La procédure de validation d'un protocole d'application est basée sur la méthode des tests dits *abstraits* car indépendants de la mise en œuvre de ces tests. Le développement des tests abstraits est une tâche complexe qui n'est pas spécifiée strictement. La définition des tests à mettre en œuvre est spécifique à chaque protocole d'application. La norme fournit les guides qui indiquent l'approche à mettre en œuvre et non la définition exhaustive des tests [PK96].

Le principe d'un test abstrait est de spécifier les moyens par lesquels un protocole d'application et plus particulièrement l'AIM, doit être testé. Un test est considéré comme une mise en situation d'exploitation d'un AIM.

La base d'une série de tests abstraits consiste en la description d'une architecture comprenant un préprocesseur et un postprocesseur, en la description des données à traiter et en la spécification des résultats attendus après le traitement des données.

Le préprocesseur et le postprocesseur traitent des données encodées dans un format spécifique à la série de tests. Le format spécifique doit être décrit. Il peut s'agir par exemple d'un format textuel que l'utilisateur peut directement interpréter. Comme le montre la figure 7.1, les données encodées dans ce format sont consommées par le préprocesseur qui doit produire les données correspondantes au format STEP ou bien produire les appels fonctionnels de la SDAI nécessaires à l'instanciation de l'AIM. Le postprocesseur doit pouvoir soit lire les données au format STEP soit exécuter les opérations de la SDAI et encoder les données ainsi consommées au format spécifique.



Une série de tests abstraits est traitée par une environnement comprenant un préprocesseur et un postprocesseur. Le préprocesseur consomme des données dans un format spécifique à la série de tests et produit un résultat analysé syntaxiquement, structurellement et sémantiquement :

- l'analyse syntaxique consiste à vérifier le produit suivant les règles de représentation des données spécifiées par la partie 21 ou suivant les règles d'utilisation des opérations de la SDAI spécifiées dans la partie 22,
- l'analyse structurelle consiste à vérifier que les données sont correctement construites par rapport à l'AIM; en particulier, toutes les contraintes globales et locales doivent être respectées,
- l'analyse sémantique consiste à valider les données suivant leur cohérence et leur adéquation vis à vis du domaine; l'ARM est ici plus particulièrement exploité.

Le postprocesseur consomme des données encodeées au format standard STEP ou exécute des opérations de la SDAI. Le produit du traitement consiste en la description des données encodées dans le format spécifique. Pour le postprocesseur, seule l'analyse sémantique du résultat est significative.

FIG. 7.1 – *Le traitement des tests abstraits*

Chapitre 8

Les environnements STEP

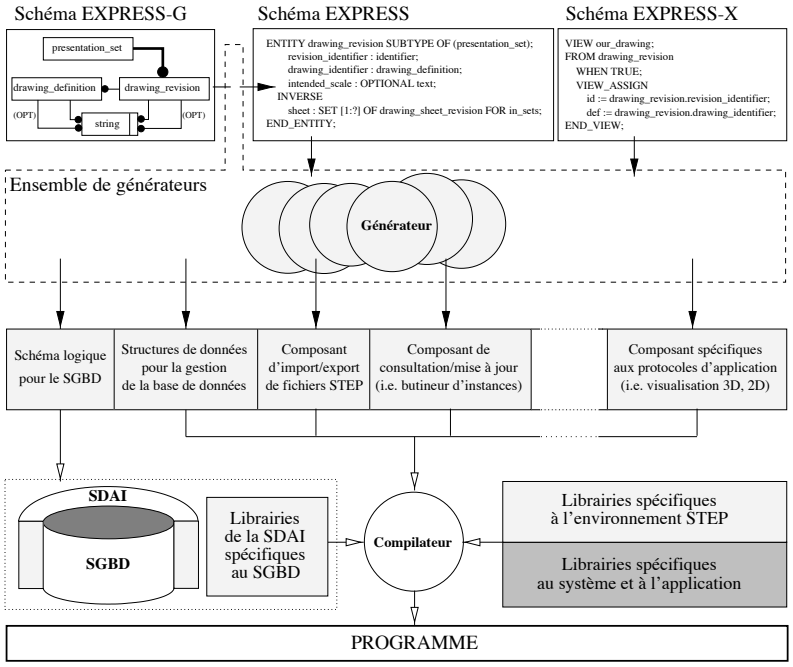
L'exploitation de la norme STEP dans un système particulier nécessite au minimum la mise en œuvre de la SDAI. Cette mise en œuvre constitue la base logicielle minimale nécessaire au développement des composants d'import et d'export des données encodées dans le format d'échange standard STEP. La production de ces composants est généralement automatique. Cette possibilité constitue un argument essentiel pour l'utilisation de la norme STEP et en particulier du langage EXPRESS pour la description des données [HLG94].

Par la capacité à échanger des données dans un format standard, l'utilisation de STEP est un facteur puissant d'intégration inter-applications.

Les environnements STEP industriels actuellement disponibles tel que, par exemple celui commercialisé par la société *STEP Tools*[1], se composent d'un ensemble d'outils de conception et de génération de code et d'un ensemble de librairies spécifiques aux systèmes et à la base de données utilisée :

- les outils de conception sont des éditeurs soit graphiques pour la spécification de schémas EXPRESS-G, soit textuels, pour la spécification de schémas EXPRESS; ces outils comprennent des gestionnaires de bibliothèque de schémas conceptuels, constitués par des butineurs de schémas; un outil de traduction de schémas graphiques EXPRESS-G peut être utilisé pour la génération vers une représentation équivalente en EXPRESS,
- un ensemble de traducteurs associés à des générateurs de code peuvent être utilisés pour au minimum, la génération de la SDAI spécialisée et d'un composant de lecture/écriture de fichiers au format standard STEP; les environnements plus complets proposent des générateurs pour les interfaces graphiques de consultation/mise à jour des données dans la base de données et, pour les applications de gestion d'objets techniques, des interfaces graphiques de visualisation en deux ou trois dimensions des données de description de produits,
- les descriptions conceptuelles spécifiées avec des extensions d'EXPRESS telles que EXPRESS-P ou EXPRESS-X sont exploitées par des générateurs pour la production de composants plus spécifiques tels que la gestion des données dans une base de données répartie, la génération de vues logiques des données ou encore la génération de composants de migration de données entre deux bases de données,
- les bases de données cibles peuvent être soit relationnelles, soit orientés-objet; elles sont de toute façon toujours manipulées indirectement, par le biais de la SDAI; les outils proposés sont spécialisés pour des systèmes et environnements cibles particulier.

1. http://www.steptools.com



Pour la construction d'une application, un environnement STEP est exploité dès la phase conceptuelle du cycle de vie. Les schémas EXPRESS-G, EXPRESS et de ses extensions sont exploités par un ensemble de générateurs qui produisent les schémas logiques des données pour le système de gestion de base de données cibles et des composants logiciels dans le langage de programmation cible, pour principalement, la gestion des données. Ces composants, associés aux composants spécifiquement liés au domaine de l'application, sont compilés pour produire le programme cible.

FIG. 8.1 – *Un environnement STEP*

Chapitre 9

En conclusion

La norme STEP est mise en œuvre pour répondre au besoin grandissant d'échange d'information et d'inter-opérabilité des applications industrielles. L'originalité de ce standard ISO est de s'intéresser non seulement aux moyens techniques à mettre en œuvre pour l'échange et le partage des données, mais aussi à la définition normalisée des informations échangées. Cette normalisation s'applique à la spécification du contenant (les schémas EXPRESS) mais aussi à l'intégration et l'interprétation du contenu des modèles de définition des données. La norme STEP comprend en effet une liste grandissante de schémas EXPRESS standards, normalisés par métier industriels, les *protocoles d'application*. STEP permet ainsi la réutilisation standard au niveau conceptuel.

9.1 Les points forts

9.1.1 L'indépendance de la norme

La norme STEP est un standard international développé en partie par et intégralement pour les utilisateurs. STEP est une norme ISO indépendante des fournisseurs d'outils informatiques et des prestataires de services. Cette caractéristique assure au standard un fort niveau d'indépendance vis-à-vis des technologies et de leurs évolutions [Lof98].

9.1.2 Le langage EXPRESS

La norme STEP est ouverte, elle est basée sur le langage de modélisation EXPRESS qui peut être utilisé pour tout métier industriel. De fait, le nombre de protocoles d'applications standardisés et d'utilisateurs de la norme s'accroît. Cette vitalité assure une certaine pérennité à la norme [Lof98].

Etant donné la nature en partie impérative du langage, EXPRESS permet la spécification de contraintes relativement simplement. L'utilisation de ce langage dans l'industrie est de plus facilitée par l'approche très pragmatique suivie pour sa définition.

EXPRESS est un langage textuel qu'il est possible d'analyser automatiquement, sans ambiguité, avec un analyseur lexical et syntaxique. De plus, EXPRESS, la SDAI et le format d'échange standard constituent les éléments d'une technologie très cohérente. Ces caractéristiques permettent une exploitation automatique des modèles et la construction d'outils de génération d'application : le format d'échange valide pour un schéma étant implicitement défini, les constituants d'échange des données (comprenant notamment la SDAI) peuvent être produits automatiquement à partir d'EXPRESS.

9.2 Les points faibles

Le processus d'interprétation des protocoles d'application est décrit de façon informelle. Les mêmes concepts peuvent être interprétés différemment d'un protocole d'application à un autre. En conséquence, lorsque plusieurs protocoles d'application standards doivent être exploités par une même application, les difficultés d'intégration peuvent être très grandes. En effet, la réutilisation conceptuelle ne s'opère qu'au niveau de l'entité. Les conflits de type qui peuvent résulter des différences d'interprétation peuvent aboutir à une impossibilité d'intégrer plusieurs protocoles d'application.

9.3 Les utilisateurs

La technologie développée par STEP est principalement exploitée pour la définition, l'échange et le partage de la définition des données de produits industriels dans les domaines de la CAO et de la CFAO. De part sa nature et de part le coût de mise en œuvre des protocoles d'application standards, les utilisateurs sont en premier lieu, les grands groupes industriels (*General Motors*, *Boing*, *Hitachi* ou encore *Dassault*). Pour ces grands groupes, la gestion cohérente et optimisée des informations tout au long du cycle de vie des produits est un enjeu stratégique fondamental.

Comme le montrent les nombreuses applications de STEP décrites dans [War98, EUG94, EUG95], cette technologie peut être utilisée pour la gestion des données, dans le cadre d'applications de natures très diverses. Notre propre expérience de l'utilisation de STEP, pour la gestion de données scientifiques [SYS96b, Pla96, SYS97] ou dans le cadre d'une application plus modeste [SYS96a][1], confirme ce point de vue.

1. environ quatre mois ingénieur

Bibliographie

[Aa94] S. Arbouy and al. *STEP Concepts fondamentaux*. AFNOR, 1994.

[Bou95a] M. Bouazza. *La norme STEP*. Hermès, Documentation multimédia, 1995.

[Bou95b] M. Bouazza. *Le langage EXPRESS*. Hermès, Documentation multimédia, 1995.

[Che94] R. Chen. The ISO STEP Pilot Product Logistic Support Application Protocal Suite, Developmental Plan. Technical report, Carderock Division, Naval Surface Warfare Center, Bethesda, Maryland 20084-5000, 1994.

[Dan93] W. F. Danner. STEP Data Specification Methodology. Technical report, ISO TC184/SC4/WG5 N50, 1993.

[EUG94] *EXPRESS User's Group, EUG'94 Conference Notes*, 1994.

[EUG95] *EXPRESS User's Group, EUG'95 Conference Notes*, 1995.

[Fed93a] Federal Information Processing Standard Publication 183, editor. *Integration Definition for Function Modeling (IDEF0), FIPS PUB 183*. National Institute of Standards and Technology, December 1993.

[Fed93b] Federal Information Processing Standard Publication 184, editor. *Integration Definition for Information Modeling (IDEF1X), FIPS PUB 184*. National Institute of Standards and Technology, December 1993.

[HLG94] Lynwood E. Hines, Craig T. Lanning, and Anthony J. Gradient. Conceptual vs Implementation Schemata in STEP: Religious Argument, Pragmatic Reality or Something Else? In EUG94 [EUG94].

[ISO92] ISO TC184/SC4/WG6 N29. *Part 31: Conformance testing methodology and framework: general concepts*, 1992.

[ISO94a] ISO 10303-1. *Part 1: Overview and fundamental principles*, 1994.

[ISO94b] ISO 10303-11. *Part 11: EXPRESS Language Reference Manual*, 1994.

[ISO94c] ISO 10303-21. *Part 21: Clear Text Encoding of the Exchange Structure*, 1994.

[ISO94d] ISO 10303-22. *Part 22: Standard Data Access Interface*, 1994.

[ISO94e] ISO TC184/SC4/WG5 WD. *PISA Information Modelling Language: EXPRESS-C*, 1994.

[ISO95a] ISO 10303-23. *Part 23: C++ Programming Language Binding to the Standard Data Access Interface Specification*, 1995.

[ISO95b] ISO 10303-23. *Part 24: C Programming Language Binding to the Standard Data Access Interface Specification*, 1995.

[ISO95c] ISO 10303-23. *Part 24: IDL Programming Language Binding to the Standard Data Access Interface Specification*, 1995.

[ISO95d] ISO TC184/SC4/WG5 N230 WD. *The Process Modelling Language EXPRESS-P*, 1995.

[ISO96] ISO TC184/SC4/WG5 WD. *EXPRESS-X Reference Manual*, 1996.

[ISO97] ISO TC184/SC4/WG11 N041 WD. *EXPRESS Language Reference Manual*, 1997.

[Lof98] David Loffredo. *Efficient Database Implementation of EXPRESS Information Models*. PhD thesis, Rensselaer Polytechnic Institute, Troy, New York, May 1998.

[NH89] G. M. Nijssen and T. A. Halpin. *Conceptual Schema and Relational Database Design: A Fact Oriented Approach*. Prentice Hall, 1989.

[Obj95] Object Management Group. *The Common Object Request Broquer: Architecture and Specification, revision 2.0*, 1995.

[Pal95] M. Palmer. Guidelines for the development and approval of STEP application protocols. Technical report, ISO TC184/SC4/WG4 N511, 1995.

[PK96] T. A. Phelps and J. D. Kindrick. Guidelines for the development of STEP abstract test suites. Technical report, ISO TC184/SC4/WG6 N100b, 1996.

[Pla96] Alain Plantec. Utilisation de la norme STEP pour la mise en oeuvre de la structure d'accueil VITAC. *Le bulletin technique, SYSECA, 66, rue Brossolette, 92240, Malakoff, France*, 4(3), October 1996.

[Spi94] P. Spiby. EXPRESS Semantic Meta-model for ISO 10303-11. Technical report, ISO TC184/SC4/WG5 N228, 1994.

[Str92] B. Stroustrup. *Le langage C++*. Addison-Wesley, 1992.

[SYS96a] SYSECA. L'application portdxf. Technical report, SYSECA, 32 quai de la douane, 29200, Brest, France, 1996.

[SYS96b] SYSECA. L'application vitac. Technical report, SYSECA, 32 quai de la douane, 29200, Brest, France, 1996.

[SYS97] SYSECA. L'application rafos. Technical report, SYSECA, 32 quai de la douane, 29200, Brest, France, 1997.

[War94] Barbara D. Warthen. STEP Architectures. In PDI 03/1994 [War98], page 7.

[War98] Barbara D. Warthen, editor. *Product Data International*. Warthen Technology Info. Services, 1995-1998.

[Weg96] Peter Wegner. Interoperability. *ACM Computing Surveys*, 28(1), 1996.