Définition d'un protocole d'application STEP pour la simulation en électromagnétisme

Singva Ma

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

|_|_|_|_|_|_|_|_|_|_|_|_|

THÈSE

pour obtenir le grade de

DOCTEUR DE L'INPG

Spécialité : Génie Électrique

préparée au **Laboratoire d'Électrotechnique de Grenoble**
dans le cadre de
l'École Doctorale Electronique Electrotechnique Automatique Télécommunications Signal

présentée

par

Singva MA
(Ingénieur ENSIEG, INPG)

29 Juin 2001

Titre :

Définition d'un Protocole d'Application STEP pour la Simulation en Electromagnétisme

---

Directeurs de thèse : Jean-Louis COULOMB, Yves MARECHAL

JURY

Monsieur	Denis Vandorpe	Président et Rapporteur
Monsieur	Patrick Dular	Rapporteur
Monsieur	Arnulf Kost	Examinateur
Monsieur	Jean-Louis Coulomb	Examinateur
Monsieur	Yves Maréchal	Examinateur

Remerciements

Ce travail de thèse a été réalisé au Laboratoire d'Electrotechnique de Grenoble et financé par une allocation du Ministère de l'Education Nationale, de la Recherche et de la Technologie. Je remercie Messieurs Jean-Claude Sabonnadière et Jean-Pierre Rognon, qui ont été successivement Directeurs du LEG pour m'avoir accueillie dans leur laboratoire. Je remercie Messieurs Gérard Meunier et Jean-Louis Coulomb qui ont été respectivement ancien et nouveau Directeur de l'Equipe Modélisation et CAO en Electromagnétisme pour m'avoir accueillie dans leur équipe.

Pour m'avoir fait l'honneur de participer à mon jury de thèse, j'adresse de respectueux remerciements à Messieurs :

Denis Vandorpe, Professeur à l'Université Claude Bernard Lyon 1, qui a accepté d'être rapporteur du mémoire de thèse et d'en présider le jury,

Patrick Dular, Docteur Chercheur Qualifié à l'Université de Liège, qui a accepté d'être rapporteur du mémoire de thèse,

Arnulf Kost, Professeur au Lehrstuhl Allgemeine Elektrotechnik de Cottbus, qui est a accepté de participer au jury de soutenance,

Jean-Louis Coulomb, Professeur à l'ENSIEG, qui a encadré cette thèse,

Yves Maréchal, Maître de Conférence à l'ENSIEG, qui a également encadré cette thèse.

Je souhaiterais avant tout exprimer une profonde gratitude à mes directeurs de thèse qui ont été également mes principaux enseignants pendant mes années d'élève ingénieur. Merci à toi Jean-Louis pour tes qualités d'enseignant, de chercheur, de chef d'équipe, pour ta diplomatie, mais également ta sagesse et ta bonne humeur. Merci à toi Yves pour ces discussions interminables pendant mon DEA et ma thèse, ton avis éclairé sur de nombreux sujets, tes idées décidément peu communes mais toujours pertinentes, tes compétences innombrables et inépuisables, ta franchise et ton soutien à chaque moment. J'ai beaucoup aimé travailler sous votre direction pendant mes dernières années d'étude et pendant ces trois années de thèse. Ma formation trop scientifique est sans nul doute insuffisante pour me permettre d'exprimer de manière juste toute ma reconnaissance pour ce que vous m'avez apporté.

Je remercie les nombreux chercheurs du LEG, dont la qualité et la renommée ne sont plus à démontrer, pour ce climat tout particulier qu'ils contribuent à créer, chacun dans leur domaine de compétence, pour faire avancer la science. Les discussions plus ou moins brèves que j'ai eues avec eux m'ont à chaque fois montré de nouveaux horizons. Je remercie en particulier Afef Lebouc, Nourdine Hadj-Said, Roland Pacaut, Jaime Fandino, Alain Bolopion, Eric Escande avec qui j'ai eu l'occasion de travailler en recherche et/ou en enseignement et dont j'ai pu apprécier l'expérience, le professionalisme et les précieux conseils.

J'adresse un merci particulier aussi à ma colocataire de bureau, Elisabeth Rullière, que j'admire pour m'avoir prouvé qu'il est possible de bien faire son travail sans négliger sa famille. Elisabeth, ton aide dans l'enseignement de l'électrotechnique m'a été inestimable pendant les

derniers mois où j'en avais le plus besoin, merci d'avoir partagé ton bureau et ton savoir.

Je remercie tous les Ingénieurs, Techniciens et Administratifs du LEG pour leur aide. Un grand Merci à Patrice Labie et Patrick Eustache qui, bien plus que des collègues, me sont devenus des amis sincères et appréciés. Patrice, je n'oublierai pas mon baptême de ski de randonnée et ces beaux champs de Sabots de Vénus que tu nous as montré un jour. Patrick, je ne saurais jamais assez te remercier pour ton immense savoir en informatique comme dans tous les domaines de la vie quotidienne, pour ta droiture et tes principes. Un grand Merci également à Florence François, Marie-Thérèse Loubinoux, Elise Riado, Monique Boizard, Etiennette Callegher.

Je remercie les chercheurs des autres laboratoires avec qui j'ai pu travailler et discuter en diverses occasions. Un grand merci particulièrement à Laurent Krähenbühl et Olivier Fabregue du Cegely, Jean-Claude Léon, Laurent Rémondini, et Jean-Francois Grabowiecki, du laboratoire 3S, Michel Tollenaere du laboratoire Gilco, Parisa Ghodous de l'Université Claude Bernard Lyon 1.

Pour ces trois années passées sur la même voie vers un même but, je remercie de tout coeur les doctorants et DEA du LEG. Chaque matin, après-midi et soirée, chaque pause café, sortie et tournois de foot, sortie de ski, chaque congrès, que nous avons passés ensemble resteront dans mes souvenirs comme autant de bons moments de franche camaraderie, de partage de connaissances et d'échange d'expérience aussi bien dans la résolution des problèmes scientifiques qu'administratifs et autres tracas de la vie moderne. C'est grâce à votre aide et par l'aide que je vous ai parfois apportée que j'ai pu progresser. Alors Fleur mon homonyme, Delphine, Afef, Gwenola, Daniela, Corinne, Charlotte, Christina, Gérald Hi G, Jean-Daniel dit Jedi, Olivier le Breton, Mickael 306 pro, Kérim rhum and bike power, Christophe dit ccm, Fabien, Nikola, Yann, Jean-Michel, Salou, Stefan, Khaled, Abbas, Djamel, David, Mauricio et tous ceux que je n'ai pas cités, bonne chance à toutes et à tous pour la suite, que vous restiez encore une année ou deux, que vous soyiez sur le point de partir ou que vous soyiez déjà partis. Merci Aktham pour tes explications sur les mystères du football et pour ta disponibilité. Merci Denis pour les lundis soirs que nous avons passés à réviser mes cours de Rock and Roll et les randonnées vers les plus hauts sommets des environs. Merci Alita, nous avons passé presque en même temps les mêmes épreuves pour ce travail de doctorat et j'espère un jour visiter avec toi ton Indonésie natale.

Merci à vous Nigel, Tina, Christophe, Christelle, et mes autres compagnons de snowboard avec qui j'ai passé une grande et belle saison remplie des joies et de l'ivresse de la glisse. J'espère un jour revoir nos traces se croiser sur la poudreuse.

Je vous remercie également Nathalie, Aaron, Fred, Phillippe pour ces quelques soirées et séjours en montagne, pour votre initiation au Roller et aux joies de l'insouciance.

Non je ne t'ai pas oublié Sven. Bon courage quelque soit l'avenir que tu as choisi.

Et vous autres qui êtes si loin maintenant, Takako, Hideo, Silvio, je n'ai pas oublié non plus les randonnées, les pistes de ski et les déjeuners et les diners qui ont scellé notre amitié. J'espère tenir mes promesses et pouvoir aller vous rendre visite pour témoigner de la beauté de vos pays respectifs dont vous m'avez tant parlé.

Je ne saurais non plus dire ou écrire assez de remerciements pour les Rabotins, ces habitants cosmopolites d'une cité universitaire perchée sur une moitié de montagne à 400 mètres au dessus de Grenoble. Merci Tanja, Maximilian, Warwrick pour avoir assisté avec moi à cette unique éclipse de soleil dans la campagne allemande. Merci Franz pour m'avoir fait découvrir les soirées pain, vin, fromage, les Calanques de Marseille à Cassis, l'escalade, la Provence française et la vie étudiante à Munich. Merci Nadia, Stève, Maria, Carole, Alexandra pour votre bonne humeur. Ces dernières années de séparation n'ont que renforcé notre amitié et mon besoin de vous revoir.

Et pour vous, mes amis qui partagez mon quotidien, Sylvie, Anthony, pourrais-je un jour arriver à vous exprimer ma gratitude pour votre présence, vos opinions, pour les innombrables soirées et sorties, les narrations de vos exploits épiques et de vos aventures à l'étranger ? Merci à Cyril, Frank, Fred, Louis. Comme je vous l'ai dit, une amitié est si difficile à construire et à conserver qu'on ne l'oublie pas pour de simple raisons de distance ou de temps. C'est à regret que je quitterai bientôt votre quotidien mais mon amitié vous reste acquise et intacte.

Merci de tout coeur à mes tantes, oncles, cousines, cousins, qui êtes si loin ou tout près. Merci pour votre aide à chaque moment, pour votre accueil chez vous, dans votre pays ou ici.

Merci Koang sok, pour vos precieux conseils et votre aide constante depuis si longtemps.

Thank you so much Cousin Voy, Chan Tie, Muy Tie, and all my cousins. Thank you for your hospitality during my stay in California.

Merci également à mes frères et soeurs. Est-il encore besoin de vous dire combien vous avez influencé chacun de mes choix ? Vous ne m'avez jamais rien refusé et je serai toujours là pour vous.

Mes derniers remerciements s'adressent à mes parents à qui je dois une infinité de fois plus que je ne pourrais leur rendre. Vous n'avez jamais économisé vos forces, votre temps ou votre argent pour nous donner à mes soeurs, mes frères et moi ce qu'il y a de mieux. Arriverais-je à vous faire comprendre un jour ce qu'est un loisir, que ce que vous appelez vacances n'est pour nous autres qu'un week-end prolongé ? Malgré votre grande ignorance de ces belles idées que nous apprenons à l'école et dans la pauvreté de mes premières années, vous avez toujours su me montrer le chemin que vous croyez être le plus juste. Trop souvent, sans vous connaître, des gens ont prétendu être plus beaux, plus savants, plus cultivés, meilleurs et vous ont peut-être méprisés mais je connais maintenant leur monde et je n'ai de véritable respect que pour vos semblables.

Table des matières

Liste des annexes

Liste des figures

Introduction Générale

Dans le domaine de la Conception Assistée par Ordinateur en Electromagnétisme, il existe des échanges de données de types très différents.

Le premier type d'échanges de données est un échange intra-logiciel entre différents modules d'une même application. En effet, une application telle que le logiciel Flux3d développé au Laboratoire d'Electrotechnique de Grenoble est composé de plusieurs modules dont le modeleur géométrique, le mailleur, le descripteur de physique, le solveur, le post-processeur. Les données échangées entre ces modules sont très variées (ce sont des données de type géométriques, maillage, physiques, numériques ou graphiques).

Un autre type d'échanges de données est un échange inter-logiciels du même domaine d'application dans le cadre d'une collaboration ou d'un changement de version. Les données échangées sont du même type que ceux décrits précédemment.

Un dernier type d'échanges de données est un échange inter-logiciels entre des applications provenant de domaines d'analyse différentes. En effet, la conception d'un produit en électrotechnique, implique de nombreux domaines. Il y a bien sûr, celui de la géométrie, mais aussi la mécanique, la thermique, les matériaux, les phénomènes électriques, l'acoustique et ceux que nous avons oubliés. Un moteur électrique par exemple met en jeu un ensemble très complexes de forces mécaniques, thermiques et électromagnétiques tous dépendants les uns des autres. Ainsi, pour pouvoir prévoir partiellement ou complètement le comportement de systèmes aussi complexes, il est apparu indispensable de disposer de modèles multiphysiques. L'intersection de ces domaines à priori différents, définit toutefois des ensembles non nuls. Ceci est facilement vérifiable puisque quel que soit le domaine étudié, certaines variables sont clairement identifiées et communes à la plupart d'entre eux (forces, températures, volumes, ...) et ces variables sont liés les unes aux autres par une multitude de relations mathématiques qui décrivent ces phénomènes physiques.

Ainsi les développeurs de logiciels de Conception Assistée par Ordinateur dans le domaine de l'Electromagnétisme sont arrivés à la conclusion qu'il existe maintenant un besoin soit de partager soit d'échanger des données, à la fois pour éviter des redondances d'information et de fastidieuses redéfinitions de la part des utilisateurs et pour optimiser le développement d'interfaces.

Les échanges de données ont été et sont encore étudiés dans les domaines de la CAO et de la CFAO en géométrie, analyse en mécanique des structures et en mécanique des fluides principalement. Ces domaines ont développé des méthodes et des concepts avancés dans la résolution des problèmes d'interfaçage entre applications et ont en particulier mis au point un certain nombre de méthodologies et de normes pour un échange optimal.

La solution la plus récente proposée dans la résolution de ces problèmes d'échanges de don-

nées inter-domaines est à la fois une norme internationale et une méthodologie : c'est la norme ISO 10303, également connue sous le nom STEP (Standard for The Exchange of Product model data).

Dans ce travail, c'est la solution STEP que nous nous proposons d'étudier et d'appliquer à notre domaine, celui de la simulation en électromagnétisme.

Présentation de la norme STEP

Les études sur les échanges de données en CAO (Conception Assistée par Ordinateur) a conduit à la nécessité de définir des consensus tels que des accords, standards ou des normes. Elles ont mené à la conclusion que pour des échanges de qualité, ces données devaient d'abord être formalisées.

Des standards tels que IGES (Initial Graphics Exchange Specification, USA), SET (Standard d'Echange et de Transfert, France), VDA (Verband Der Automobilindustrie, Allemagne) formalisent les données pour en permettre l'échange. Mais ces standards ne concernent que le format physique des données échangées : ils imposent la disposition et la syntaxe des informations dans un fichier. De ce fait, ils simplifient les structures de données et donc ne communiquent qu'un minimum d'information sur le modèle. Le processus par lequel le fichier a été généré, par exemple, n'est pas spécifié.

La norme STEP (STandard for the Exchange of Product model data) est la convergence de ces standards. Elle définit aussi un format physique (fichier d'échange) mais, en plus, elle spécifie le contenu des données échangées et formalise la méthodologie de modélisation de ces données.

De plus, au-delà du seul domaine de la CAO (Conception Assistée par Ordinateur), STEP se veut applicable dans tous les secteurs industriels. C'est donc un projet d'une grande complexité, difficile à mettre en oeuvre de par sa dimension internationale et sa nécessaire flexibilité d'application. Des travaux sont menés, par exemple, dans des secteurs aussi variés que ceux de l'automobile, l'aéronautique, l'électronique, l'informatique et le BTP. La conformité à la norme STEP semble maintenant intéresser de nombreuses grandes entreprises. Les projets de création de nouvelles *parties* ou simplement d'utilisation des *parties* déjà définies sont de plus en plus nombreux. Parmi ceux-ci, peuvent être cités :

- les projets Esprit et parmi ces derniers les projets ADCOMS et SEDRES qui associent Aliena Aerosazio et British Aerospace pour l'élaboration d'une interface standard pour la conception des avions et des navettes spatiales, le projet CIREP (Thomson), le projet européen GEM qui s'intéresse (entre autres) à la *partie* sur les éléments finis.
- le projet CSTAR qui associe McDonnell Douglas, Northrop Grumman, ITI, IBM pour l'utilisation, en production, de STEP dans les échanges de configuration de données pour le management.

- le projet General Motors STEP Translation Center : STEP sera utilisé comme norme pour transférer des modèles de produits entre les divisions GM, leurs clients et leurs fournisseurs.
- le projet d'ouvrir le premier laboratoire de test de conformité à STEP à l'institut GOSET (France).
- les projets du National Institute of Standards and Technology pour la création de nouvelles parties de STEP
- les projets du consortium Product Data Exchange
- les projets des grandes compagnies telles que la NASA, Boeing, Lockheed Martin, Snecma,... sur les différentes parties de STEP.

Contexte de l'étude

Dans la simulation en électromagnétisme, les phénomènes physiques et le calcul numérique des champs sont parfaitement connus : la communauté scientifique et technique a abordé les aspects les plus variés de la modélisation de ces phénomènes, des méthodes de calcul formels et numériques pour atteindre aujourd'hui un haut degré de maturité. Un logiciel de calcul en électromagnétisme tel que Flux3D met en oeuvre une cinquantaine de formulations différentes de problèmes, une dizaine de modèles, et autant d'algorithmes et de méthodes.

Cependant, l'intégration dans la chaîne de Conception Assistée par Ordinateur, par ailleurs bien maîtrisée dans le secteur de l'analyse en mécanique, est encore trop peu présente.

C'est pour introduire cet aspect complémentaire du calcul numérique que le projet international "ICS (International Compumag Society) STEP Committee" a récemment été mis en place. Constitué d'universitaires et de sociétés de service de différents pays, il a pour principal objectif d'introduire le domaine de l'électromagnétisme dans STEP. La présente étude se situe dans le cadre général de ce groupe de travail. La diffusion des expériences acquises sur STEP en est donc un aspect essentiel.

Contenu de ce travail

Un des aspects essentiels de notre travail a été la réalisation pratique des méthodes de modélisation de données que nous voulons appliquer au domaine de l'électromagnétisme. Notre principale motivation est bien sûr de participer à cette immense ambition qu'est STEP mais aussi et surtout de nous donner les connaissances et les moyens de l'aborder sans trop de retard. Il est certain que le coût de l'entrée dans la communauté STEP est extrêmement élevé en temps et en nombre de travaux réalisés : l'existant est très théorique et les réalisations encore à l'état de prototypes peu avancés. Etant parmi les premiers dans notre domaine à nous y intéresser, nous nous sommes plus attardés sur cette méthodologie complexe et sur son application informatique que sur des aspects résolutions numériques, par ailleurs si bien connus dans notre domaine.

Ce travail se propose de présenter la méthodologie STEP et d'en étudier l'applicabilité à

l'électromagnétisme. L'étude aboutit en particulier à une conclusion sur un modèle de données en magnétostatique et une démarche de modélisation que nous pensons utilisable dans le contexte plus général de la simulation en électromagnétisme. Il présente ensuite des outils logiciels pour la mise en oeuvre d'un tel modèle.

Ce document est divisé en cinq chapitres. Le premier chapitre présente une étude des principales techniques de modélisation de données. Le chapitre deux est un résumé de la méthodologie de modélisation et d'échanges de données introduites par STEP. Les principes et la mise en oeuvre de STEP y seront décrits. Le chapitre trois propose un modèle de données pour la simulation en Magnétostatique. Le chapitre quatre met en place des outils logiciels pour l'implantation de ce modèle. Le dernier chapitre est un exemple de mise en oeuvre pour le calcul éléments finis en électromagnétisme.

Notations et Conventions typographiques

L'aspect international de STEP impose l'utilisation de termes en anglais. Les traductions données par l'Association Française de NORmalisation seront utilisées pour la plupart. Dans le doute, les termes anglais seront gardés. Une traduction des sigles est fournie en Annexe C.

D'autre part, il existe des termes communs entre les différents langages et représentations (Java, C++, Express, UML,...). Par exemple, une *classe* pour STEP est un ensemble de *schemas* et pour Java ou C++ elle sera la définition d'un objet. Les termes spécifiques à la norme STEP seront notés en lettres *italiques*. Les termes spécifiques aux méthodes et langages orientés objets n'auront pas de convention typographique particulière. Les définitions de certains termes utilisés dans ce mémoire seront données dans l'Annexe A

Enfin, la convention typographique suivante sera adoptée pour l'écriture des codes informatiques (Express, Java, JavaCC,...) :

```
SCHEMA sche;
ENTITY ent;
END_ENTITY;
END_SCHEMA;
```

```
public class MyClass
{
  public MyClass()
  {
     super();
  }
}
```

Chapitre I

Modélisation de Données

Dans le domaine de la Gestion de Données Produit, la modélisation de données permet de formaliser les descriptions de produits pour une meilleure gestion des bases de données. En effet, actuellement, la complexité d'un produit est telle que, depuis sa conception jusqu'à sa fabrication et son service après-vente, des catégories de métiers très différents sont impliquées. Ces métiers ont besoin de communiquer mais les systèmes qu'ils utilisent, étant mal adaptés les uns aux autres, ne pouvaient échanger des données sans une phase de réplication ou parfois même de réécriture. De nombreuses études telles que [5], [1] montrent alors qu'il est nécessaire de disposer d'un modèle produit fédérateur. Ainsi, les systèmes les plus récents tentent, en définissant et en implémentant des modèles adaptés à chaque métier mais cohérents entre eux, d'apporter des solutions globales permettant d'intégrer ces métiers et leurs spécificités.

Dans le secteur du Génie Logiciel, la principale préoccupation est de produire des logiciels de qualité. Il faut bien sûr tout d'abord un logiciel contenant un minimum ou même aucune erreur. Le coût d'une erreur informatique peut être très élevée autant pour les développeurs que pour les utilisateurs. Mais parce que l'erreur n'est pas toujours prévisible, il faut également, dès la conception, en prévoir la maintenance. Une application mal documentée où seul le programmeur est capable de changer la moindre ligne réduit sa durée de vie. Il faut enfin, également à la conception, prévoir l'évolution de l'application. Si l'évolution est bien prise en compte, il est souvent moins coûteux de la modifier que de la refaire. Pour répondre à ces besoins, il est alors apparu avantageux de disposer d'un modèle de données permettant d'avoir une vue à la fois globale et détaillée des structures utilisées et des utilisations de chacune de ces structures, de modifier à moindre coût l'application conçue et de la faire évoluer au besoin.

Dans tous les domaines des technologies de l'information, la modélisation des données peut apporter une révolution importante à la fois dans la conception des systèmes et dans la conception de logiciels intégrés et surtout intégrables. Suite à l'expression de ces besoins, des techniques de modélisation ont été mises au point dans les différents domaines cités ci-dessus et par la suite utilisées largement par-delà leurs vocations premières.

Ce chapitre est une étude non exhaustive de ces techniques de modélisation. La section un décrit trois des principales méthodes et langages de modélisation utilisés dans Internet (XML), dans le Génie Logiciel (UML) et dans les modèles de données produit (STEP). La section deux fait une étude comparative de ces trois méthodes. Elle met en valeur les domaines de similitude et les manques de chacunes d'elles. La section trois décrit les évolutions de ces méthodes et conclut sur la possibilité d'une norme unique capable de recouvrir tous les domaines. La section quatre décrit nos motivations dans ce travail. La dernière section est une conclusion critique sur notre choix de STEP.

1 Techniques de Modélisation

Suivant le domaine pour lequel on veut modéliser, il existe des techniques et des langages spécifiques :

- XML, eXtended Mark-up Language, est adapté aux données destinées à la diffusion sur Internet
- UML, Unified Modeling Language, fait l'unanimité dans le domaine du Génie Logiciel
- STEP, Standard for the Exchange of Product model data, a été adopté dans le domaine des Données Produit et de la Conception Assistée par Ordinateur

Cette section est une description très succincte de ces trois techniques qui sont, dans leurs domaines respectifs, les plus récentes et les plus utilisées.

1.1 eXtended Mark-up Language : XML

La norme XML, eXtended Mark-up Language, est un ensemble de spécifications désignant les règles à adopter afin de créer des documents universels utilisables à travers Internet. Les données échangées sont contenues dans un fichier XML et le modèle de ces données est décrit dans un fichier DTD (Document Type Definition). La norme XML permet également de décrire la présentation des données dans un fichier XSL (eXtensible Stylesheet Language).

1.1.1 Généralités

La première norme industrielle de XML a été publiée le 10 février 1998 par le consortium World Wild Web Consortium (W3C).

XML est né de la volonté d'intégrer le savoir faire résultant d'une décennie d'expérience de SGML, en ne retenant que les caractéristiques essentielles, indispensables à un usage et une implémentation facile et surtout de différencier données, modèle de données et présentation de ces données.

XML a pour ambition d'optimiser les 3 points suivants :

- Universalité pour une utilisation inter-plateformes et inter-logiciels
- Flexibilité pour que chacun puisse définir son propre dérivé de XML dans ses documents
- Séparation du modèle de données, des données et de leur mise en page pour faciliter la mise à jour

1.1.2 Le document XML

Le document XML est celui qui contient les données. C'est un langage de marquage du type HTML. Chaque donnée est "marquée" par une balise qui indique sa sémantique. Un titre sera par exemple balisé par <title> et </ title>. Ainsi, un exemple de fichier XML pour un article de journal peut-être :

```
<!DOCTYPE ARTICLE PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
<article id=article>
<artheader>
  <title>Article Title</title>
  <subtitle>Article Subtitle</subtitle>
  <subtitle>Another Article Subtitle</subtitle>
  <mediaobject>
    <imageobject>
      <imagedata fileref="emc2.gif">
    </imageobject>
  </mediaobject>
  <author><surname>Walsh</surname><firstname>Norman</firstname><affiliation>
        <orgname>nwalsh.com</orgname>
        </affiliation></author>
  <abstract>
  <para>
This is an abstract.
  </para>
  </abstract>
</artheader>
</article>
```

1.1.3 Modèle de données : Document Type Definition, DTD

Le Document Type Definition (DTD) est la définition des structures du document XML. Son rôle est de permettre de définir ses propres balises. Il définit ainsi la structure que doit adopter un document XML pour être valide, c'est à dire conforme au langage.

Par exemple, le morceau de DTD qui décrit la balise HTML <TABLE> sera donné par le code suivant :

```
<!ELEMENT table - - (caption?, tr+)>
<!ELEMENT tr - O (th|td)*>
<!ELEMENT (th|td) - O %body.content>

<!ATTLIST table                          -- table element --
        align     %Where;   #IMPLIED     -- table position relative to window --
        width     %Length   #IMPLIED     -- table width relative to window --
        border    %Pixels   #IMPLIED     -- controls frame width around table --
        cellspacing %Pixels #IMPLIED     -- spacing between cells --
        cellpadding %Pixels #IMPLIED     -- spacing within cells --
        >
```

La définition des attributs qu'il faut donner après cette balise <TABLE> est donnée par le

mot clé *!ATTLIST*. On voit ci-dessus les attributs *align, width, border, cellspacing, cellpadding.*

1.1.4 eXtensible Stylesheet Language : XSL

le eXtensible Stylesheet Language, XSL, est un langage de transformation d'un document XML source vers un autre document XML que l'on appelle document résultat. Toute l'information peut être sélectionnée et réorganisée selon les besoins d'une présentation. XSL est constitué d'un ensemble de règles de transformation qui expliquent comment un document XML doit être converti en un document résultat mis en forme.

Il est aussi bien possible d'utiliser XSL du côté du serveur que du côté du poste client. Dans le premier cas, le serveur va avoir pour charge de préparer l'information utile pour le client et va ensuite la lui transmettre. De cette manière il n'existe aucune contrainte pour le poste client en terme de navigateur. Dans le deuxième cas, le poste client va télécharger les fichier XML et XSL et c'est le navigateur ou l'application web compatible XML qui va se charger de réaliser la transformation.

Le document XSL pour le fichier XML "article" précédent pourra être :

```
<!DOCTYPE style-sheet PUBLIC "-//James Clark//DTD DSSSL Style Sheet//EN" [
<!ENTITY docbook.dsl PUBLIC "-//Norman Walsh//DOCUMENT DocBook HTML
Stylesheet//EN" CDATA DSSSL>
]>
<style-sheet>
<style-specification use="docbook">
<style-specification-body>
(define (article-titlepage-recto-elements)
  (list (normalize "title")
        (normalize "subtitle")
        (normalize "mediaobject")
        (normalize "corpauthor")
        (normalize "authorgroup")
        (normalize "author")
        (normalize "releaseinfo")
        (normalize "copyright")
        (normalize "pubdate")
        (normalize "revhistory")
        (normalize "abstract")))
</style-specification-body>
</style-specification>
<external-specification id="docbook" document="docbook.dsl">
</style-sheet>
```

1.2 Unified Modeling Langage : UML

Comme son nom l'indique, UML, Unified Modeling Langage, est un langage de modélisation. Il est actuellement celui qui a obtenu le plus de succès dans le domaine du Génie Logiciel. Son objectif est de donner un langage de modélisation sans spécifier les méthodes de conception d'une application.

1.2.1 Historique

Le développement d'UML a débuté en 1995. Grady Booch et James Rumbaugh de Rational Software Corporation ont décidé de travailler ensemble pour réaliser une unification des principales méthodes à objet : Booch [3] et OMT (Object Modeling Technique) [23]. L'apport de la méthode OOSE (Object Oriented Software Engineering) [14] a eu lieu en 1996 lorsque Ivar Jacobson a intégré l'équipe initiale.

UML résulte donc de l'évolution de ces dernières méthodes avec les contributions d'autres méthodologistes (sous la responsabilité de l'Object Management Group), de distributeurs de logiciels (HP, IBM, Microsoft) et de nombreux utilisateurs et groupes d'utilisateurs. Il est ainsi possible de travailler facilement sur UML si on possède déjà de l'expérience dans les autres méthodes.

1.2.2 Diagrammes UML

UML définit 9 diagrammes pour représenter les différents points de vue de la modélisation. Ils permettent de visualiser et de manipuler les éléments de la modélisation. Les diagrammes définis par UML sont les suivants :

- les diagrammes d'activité : représentation du comportement d'une opération en terme d'action
- Les diagrammes de cas d'utilisation : représentation des fonctions du système du point de vue de l'utilisateur.
- Les diagrammes de classes : représentation de la structure statique en terme de classes et de relations
- Les diagrammes de collaboration : représentation spatiale des objets, des liens et des interactions.
- Les diagrammes de déploiement : représentation du déploiement des composants sur les dispositifs matériels.
- Les diagrammes d'états-transitions : représentation du comportement d'une classe en terme d'état.
- Les diagrammes d'objets : représentation des objets et de leurs relations, correspond à un diagramme de collaboration simplifié, sans représentation des envois de message.
- Les diagrammes de séquence : représentation temporelle des objets et de leurs interactions.

Avec ces diagrammes UML offre des éléments de modélisation pour l'ensemble de la conception orientée objets.

En effet, tout d'abord, UML permet la modélisation des cas d'utilisation. La modélisation des cas d'utilisation définit les besoins fonctionnels d'un système selon le point de vue d'une catégorie d'utilisateurs à la fois. La modélisation par les cas d'utilisation fait le lien entre les analystes, les utilisateurs et les développeurs.

UML permet également la modélisation de classes. Elle permet de définir des objets-métier et l'architecture de l'application. Les objets sont créés comme instances de classes. Ils interagissent dynamiquement pour permettre le comportement décrit par les cas d'utilisation. La modélisation objet définit le comportement requis par les différentes classes pour assurer la bonne mise en place des cas d'utilisation et des règles de gestion. Les objets-métier constituent la base de l'architecture des applications. Ces objets peuvent être réutilisés à travers des domaines d'application ou encore être identifiés et dérivés directement des cas d'utilisation ou du domaine de l'application. La modélisation des classes est le détail de la structure des objets-métiers. Les classes constituent la base pour la génération de code et pour la génération des schémas de bases de données. Les définitions des classes et de leurs relations sont regroupées dans des paquetages afin de définir l'architecture des applications. Ces paquetages peuvent être emboîtés les uns dans les autres. Les relations entre paquetages définissent les dépendances dans l'application et déterminent la stabilité de l'architecture.

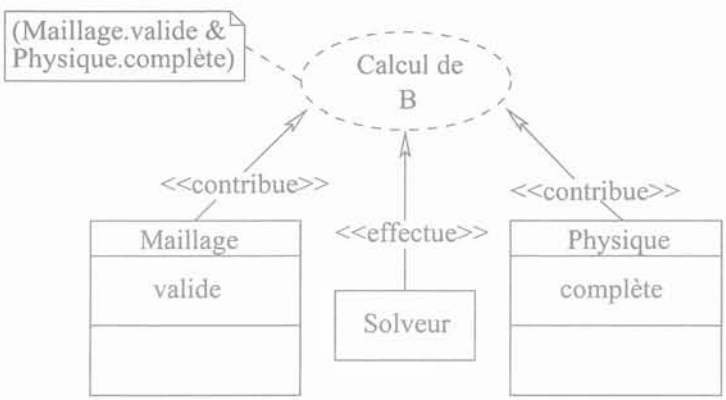


FIG. 1.1 – Diagramme de collaboration UML

UML permet ensuite la modélisation des composants. Les composants sont les unités physiques de code source et les unités exécutables qui sont assemblées pour former des applications. Les classes sont affectées à des composants fournissant des briques réutilisables pour la construction des applications. Ces composants formeront la base pour une architecture d'application plug-and-play. La réutilisation dans le langage UML intervient avant la compilation sous forme de classes réutilisables ou de frameworks et après la compilation sous la forme d'assemblage de composants.

UML permet enfin la modélisation de la distribution et du déploiement. La modélisation de déploiement représente la façon dont l'application est distribuée dans un réseau. La modélisation

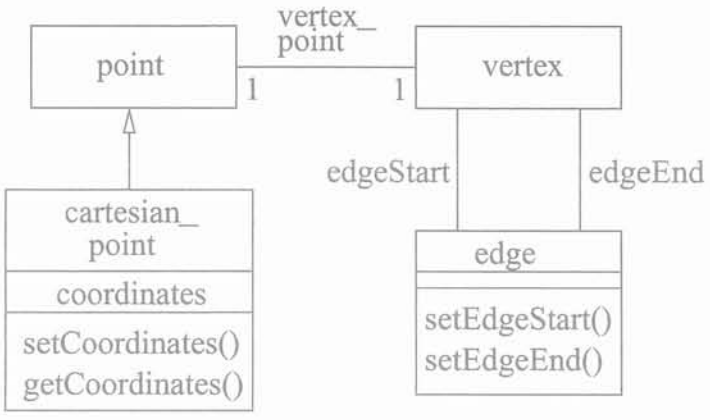


FIG. 1.2 – Diagramme de classes UML

du déploiement tient compte des différentes topologies de réseau, par exemple les architectures client/serveur, trois-tiers ou Internet/Intranet. UML permet de décrire la topologie des noeuds dans le réseau, la façon dont ces noeuds sont connectés et la manière dont l'application est partitionnée et distribuée sur ces noeuds.

1.2.3 Exemples

La Figure 1.1 montre un exemple de diagramme de collaboration. La Figure 1.2 montre un exemple de diagramme de classes.

1.3 Standard for The Exchange of Product model data : STEP

La norme ISO 10303, plus connue dans les entreprises sous le nom de STEP (STandard for the Exchange of Product model data), a pour ambitions de donner non seulement un modèle de produit mais également une méthodologie de modélisation de ce produit. Elle est constituée d'un ensemble de textes normalisés, encore en cours de développement pour la plupart, mais dont certains ont déjà obtenu le statut de norme internationale et donc reçu l'approbation d'un très grand nombre de pays et d'industries.

1.3.1 Historique

Dès la fin des années 70, le domaine de la Conception Assisté par Ordinateur a défini des standards tels que IGES (Initial Graphics Exchange Specification, USA), SET (Standard d'Echange et de Transfert, France), VDA (Verband Der Automobilindustrie, Allemagne). Ces standards formalisent les données pour en permettre l'échange. En 1990, les trois organismes de normalisation ANSI (USA), DIN (Allemagne), AFNOR (France) décident d'une mise en commun de leurs standards et adoptent STEP. La Figure 1.3 résume l'évolution des normes d'échanges de données dans le domaine de la Conception Assistée par Ordinateur.

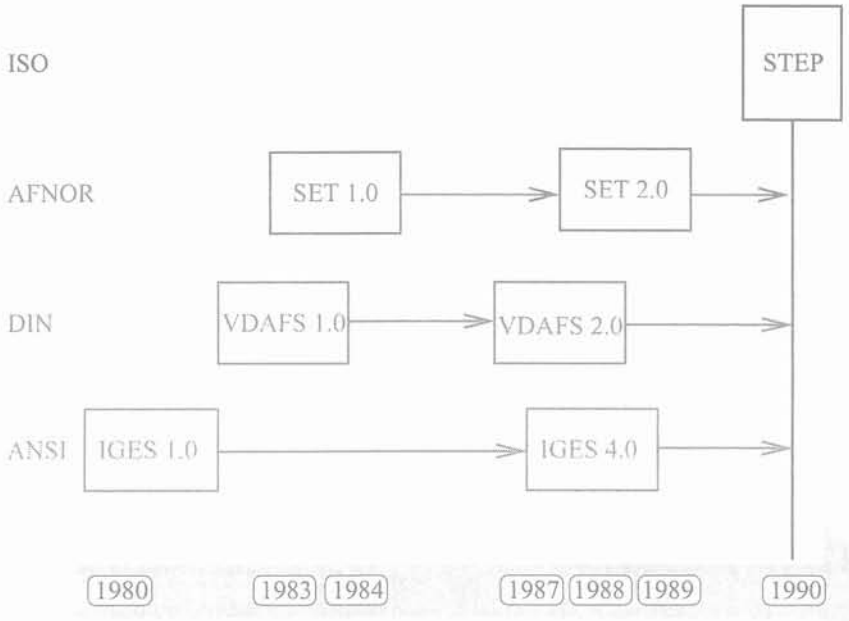


FIG. 1.3 – Historique de STEP

1.3.2 Modèle de données STEP

Les modèles de données STEP sont rangés dans sept *classes* dont nous ne décrirons ici que les deux principales. La première classe contient le modèle de ressources génériques. Elle spécifie toutes les définitions de données requises par tous les autres modèles. Par exemple, la partie 42 est le modèle de données pour la géométrie et la topologie. Cette classe forme le coeur de STEP. La deuxième classe est celle des Protocoles d'Application. Elle contient les données nécessaires pour un domaine d'application spécifique. Ce sont ces parties qui sont directement mises en oeuvre.

1.3.3 Méthodes d'implantation

Une des caractéristiques du standard STEP est qu'il spécifie également comment les modèles doivent être implantés dans un langage particulier. Par exemple la partie 23 définit comment un Protocole d'Application STEP est traduit en langage C++. Il est envisagé ainsi une partie pour chaque langage informatique (C, C++, Java, IDL, XML, ...).

2 Etude Comparative

La section précédente a décrit succinctement les trois techniques de modélisation les plus récentes XML, UML et STEP ainsi que les domaines dans lesquels elles sont nées et les domaines dans lesquelles elles sont utilisées. Cette section fait une étude comparative entre elles. Après une sous-section sur l'historique de synthèse, nous montrerons les points communs et les divergences. Puis nous décrirons les liens qui ont été développés par les organismes responsables de la

maintenance de ces standards pour communiquer entre eux.

2.1 Chronologie

Le schéma Fig.1.4 résume les étapes d'évolution de trois normes XML, UML et STEP. STEP étant issu du monde de la CAO est plus ancien. Le Génie Logiciel est une science encore jeune et les méthodes (OMT, Booch, OOSE, UML) qui s'y rapportent sont donc plus récentes. Quant à XML, norme issue du développement d'Internet est bien sûr la plus récente des trois.

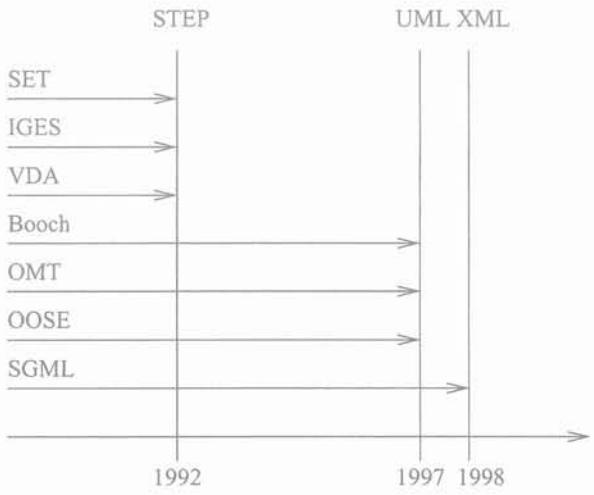


FIG. 1.4 – Historique des Normes XML, UML et STEP

2.2 Convergence et divergence

Le schéma Fig.1.5 résume les notions communes et celles qui diffèrent dans les trois notations décrites dans la section précédente.

2.2.1 Convergence : une base commune

Quelle que soit la notation utilisée, la représentation des données passe par une phase de modélisation "conceptuelle" où les différentes notions relevant de l'univers du discours sont identifiées, ainsi que leurs relations et leurs attributs descriptifs.

Ainsi, les normes XML, UML et STEP reconnaissent toutes trois tout d'abord le besoin de structurer les données, c'est à dire de formaliser les relations entre les instances. Dans tous les cas, on peut définir un arbre de dépendance pour les données échangées : les branches de cet arbre représentent des liens d'héritage/dérivation, d'association, d'agrégation et d'appartenance.

Un autre point de convergence de ces méthodologies est la séparation entre le modèle de données et les données elles mêmes : le modèle de données décrit les données et les données

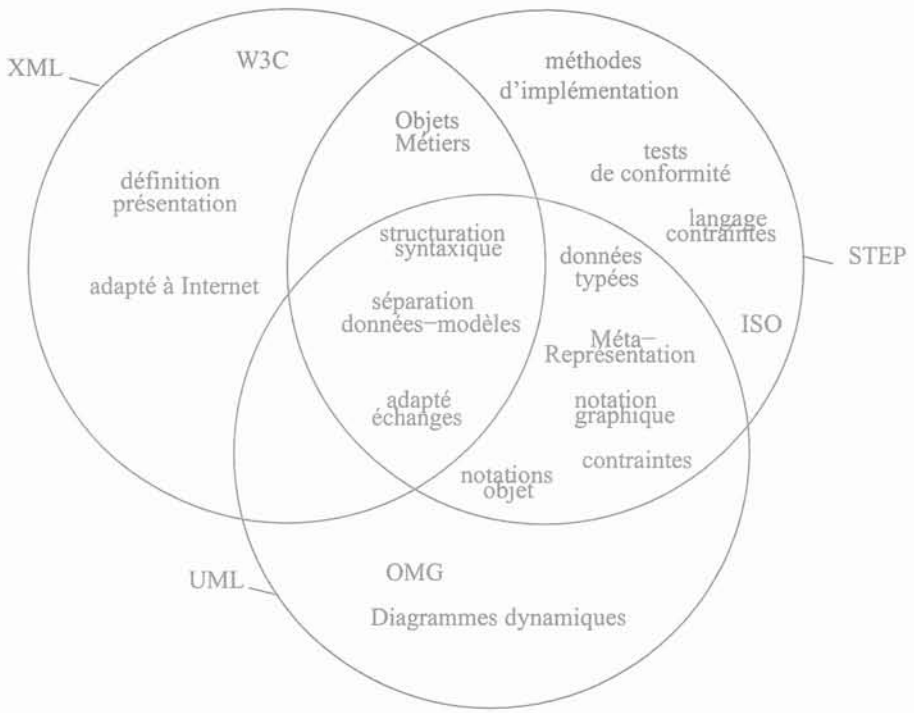


FIG. 1.5 – XML, UML, STEP : domaines de convergence

décrivent l'objet échangé. Partager ces deux niveaux d'abstraction a été reconnu comme le meilleur moyen d'avoir des standards fiables.

Enfin, il est certain que chacune de ces méthodologies est l'expression d'un besoin d'optimiser les échanges : minimiser les pertes d'information, automatiser autant que possible les conversions du modèle natif vers le modèle standard et inversement. D'où le développement d'outils de manipulation des données et des modèles de données : éditeurs spécialisés, scanneurs lexicaux et analyseurs syntaxiques, analyseurs sémantiques, générateurs de code, gestionnaires de bases de données.

2.2.2 Divergences

On remarque tout d'abord que XML dans sa version actuelle est peu adapté à la modélisation orientée objets de produits complexes. Cette norme n'admet effectivement aucun typage de données. Pis encore, il est impossible de contraindre les relations. Il n'est pas possible, par exemple, de préciser qu'un champ doit contenir un nombre, ni a fortiori que ce nombre doit être un entier positif.

Il manque également dans XML une notation graphique plus facile à lire qu'une notation texte. S'il est vrai que la notation graphique peut être mise en oeuvre à travers une XSL, celle-ci devra se faire avec des conventions "maison" qu'il faudra étudier avec tous les partenaires. C'est donc une source supplémentaire d'incertitude.

XML est donc une norme qui ne peut être, pour le moment, utilisée pour des échanges autres que les échanges de données de publication sur Internet.

L'une des principales différence entre STEP et UML est la définition d'un langage littéral. En effet, les deux méthodologies proposent un langage graphique. Mais l'une des clés de la méthodologie STEP est l'utilisation du langage littéral EXPRESS. En ajoutant à la notation graphique un langage littéral associé à une syntaxe formelle et une sémantique bien définie, et en intégrant dans ce langage les constructions permettant d'exprimer n'importe quel type de contraintes, STEP permet de passer d'une méthode de modélisation approximative d'un domaine à une méthode de spécification de données. Une spécification de données en EXPRESS n'est pas un modèle utilisé à l'intérieur d'une démarche. Il constitue une spécification formelle à partir de laquelle de nombreuses représentations peuvent se générer ou se vérifier.

Ce qui manque également dans UML est la spécialisation par métier. Comme son nom l'indique, c'est un langage et les contenus des modèles décrits dans ce langage ne seront pas diffusés ni normalisés. Les échanges ne seront donc pas d'ordre sémantiques mais exclusivement syntaxiques. Le reste de la démarche de conception, à savoir le contenu même des modèles sera laissé à la discrétion des partenaires de chaque projet. Par exemple, il sera de la responsabilité de chacun des partis de déterminer le nombre de schémas nécessaires et suffisants et le contenu de ces schémas.

Quant à STEP, il manque un typage plus précis des relations d'attribut (agrégation, association,...) et surtout un modèle dynamique comme celui prévu dans UML à travers les diagrammes d'activité et les diagrammes de cas d'utilisation (voir la sous-section 1.2.2). On pourrait dire que le modèle d'activité existe déjà dans STEP du fait que chaque Protocole d'Application doit définir un modèle IDEF0 de l'activité modélisée. Mais ce modèle est moins puissant que les mécanismes proposés par UML. UML permet en effet de décrire de manière générique la structure d'une application orientée objets, les situations dans lesquelles cette application sera utilisée, les différents états dans lesquels seront chacun des objets décrits et le séquencement exact dans lequel les objets interviendront. Il permet un niveau de détails tel que la mise en oeuvre dans un langage donné du modèle soit la plus courte et la plus automatique.

2.2.3 Des ponts entre les différentes méthodologies

Des travaux ont été faits pour construire des liens entre STEP et UML. Ou plutôt entre certaines parties de STEP et certaines parties d'UML. Mais pour la plupart, ils sont trop récents pour qu'on puisse conclure. STEP et UML disposent tous les deux d'un méta modèle. STEP et UML sont des formalismes qui ont obtenu chacun dans leur domaine le consensus de tous les acteurs du domaine. Il est vrai que la partie dynamique d'EXPRESS n'est qu'à l'état de projet et manque dans le domaine du Génie Logiciel. Cependant, UML ne formalise pas l'approche métier. Il sera donc nécessaire d'avoir d'autres accords (quels schémas sont requis par les partis, quel domaine doit être couvert,...).

UML sert surtout pour un développement structuré d'une application complexe, orientée-objets, distribuée. STEP appartient au domaine de la Technologie des Données Produits et

donc est adapté à la description des bases de données.

Par contre il existe dans STEP une partie normalisée pour traduire les données en XML ([38] et [37]. Il existe également des propositions de standards d'échanges entre UML et XML.

3 Evolutions en Cours

Les trois normes décrites dans la section précédente sont encore en évolution. Actuellement, avec l'avancée d'Internet, SGML et XML prennent de plus en plus d'ampleur dans le domaine de la présentation et l'échange de documents. Du côté d'UML, le Génie Logiciel continue également à mettre en place de nouvelles méthodes de travail et d'échange de spécification. Quant à STEP, de nouvelles parties s'ajoutent et un nombre impressionnant d'acteurs industriels sont maintenant impliqués dans les spécifications de la norme. Ainsi les trois organismes responsables, le W3C pour XML, l'OMG pour UML et l'ISO pour STEP interagissent de plus en plus les uns avec les autres et semblent évoluer vers le même but : définir une méthodologie qui permettent d'échanger des données de manière distribuée, évolutive, à la fois globale et spécifiable pour chaque métier, sans perte d'information entre des systèmes informatiques complètement différents.

3.1 XML-schema

Deux problèmes remettent actuellement en cause le format DTD. Le premier, c'est qu'il n'est pas lui même au format XML, ce qui peut être gênant car pour écrire une DTD, il faut prévoir un outil différent de celui utilisé pour écrire du XML. Le deuxième problème, peut être encore plus important, est qu'il est impossible de typer un élément. Si on peut dire qu'un élément est obligatoire ou non, il est par contre impossible de définir son type : nombre, texte, date, etc...Cela peut induire de nombreuses erreurs indétectables à la compilation mais préjudiciable à l'utilisation. C'est pourquoi le W3C travaille actuellement sur une norme appelée XML-data, qui apporterait les mêmes services que les DTD, et plus encore. Cette norme permet de spécifier de manière fine les éléments (texte, chiffres, date, etc...) et la structure (manière de placer ces éléments), et tout cela sous une forme XML ce qui permettra de rédiger un XML-data avec le même outil que celui utilisé pour rédiger un fichier XML. Le W3C a donc émis une demande, et pour l'instant, la proposition la plus avancée pour répondre à ce besoin a été proposée par Microsoft sous l'appellation XML-schema. C'est pourquoi il est possible d'entendre parler soit de XML data, qui est la demande de proposition du W3C, soit de XML schema qui est la réponse à cette demande faite par Microsoft. Au final, il y a de fortes chances que ce soit la proposition de Microsoft qui soit adoptée. Une fois validée par le W3C, cette proposition deviendra une nouvelle norme qui sera intégrée à XML.

3.2 XMI et OCL

Le XMI, eXtended Model Interchange, proposé par l'Object Management Group est l'un des développements les plus récents pour la communauté des développeurs de l'UML. XMI est un format d'échange qui a la capacité d'autoriser le partage en continu des modèles entre

les outils de développement les meilleurs de la nouvelle génération. Par exemple, plutôt que d'écrire les scripts dans un outil de modélisation UML pour créer des rapports, un utilisateur pourrait exporter le modèle en cours de conception en utilisant XMI et simplement importer le modèle dans un outil spécialisé en écriture de rapports. De plus, puisque XMI utilise XML pour représenter l'information du modèle, une famille de solutions XML est disponible immédiatement, telles que les feuilles de style XSL pour la présentation basée sur le navigateur et les outils d'interrogation XQL pour les fonctions de recherche.

XMI définit entre autre un langage de contraintes (l'Object Constraint Language) pour mieux spécifier la sémantique des modèles UML. La description des contraintes était jusqu'à présent laissé à la charge du développeur.

Le standard XMI est complexe et la résolution des nombreux problèmes de compatibilité inévitables prendra du temps avant que son déploiement ne se fasse sur une large échelle. Cependant, depuis que XMI est développé par les poids lourds de l'industrie IBM et Unisys, entre autres entreprises de pointe, il est prévisible que les produits utilisant ce standard sortent dans un délai assez court.

3.3 Vers une norme unique ?

Les trois méthodes de modélisation à laquelles nous nous interressons sont en évolution (xml-schema pour xml, xmi pour uml, Express pour STEP) et envisagent de créer des liens les uns avec les autres. Des ponts entre STEP et XML (ISO 10303-28) existent déjà. Il en est de même du côté d'UML. De plus, de nombreux travaux sont en cours pour créer des ponts entre STEP (Express, Express-G) et UML.

Ainsi, toutes les représentations semblent converger vers une même représentation qui serait alors un consensus entre toutes les disciplines qui nécessitent un échange d'informations. Les caractéristiques d'une telle norme seraient alors issues de la fusion de toutes les normes existantes, c'est à dire principalement :

- séparation entre donnée, modèle de données, représentation des données
- nécessité d'un langage graphique de description
- utilisation intensif de l'orienté objet
- objets métiers
- modélisation de la dynamique des données

4 Motivations pour ce travail

Dans ce travail, nous nous plaçons du côté de la conception de logiciels de simulation en Electromagnétisme. Ces logiciels servent pour l'étude de la vue électromagnétique de produits tels que des machines électriques.

Nous utilisons STEP non seulement comme un moyen d'échanger des données mais aussi et surtout comme une méthodologie pour décrire de manière structurée et d'échanger efficacement

des connaissances. En effet, l'ensemble des parties de STEP, résumé sur une page dans l'Annexe B, est un ensemble colossal de savoirs et de savoir-faire. Il y a de quoi faire une grande base de données cohérente capable de contenir des données provenant de domaines aussi variés que la mécanique des structures, la mécanique des fluides, l'aéronautique, la cinétique, la thermique tous rangés chacun dans leur domaine mais dont toutes les parties communes sont mises en évidence.

Il semble donc intéressant d'étudier cette méthodologie et d'en acquérir le savoir-faire pour ensuite pouvoir y placer la vue électromagnétique d'un produit. Le savoir-faire de STEP inclut la modélisation de données produit et l'intégration de modèles dont les sémantiques diffèrent. Elle inclut également des méthodes d'implantation dont bien sûr la traduction vers un langage informatique et les méthodes du Génie Logiciel orienté objet. Le savoir-faire en conception en électromagnétisme inclut les méthodes numériques, des données spécifiques à un domaine particulier de la physique. La rencontre de ces deux savoir-faire peut aboutir à des logiciels de simulation capables d'échanger avec d'autres logiciels de simulation, dans d'autres domaines, sur des parties pourtant communes : la géométrie, la représentation, les matériaux. Cela permet ainsi à chacun de se concentrer sur son propre domaine exclusivement.

5 Conclusions

Dans ce chapitre, nous avons tout d'abord voulu montrer que dans la modélisation de données, il existe différentes techniques et plusieurs notations mais que leur évolution tend vers un même but, à savoir, la définition d'une méthodologie qui permette d'échanger des données de manière distribuée, évolutive, à la fois globale et spécifiable pour chaque métier, sans perte d'information entre des systèmes informatiques complètement différents. Puis nous avons expliqué notre position dans le choix d'une de ces méthodes. A l'heure actuelle nous pensons que la méthodologie STEP est un bon départ pour la spécification des formats de données nécessaires au domaine de la simulation en électromagnétisme.

Chapitre II

La Méthodologie STEP

STEP a pour ambition de normaliser toutes les manières possibles de décrire un quelconque produit. Le volume des informations qu'il faut normaliser est donc proportionnel au nombre de produits existants multiplié par le nombre de domaines dans lesquels chaque produit est défini. Cela constitue donc une grande quantité de données qu'il faut maintenir et gérer de façon cohérente. Cette constatation a conduit les initiateurs de STEP à définir des méthodes basées sur la modélisation orientée objets aussi bien pour la structuration des données de la norme que pour son contenu. Actuellement, STEP possède une définition précise, une structure globalement stable ou qui tend à le devenir et de nombreux outils dont un langage de description (EXPRESS), et un langage de notation graphique (EXPRESS-G) conçus pour être compréhensibles sans avoir recours à des notions informatiques avancées.

Ce chapitre présente la méthodologie de modélisation et d'échange de données conçue par les développeurs de STEP et proposée à ses utilisateurs.

La première section définit les principes retenus, c'est-à-dire les fonctionnalités que la norme doit pouvoir offrir. La deuxième section montre comment ces principes ont été appliqués pour la construction des principales *classes* de STEP. La troisième section décrit les langages EXPRESS et EXPRESS-G. La dernière section décrit le développement d'un *Protocole d'Application* comme une méthode efficace de création d'un consensus entre des acteurs internationaux d'un même domaine.

1 Principes de STEP

Le problème consiste à créer un ensemble de conventions qui permette la description d'un produit indépendamment de son évolution dans son cycle de vie, capable de couvrir tous les domaines d'application susceptibles d'utiliser ce produit de telle sorte que ces domaines se comprennent entre eux et puissent échanger chaque donnée sans jamais altérer ou simplifier les informations échangées mais avec une quantité minimale de supports physiques. Le cahier des charges imposé pour le développement de la norme comporte donc un très grand nombre de contraintes dont les principales sont détaillées dans cette section.

1.1 Description d'un produit tout au long de son cycle de vie

Les modèles de données normalisées ne doivent pas dépendre du cycle de vie du produit. Cette contrainte signifie que les modèles de données qui seront normalisées par STEP doivent permettre de considérer la description d'un produit depuis sa conception du produit jusqu'à sa commercialisation et son suivi après vente.

1.2 Des vues pour chaque domaine d'application

Il n'est pas envisageable pour un domaine d'étude particulier d'utiliser toutes les propriétés d'un produit. Le volume d'information à mettre en place serait inutilement trop grand. Une simplification de la réalité est donc toujours opérée. Cette simplification peut porter sur différentes propriétés du produit, créant ainsi de multiples modèles pour un même produit. Chaque personne ou chaque logiciel a ainsi une *vue* du produit qui lui est propre et qui dépend de l'usage auquel est destiné le modèle. Cette *vue* ne comporte que certaines propriétés du produit réel, cachant celles qui ne lui sont pas nécessaires. Par exemple, un logiciel de calcul des structures d'un bâtiment ne tiendra pas compte des propriétés électrostatiques des matériaux qui le composent.

Pour couvrir tous les domaines d'application, STEP doit donc définir une *vue* adaptée à chaque domaine utilisateur du produit. C'est pourquoi il a été décidé de diviser STEP en plusieurs *parties*. Chaque *partie* de STEP couvre un domaine particulier (dessin, gestion, configuration, mécanique,...). Une *partie* agit comme un filtre dont l'utilisateur de la norme se servira pour modéliser le produit de telle manière que seules les propriétés qui l'intéressent apparaissent (Fig. 2.1).

Pour ensuite rendre possible le partage des données entre tous les domaines d'application, la norme doit définir un langage commun, EXPRESS et EXPRESS-G (voir Section 3) : la norme STEP permet des communications entre des utilisateurs spécialistes dans leur propre domaine mais ne connaissant pas le domaine des autres parce qu'elle définit des vues pour chaque domaine dans un langage commun à tous.

1.3 Des *parties* communes à tous les domaines d'application

Les *vues* peuvent avoir des zones communes. La superposition de toutes les *vues* fait alors apparaître un noyau dans lequel les structures de données sont génériques, c'est-à-dire communes à toutes les vues de tous les produits. Ce noyau contient, par exemple, les structures de

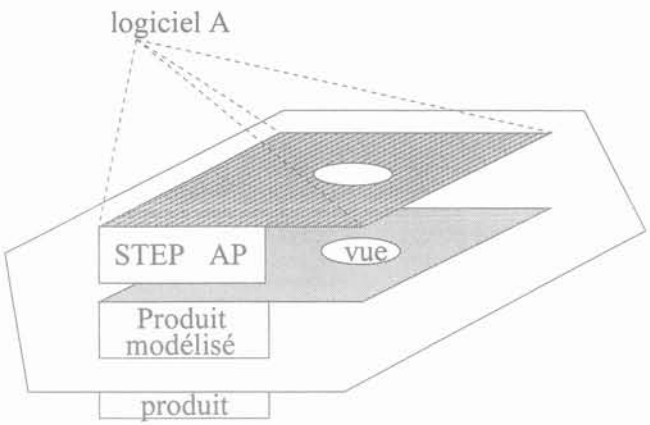


FIG. 2.1 – Une vue du produit

données nécessaires pour décrire la géométrie du produit : tous les produits ont une géométrie et chaque élément de la géométrie d'un produit doit pouvoir être décrit par STEP. Les *parties* génériques sont destinées à être implantées dans chaque logiciel conforme à la norme.

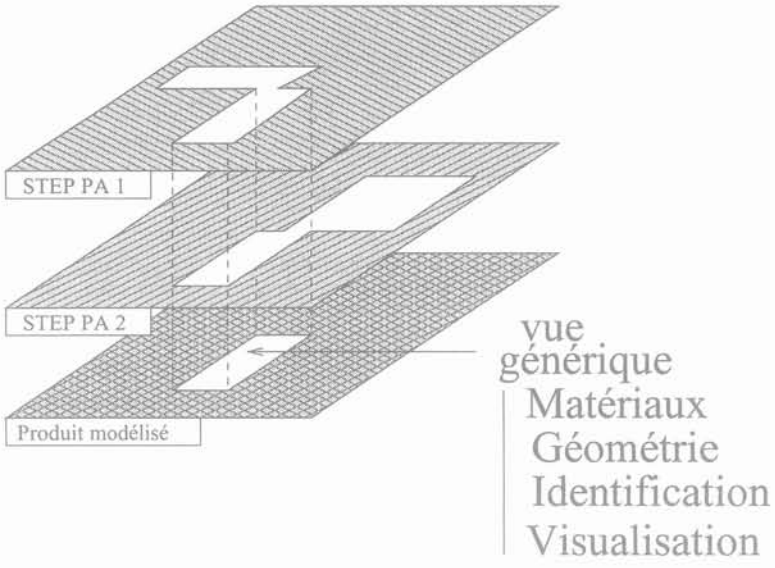


FIG. 2.2 – Une vue générique

Il est utile de séparer la *vue* générique des autres *vues* pour éviter les redondances d'information : les autres vues n'auront plus à la définir mais peuvent y faire référence si besoin est.

1.4 Echanges de données et partage de données

Les *échanges de données* se font par l'intermédiaire d'un format neutre tel que le format physique décrit dans STEP ([35]).

Mais en plus de ces échanges de données, STEP permet le *partage de données*. Les partages de données se font grâce à une base de données commune et un standard d'accès aux données de la base. Par l'intermédiaire de la partie 22 de STEP : Standard Data Access Interface ([36]), il est possible à des applications d'accéder de manière normalisée à des données de la base.

Les échanges de données, tout comme les partages de données permettent de diminuer le nombre d'interfaces à développer. En effet, si n logiciels veulent échanger des données entre eux, chacun devra mettre en place n-1 interfaces d'échange et au total, il faudra n(n-1) interfaces : c'est le problème des "4 et plus", bien connu dans dans le domaine de l'interconnexion des réseaux (voir Figure 2.3). S'ils utilisent une base de donnée commune, chaque logiciel ne communique qu'avec la base commune : il ne faudra que 2n interfaces. La deuxième méthode devient avantageuse en terme de nombre d'interfaces à développer dès que n est supérieur à 4.

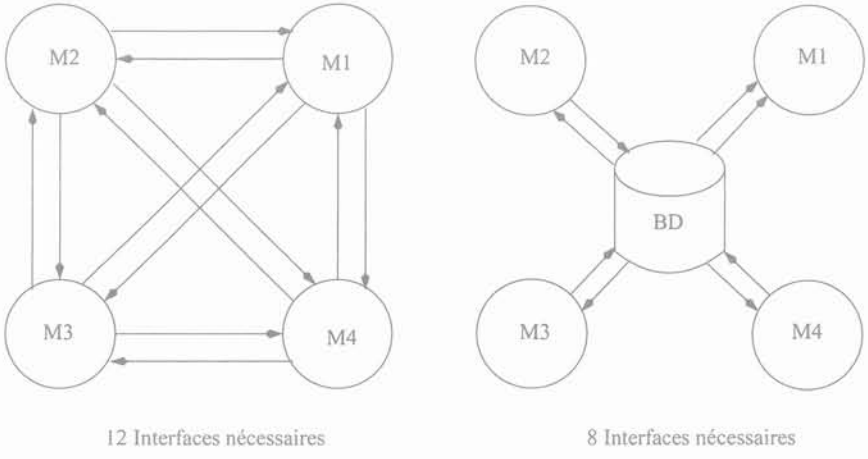


FIG. 2.3 – Echanges centralisés

Parce que la norme STEP a des *parties* génériques complétées par des *parties* lues particulières, elle permet l'utilisation d'une base de données commune : la base de données sera décrite par les *parties* génériques et sera accédée par les *parties* particulières à chaque domaine d'application.

L'exemple Fig. 2.4 montre un exemple de structure possible d'une entreprise dans laquelle les différents secteurs (Conception Assistée par Ordinateur, Approvisionnement et Système de Gestion des Données Techniques) peuvent partager les données d'un produit (ses dimensions, les matériaux qui le composent, son prix, sa classification) à travers des *parties* de STEP qui gèrent l'accès à la base de données, laquelle contient les informations concernant le produit, décrit par les *parties* génériques.

2 Principales *classes* de STEP

Les structures orientées objets permettent de répondre aux exigences définies précédemment. En effet l'encapsulation des données et l'utilisation de l'héritage/dérivation permettent

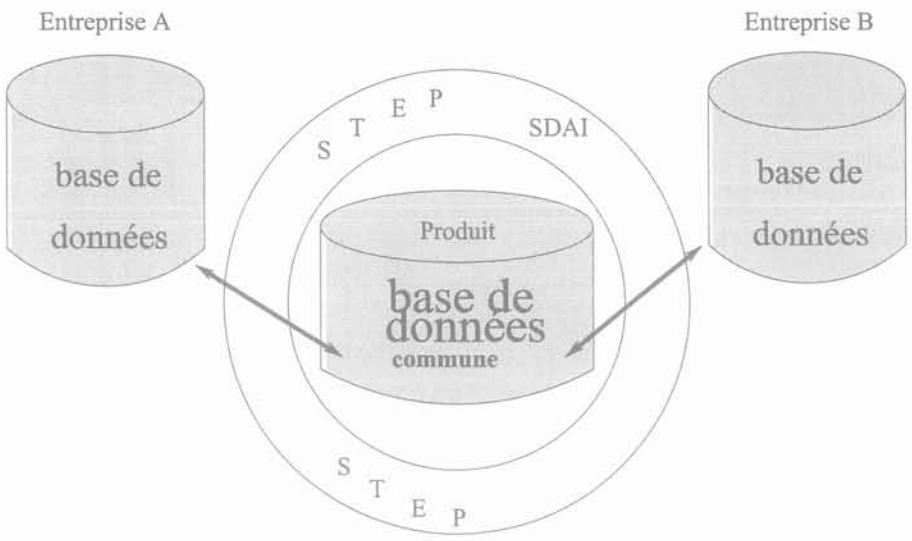


FIG. 2.4 – Echanges par base de données commune

de définir des nouvelles structures à partir de celles existantes (définition de nouvelles *vues* à partir de la *vue* générique sans modification de cette dernière). Il a été vu précédemment que STEP devait être divisée en *parties*. Pour bien définir le rôle de chacune de ces *parties*, elles ont été rassemblées en sept *classes* suivant le degré de spécialisation des *parties*.

Après une description de la vue d'ensemble, cette section résume le contenu des *classes* de STEP : Introduction, Ressources d'Information Intégrées, Méthodes de Description, Méthodes d'Implantation, Méthodes de Tests de Conformité, Suite de Tests Abstraits, Protocoles d'Application.

L'Annexe B donne le contenu de toutes les *classes* de STEP ainsi que les intitulés des *parties* et leur niveau de normalisation.

2.1 Vue d'ensemble

La Figure 2.5 montre la division de STEP en *classes* et la Figure 2.6 montre des exemples de la répartition des *parties* de STEP dans ces *classes*.

La *classe Protocole d'Application* représente la vue la plus externe de STEP. C'est la vue la plus proche de l'utilisateur, celle qu'il peut comprendre sans nécessité d'avoir recours à un système informatique. Les langages qui y sont utilisés sont les diagrammes IDEF0 (ou SADT) et le langage "usuel".

La *classe* Méthodes d'Implantation est le niveau le plus interne du système d'information. Elle concerne le stockage physique des données et leur accès. Elle s'adresse donc à ceux qui vont programmer et mettre en oeuvre le modèle.

La *classe* des Ressources d'Information Intégrée contient la vue conceptuelle des produits. Elle fait la liaison entre les *vues* précédentes.

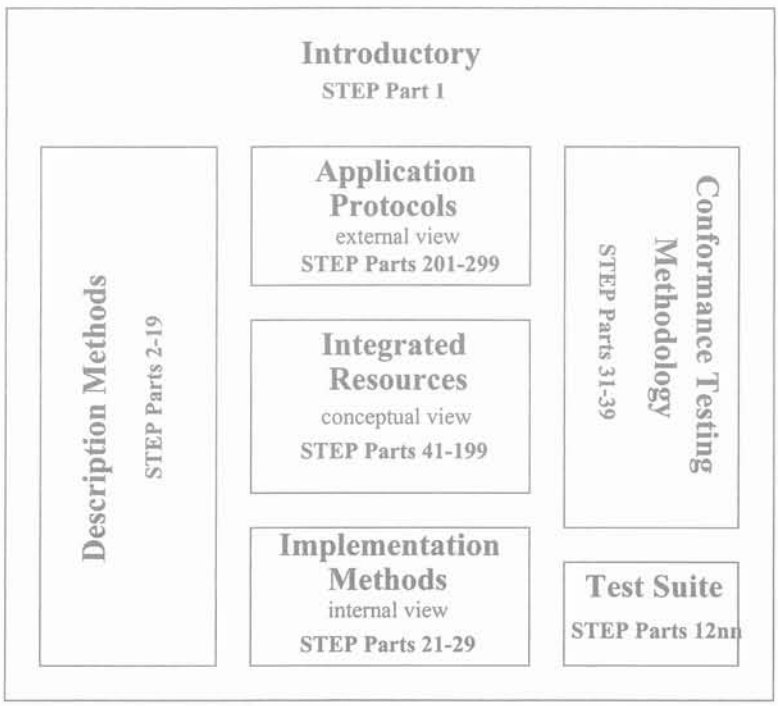


FIG. 2.5 – *Classes* de STEP

2.2 Ressources d'Information Intégrées

Cette *classe* est elle même divisée en *sous-classes*, suivant la généricité des *parties* qu'elle contient :

- les *Ressources d'Information Génériques*
- les *Ressources d'Application Intégrées*
- les *Constructions d'Applications Interprétées*
- les *Modules d'Application*

2.2.1 Ressources d'Information Générique

Cette *sous-classe* est le coeur de STEP : elle est la plus générique de toutes les *classes* et *sous-classes*. Elle contient les modèles de données communs à tous les produits. Ce sont les descriptions des données concernant par exemple la géométrie (points, courbes, surfaces, volume), la topologie (arêtes, chemins), les mathématiques (vecteurs, matrices), la représentation (label, description), les matériaux (désignation), les tolérances...

2.2.2 Ressources d'Application Intégrées

Un peu moins générique que la *sous-classe* des *ressources génériques*, la *sous-classe* des *ressources d'application intégrées* reste néanmoins cohérente, c'est à dire que l'intersection de

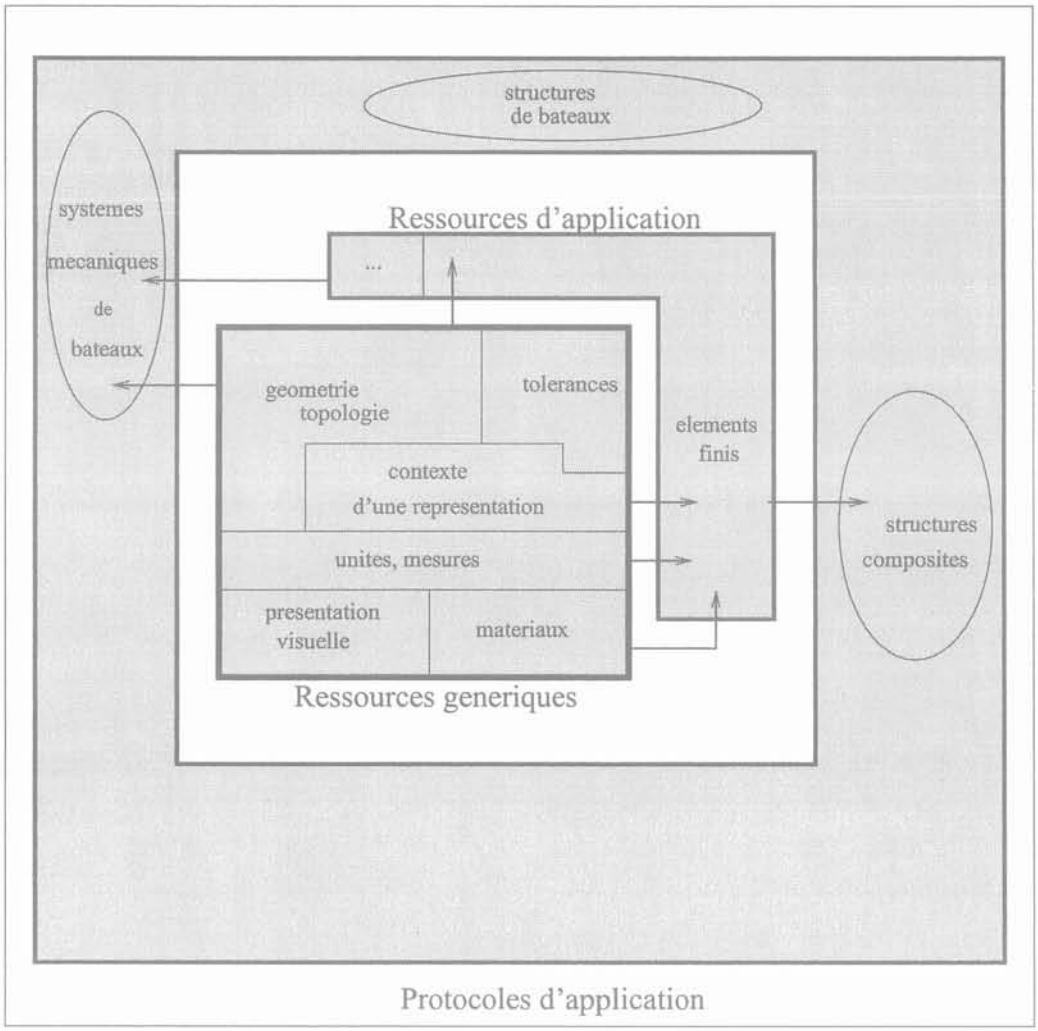


FIG. 2.6 – Exemples de *parties* de STEP

deux *parties* quelconques est vide. Elle contient des *parties* qui sont déjà orientées vers un ou plusieurs domaine particuliers. Ces *parties* sont par exemple les éléments finis (noeuds, éléments), la cinématique...

2.2.3 Constructions d'Applications Interprétées

Cette *sous-classe* a été créée pour maintenir la cohérence entre deux ou plusieurs Protocoles d'Application. Les *parties* qui y sont rangées sont des intersections d'au moins deux Protocoles d'Application. Un *Protocole d'Application* qui référence une construction appartenant à l'une de ces *parties* doit référencer également toute la *partie*. Ainsi, les domaines, représentés chacun par des Protocoles d'application, qui ont des définitions de données identiques ont également une description STEP commune. Cela permet de mettre en valeur les intersection de domaine d'application.

2.2.4 Modules d'Application

Un Module d'Application est un morceau de *Protocole d'Application*. Il est donc spécifique à un domaine particulier.

Les Modules d'Application ont été introduits très récemment dans STEP. Ils proviennent du problème d'intégration des Protocoles d'Application. En effet, les AIC (Constructions d'Application Interprétées) étaient toujours construits après l'intégration d'un nouveau *Protocole d'Application* et obligeaient la modification de tous les Protocoles d'Application qui avaient une intersection non nulle avec le nouveau *Protocole d'Application*.

Le développement modulaire des Protocoles d'Application devrait éviter de faire appel aux AIC.

2.3 Méthodes de Description

Cette *classe* contient les *parties* qui décrivent les langages utilisés dans STEP. Par exemple, la *partie* 11 décrit le langage Express et sa notation graphique Express-G et plus récemment le langage d'échanges Express-X. Toutes ces *parties*, comme l'indique le nom de la *classe*, doivent servir à décrire le modèle et concernent donc à la fois la méthodologie de description et les langages et notations utilisées.

2.4 Méthode d'Implantation

Les Méthodes d'Implantation spécifient comment un modèle STEP doit être implanté dans un logiciel. Cette *classe* contient entre autre la *partie* 21 pour le format neutre, la *partie* 22 pour l'interface générique d'accès aux données et les *parties* 23 à 29 pour la traduction de la *partie* 22 dans un langage particulier. Cette *classe* sera mieux détaillée dans le chapitre IV.

2.5 Méthodes de Tests de Conformité

Cette *classe* spécifie les installations requises pour un laboratoire de test de conformité STEP ainsi que les tests "abstraits", c'est à dire faisables "à la main", pour les implantations des *parties* 21 et 22.

2.6 Suite de Tests

En ce qui concerne les tests, il est à remarquer que chaque *Protocole d'Application* doit contenir également une batterie de tests qui doivent servir à vérifier qu'une implantation logicielle est conforme au *Protocole d'Application*. Cette batterie de tests a un numéro correspondant au numéro du *Protocole d'Application* avec un 3 à la place du 2 : la suite de tests du *Protocole d'Application* 203 est la *partie* 303.

2.7 Protocoles d'Application

C'est la *classe* la plus spécialisée de STEP. Chaque *partie* de cette *classe* est spécifique à un domaine ou même à une *partie* d'un domaine. L'activité dans laquelle est utilisée le protocole

est également définie dans la *partie*, à l'aide du formalisme IDEF0.

3 Langages utilisés dans STEP

La section précédente a présenté la nécessité d'avoir un langage commun pour les échanges. Cette section présente les deux langages introduits par STEP : le langage de description EXPRESS et la notation graphique EXPRESS-G. Le modèle que nous proposons dans le chapitre suivant sera décrit à l'aide de ces deux représentations.

3.1 Le langage EXPRESS

EXPRESS est un langage de description de données créé sur la base de plusieurs exigences définies dans le cadre ISO TR 9007, au milieu des années 80. Il est structuré en *schémas*, qui représentent le domaine d'étude d'un produit. Le *schéma* est constitué d'un ensemble de déclarations dont les plus importantes sont les *entités*. Dans les *entités*, sont définis des attributs et des contraintes qui restreignent les valeurs que les attributs peuvent prendre. Un *schéma* peut également comporter des *fonctions*, des *procédures* et des *règles*. La description complète du langage est donnée dans le document ISO 10303-11, *Description Methods : EXPRESS Language Reference Manual*. Les documents [24] et [4] donnent également une description du langage.

3.1.1 Exemple d'un fichier EXPRESS

L'exemple suivant, donné dans la part ISO 10303-21, montre une forme simple de fichier EXPRESS suivi d'un fichier neutre d'instances.

Fichier EXPRESS :

```
SCHEMA example_geometry;

TYPE length_measure=NUMBER;
END_TYPE;

ENTITY geometry
SUPERTYPE OF (ONEOF(point));
END_ENTITY;

ENTITY point
SUPERTYPE OF (ONEOF (cartesian_point))
SUBTYPE OF (geometry)
END_ENTITY;

ENTITY cartesian_point
SUBTYPE OF (point);
x_coordinate : length_measure;
y_coordinate : length_measure;
z_coordinate : OPTIONAL length_measure;
END_ENTITY;

END_SCHEMA;

SCHEMA example_topology;

REFERENCE FROM example_geometry;

TYPE edge_or_logical = SELECT (edge, edge_logical_structure);
END_TYPE;

ENTITY topology
SUPERTYPE OF (ONEOF(vertex, edge, loop));
END_ENTITY;

ENTITY vertex
SUBTYPE OF (topology);
vertex_point : OPTIONAL point;
END_ENTITY;
```

```
ENTITY edge
SUBTYPE OF (topology);
      edge_start      : vertex;
      edge_end        : vertex;
END_ENTITY;

ENTITY edge_logical_structure;
      edge_element    : edge;
      flag            : BOOLEAN;
END_ENTITY;

ENTITY loop;
SUPERTYPE OF (ONEOF(edge_loop))
SUBTYPE OF (topology);
END_ENTITY;

ENTITY edge_loop
SUBTYPE OF (loop);
loop_edge : LIST [1 :?] OF edge_or_logical;
END_ENTITY;

END_SCHEMA;
```

Les noms utilisés dans un fichier EXPRESS sont souvent longs mais, dans un fichier neutre STEP, il existe des abréviations qui allègent l'écriture. Ces abréviations ont un code précis déterminé par STEP. Dans cet exemple, les noms du modèle EXPRESS seront remplacés par des abréviations :

cartesian_point cpt

vertex vx

edge ed

edge_logical_structure ed_strc

edge_loop ed_loop

Le fichier suivant montre la structure de quelques instances du modèle EXPRESS précédent et donne un exemple de structuration d'un fichier neutre STEP.

```
ISO/DIS 10303-21
'SUPER CIM SYSTEM RELEASE 4.0',
'APPROVED BY JOE BLOGG');
FILE SCHEMA ('EXEMPLE_GEOMETRY','EXEMPLE_TOPOLOGY);
END_SEC;
DATA;
/* THE FOLLOWING 13 ENTITIES REPRESENT A TRIANGULAR EDGE LOOP
*/
#1=CPT(0.0,0.0,0.0);
#2=CPT(0.0,1.0,0.0);
#3=CPT(1.0,0.0,0.0);
#11=VX(#1);
#12=VX(#2);
#13=VX(#3);
#16=ED(#11,#12);
#17=ED(#11,#13);
#18=ED(#13,#12);
#21=ED_STRC(#17, .F.);
#22=ED_STRC(#18,.F.);
#23=ED_STRC(#16,.T.);
#24=ED_LOOP(#21, #22, #23);
ENDSEC;
END-ISO-10303-21.
```

Cet exemple montre la mise en oeuvre des *schémas* (SCHEMA) , *types* (TYPE), *entités* (ENTITY), la déclaration d'un héritage (SUBTYPE), l'appel à des *entités* et *types* provenant d'un autre *schéma* (REFERENCE FROM).

Une autre déclaration dont il faut également tenir compte est celle des *règles locales* (WHERE) et *règles globales* (RULE). Ces règles assurent la cohérence entre les *attributs* et les *entitiés*. On aura par exemple :

```
ENTITY nombre_complexe;
```

```
    partie_reelle : REAL;
    partie_imaginaire : REAL;
END_ENTITY;
ENTITY imaginaire_pur
SUBTYPE OF nombre_complexe;
WHERE self\nombre_complexe.partie_reelle=0;
END_ENTITY;
```

EXPRESS n'est pas un langage de programmation dans la mesure où il n'y a pas de compilateur EXPRESS ni de code EXPRESS exécutable. Du fait qu'il est tout d'abord un langage de description, il ne contient aucun élément permettant de manipuler des données (pas d'entrée, pas de sortie, peu de calcul). Cependant, il contient un très grand nombre d'éléments qui permettent de définir un objet sans ambiguïté. De plus, EXPRESS est conçu pour être compris à la fois par les humains et par les ordinateurs. Il y a donc possibilité de développer des logiciels qui lisent les définitions d'EXPRESS et construisent à partir de ces définitions des bases de données dans un format spécifié.

3.2 La notation EXPRESS-G

La notation EXPRESS-G est un sous-ensemble du langage EXPRESS. Elle permet de décrire un modèle sous forme graphique. La plupart des déclarations EXPRESS présentées dans la sous-section précédente *(schéma, type, entité,...)* sont représentables par le formalisme EXPRESS-G. Il est à noter que les *règles* ne sont pas explicités entièrement : une étoile placée sur la relation signale qu'il existe des contraintes sur celle-ci.

La forme graphique a l'avantage de ne pas introduire la notion de programmation, ce qui permet aux utilisateurs non informaticiens de la norme STEP de concevoir et spécifier leur modèle de manière simple.

EXPRESS-G est basé sur une modélisation orientée objet. Il utilise les concepts graphiques d'entités-relations : les objets sont représentés dans des "boites" reliées entre elles par des "flèches". Les boites représentent un objet. Les flèches représentent les relations d'attribut (traits fins, en pointillé si l'attribut est en option) ou d'héritage (traits épais) entre les objets. Un cercle vide indique le sens des flèches.

3.2.1 exemple

Le *schéma* ***example_geometry*** décrit en EXPRESS dans la sous-section précédente sera représenté Figure 2.7.

4 Développement d'un *Protocole d'Application*

Dans la méthodologie STEP, le développement d'un *Protocole d'Application* est assuré par une équipe de spécialistes de la norme et de spécialistes du domaine couvert. Ce développement suit le processus assez lent et rigide d'édition des normes ISO pour que tous les intéressés soient

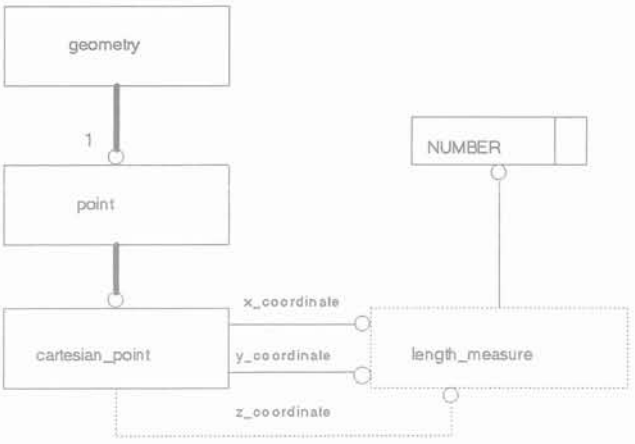


FIG. 2.7 – Schéma example_geometry

informés et puissent avoir l'opportunité de faire des remarques et demander des modifications.

La Figure 2.8 résume les principales étapes de ce développement. Ces étapes sont décrites dans la sous-section suivante.

4.1 Etapes du développement d'un *Protocole d'Application*

4.1.1 Expression d'un besoin

Des industriels d'un même domaine se mettent d'accord sur leurs besoins et produisent un document appelé "Planning Project". Ce document est étudié par le SC4 (Sub Committee 4) de l'ISO pour vérifier que les besoins n'ont pas encore été couverts par des Protocoles d'Application existants, que les moyens, le nombre d'industriels et le nombre de pays concernés sont suffisants. Le formalisme utilisé pour décrire le domaine est IDEF0 (connu également sous le nom SADT).

4.1.2 Correspondances avec STEP

Les initiateurs du projets et les experts STEP se mettent d'accord sur la définition finale du domaine et identifient le nombre de Protocoles d'Applications nécessaire pour le couvrir. Ils fournissent alors une proposition de projet qui devra être validée par le PMAG (Project Management Advisory Group). Cette proposition n'est complète qu'après une enquête auprès des professionnels du métier et une revue des objectifs et domaines couverts par le futur *Protocole d'Application.*

4.1.3 Planning de développement

Ce document définit les étapes nécessaires pour le développement du Protocole d'Application.

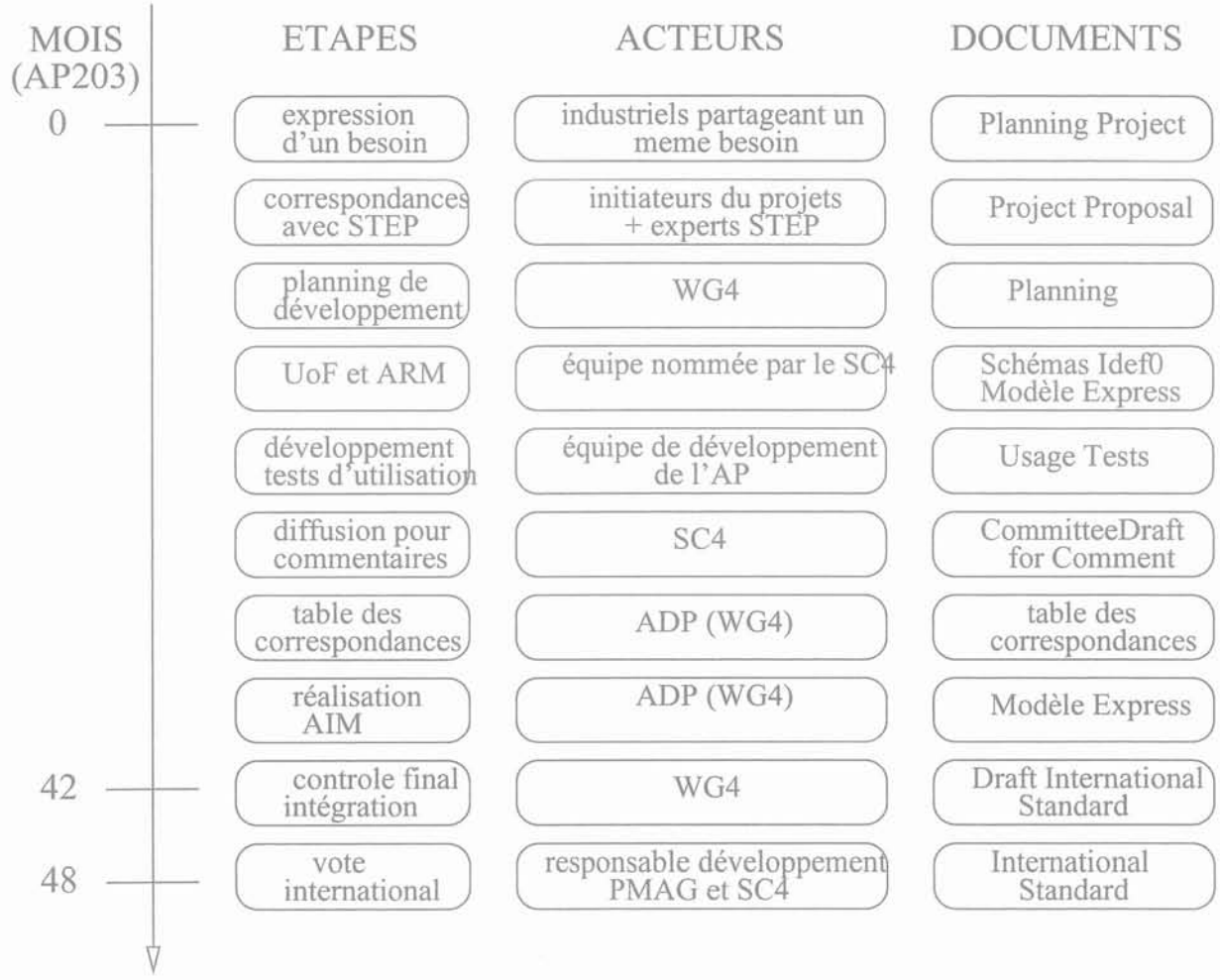


FIG. 2.8 – Etapes de développement d'un *Protocole d'Application*

4.1.4 UoF et ARM

L'UoF (Unit of Functionality) définit une fonctionnalité du Protocole d'Application. Chaque structure du futur *Protocole d'Application* devra appartenir à l'une des UoF définies. L'ARM (Application Reference Model) est le modèle décrit avec des termes propres au domaine. Il est décrit en Express.

4.1.5 Développement des tests d'utilisation

Chaque *Protocole d'Application* doit fournir des scénarios d'utilisation et des cas tests pour valider son implantation. Les tests sont définis dans cette étape.

4.1.6 Diffusion pour commentaires

Le premier document diffusé pour commentaire est le Committee Draft for Comment. C'est un document qui permet une revue élargie du cahier des charges du *Protocole d'Application*. La diffusion est faite à chaque délégation nationale. Un délai de quatre mois est donné avant le début de l'étape suivante.

4.1.7 Table des correspondances

L'intégration du modèle ARM passe par l'identification des constructions qui se trouvent déjà dans les ressources génériques. Pour cela, le WG4 construit une table des correspondances. Cette table des correspondances n'est pas forcément une correspondance un à un dans un sens ou dans l'autre. En effet, une construction de l'ARM peut être divisée en plusieurs constructions dans les ressources génériques. L'inverse est aussi possible.

4.1.8 Réalisation de l'AIM

L'AIM (Application Interpreted Model) est le modèle Express final. Il tient compte des ressources génériques pour qu'il n'y ait pas de duplication de structures.

4.1.9 Contrôles final de l'intégration

Après un contrôle technique fait par le responsable du groupe de travail, le comité d'édition vérifie la rédaction et la présentation du document. Le document peut alors devenir une norme internationale expérimentale (Draft International Standard)

4.1.10 Vote international

Six mois après l'étape précédente, les commentaires internationaux sont pris en compte, la norme expérimentale est à nouveau soumise à un vote pour devenir une norme internationale (International Standard).

4.2 Intersection entre Protocoles d'Application

La construction de STEP a été faite de telle sorte qu'aucune structure ne soit dupliquée. Or deux Protocoles d'Application peuvent parfois avoir une intersection non nulle. Une gestion correcte de ces intersections est nécessaire pour éviter à la fois la duplication inutile de données identiques et une réutilisation optimale des travaux déjà effectués. Cette gestion doit tenir compte à la fois de la structure actuelle de STEP (les *parties* existantes) et le développement futur ou en cours de la norme. La première prise en compte de ces intersections est la division en *classe*. Ainsi, les *parties* de la *classe* Applications Génériques sont des *parties* communes à presque tous les Protocoles d'Application, les *parties* de la *classe* Application Intégrées sont des *parties* plus spécialisées. Mais il est parfois des cas où deux Protocoles d'Applications avaient des *parties* communes qui ne pouvaient pas être une *partie* entière de la *classe* Application Générique ou Application Intégrée. La *classe* Application Interpreted Construct contient ces

parties communes à deux ou plus Protocoles d'Applications. Enfin, nous mentionnerons également parmi les travaux les plus récents menés sur la norme, la possibilité de ne développer que des Protocoles d'Application modulaires dont chaque module appartiendrait à la *classe* des Modules d'Application.

4.2.1 Application Interpreted Construct

L'AIC est la *classe* de STEP qui comporte les *parties* 500 à 600. Un AIC contient des structures de données communes à deux ou plusieurs Protocoles d'Application. La construction d'une *partie* AIC se fait donc pendant l'intégration d'un nouveau *Protocole d'Application*. Cette construction se fait au niveau du modèle Express final après intégration (le modèle AIM). Si une AP référence une AIC, elle doit prendre tout le modèle AIC.

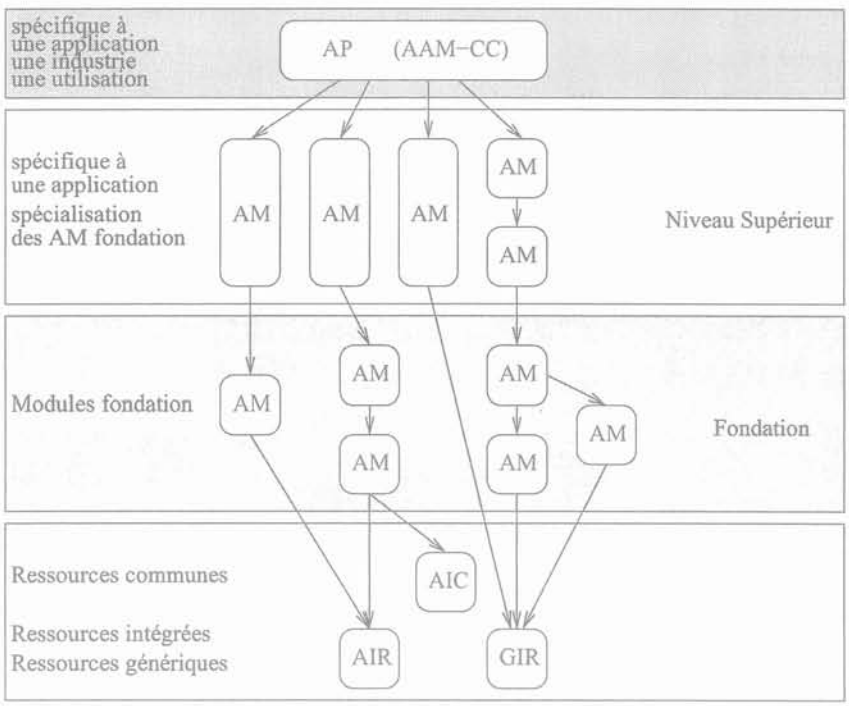


FIG. 2.9 – Application Module

4.2.2 Application Module

Les Modules d'Application sont des *parties* communes à deux *parties* de STEP (même si ces deux *parties* ne sont pas des Protocoles d'Application), comme montré sur la Figure 2.9. Les AM ne concernent pas uniquement le niveau AIM d'un *Protocole d'Application* mais aussi au

niveau ARM. Ainsi, deux Protocoles d'Application qui ont un module d'application commun auront en commun également le vocabulaire propre aux deux métiers.

Le développement de Protocoles d'Application modulaires est en cours de réalisation. Mais, à terme, un *Protocole d'Application* ne sera qu'un ensemble d'AM. la Figure 2.10 donne l'exemple de l'approche modulaire du *Protocole d'Application* 203 : Configuration-Controlled Design.

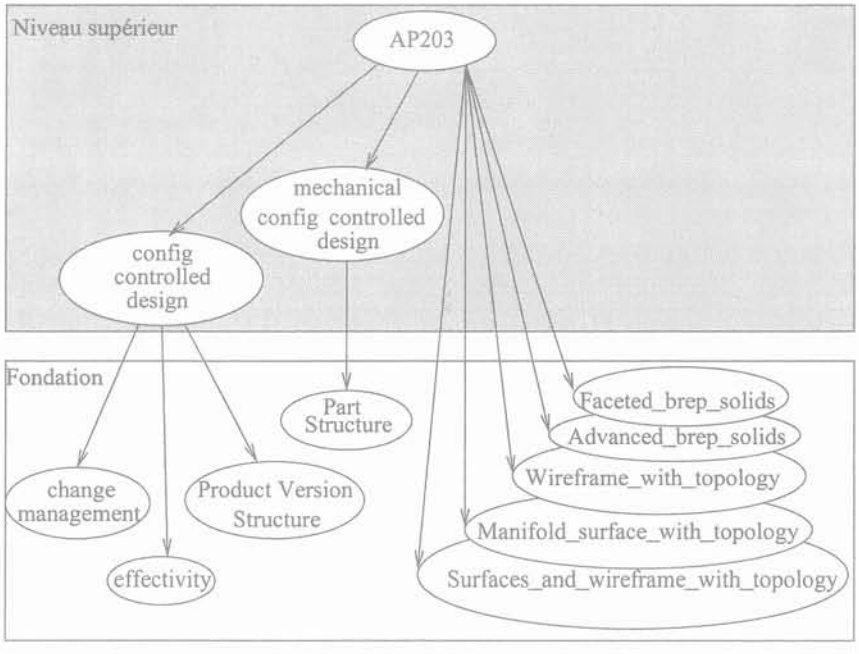


FIG. 2.10 – Approche modulaire de l'AP203

Le problème avec les AIC était que leur construction se faisait après la publication du *Protocole d'Application* et au niveau de l'AIM. Cela imposait donc de modifier les Protocoles d'Applications qui ont une *partie* commune avec le nouveau *Protocole d'Application* créé pour qu'ils référencent la *partie* de l'AIC correspondante.

Avec l'approche AM, les modules seront créés dès la construction du *Protocole d'Application*. Ainsi, les anciens Protocoles d'Application ne seront pas modifiés si un autre Protocole est créé. L'intersection non nulle d'un nouveau *Protocole d'Application* avec les autres sera un ou des modules d'application existants.

5 Analyse de quelques *parties* de STEP

5.1 Unités et mesures

La *partie* 41 de STEP [39] concerne la représentation de la structure du produit. Ce qui nous intéresse ici est le *schéma* measure_schema car il donne la description des quantités physiques.

La principale *entité* pour désigner une unité est appelée named_unit. La Figure 2.11 en donne son diagramme Express-G. Ce diagramme montre qu'une unité est représentée par une composition des puissances de sept dimensions de base : regroupés dans l'*entité* dimensional_exponents. Les sept dimensions de base sont des réels qui désignent :

- la température thermodynamique
- l'intensité lumineuse
- le courant électrique
- la longueur
- la masse
- le temps
- la quantité de matière

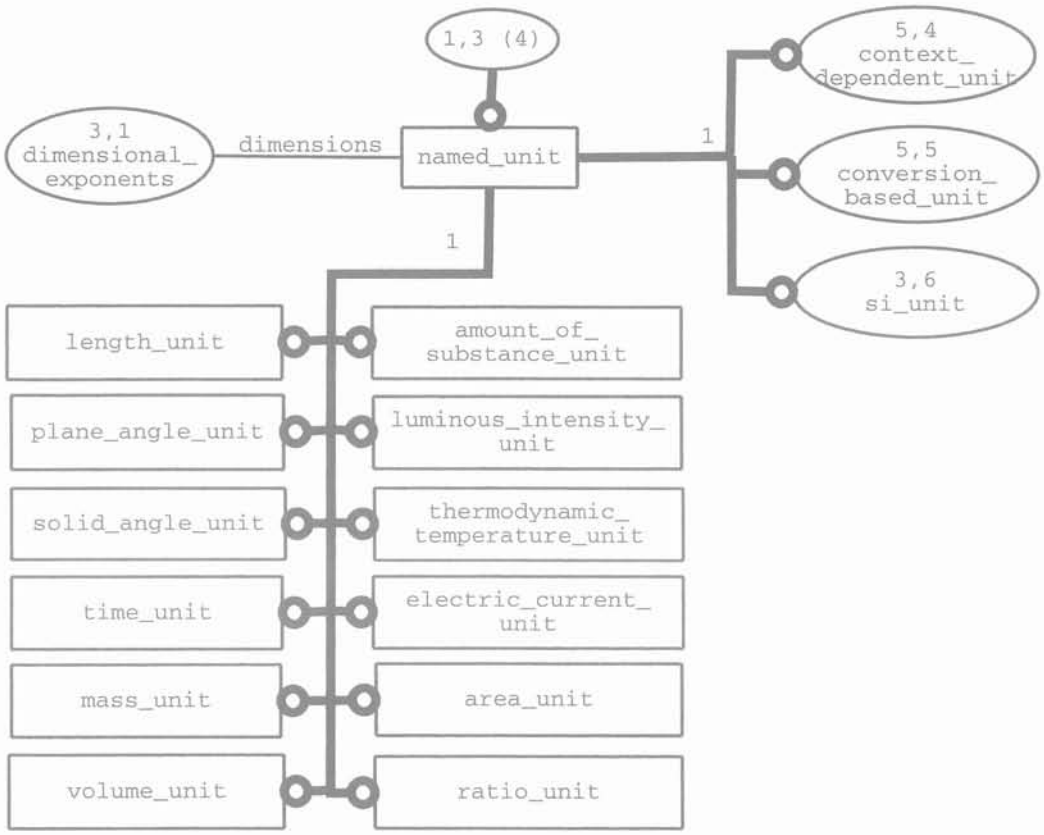


FIG. 2.11 – named_unit [39]

La représentation en Express de l'*entité* dimensional_exponents est :

```
ENTITY dimensional_exponents;
  length_exponent                    : REAL;
  mass_exponent                      : REAL;
  time_exponent                      : REAL;
  electric_current_exponent          : REAL;
  thermodynamic_temperature_exponent : REAL;
  amount_of_substance_exponent       : REAL;
  luminous_intensity_exponent        : REAL;
END_ENTITY;
```

5.2 Matériaux

La *partie* 45 de STEP concerne la description générique d'un matériau. C'est une *partie* très générique : elle ne décrit aucun matériau en particulier.

Elle contient trois *schemas* :

- material_property_definition_schema
- material_property_representation_schema
- qualified_measure_schema

5.2.1 material_property_definition_schema

Ce *schéma* décrit les structures de données nécessaires pour nommer les propriétés d'un materiau, comme par exemple la structure suivante :

```
ENTITY material_designation;
  name        : label;
  definitions : SET [1:?] OF
   characterized_definition;
END_ENTITY;
```

Il donne également les structures nécessaires pour décrire les relations entre un matériau et sa composition, comme par exemple la structure suivante :

```
ENTITY product_material_composition_relationship
SUBTYPE OF (product_definition_relationship);
  class                : label;
  constituent_amount   : SET [1:?] OF
                         measure_with_unit;
  composition_basis    : label;
  determination_method : text;
END_ENTITY;
```

Comme les deux autres *schémas* de la *partie*, il reste très générique et doit pouvoir convenir à n'importe quel matériau. Le diagramme 2.12 en est une représentation en Express-G.

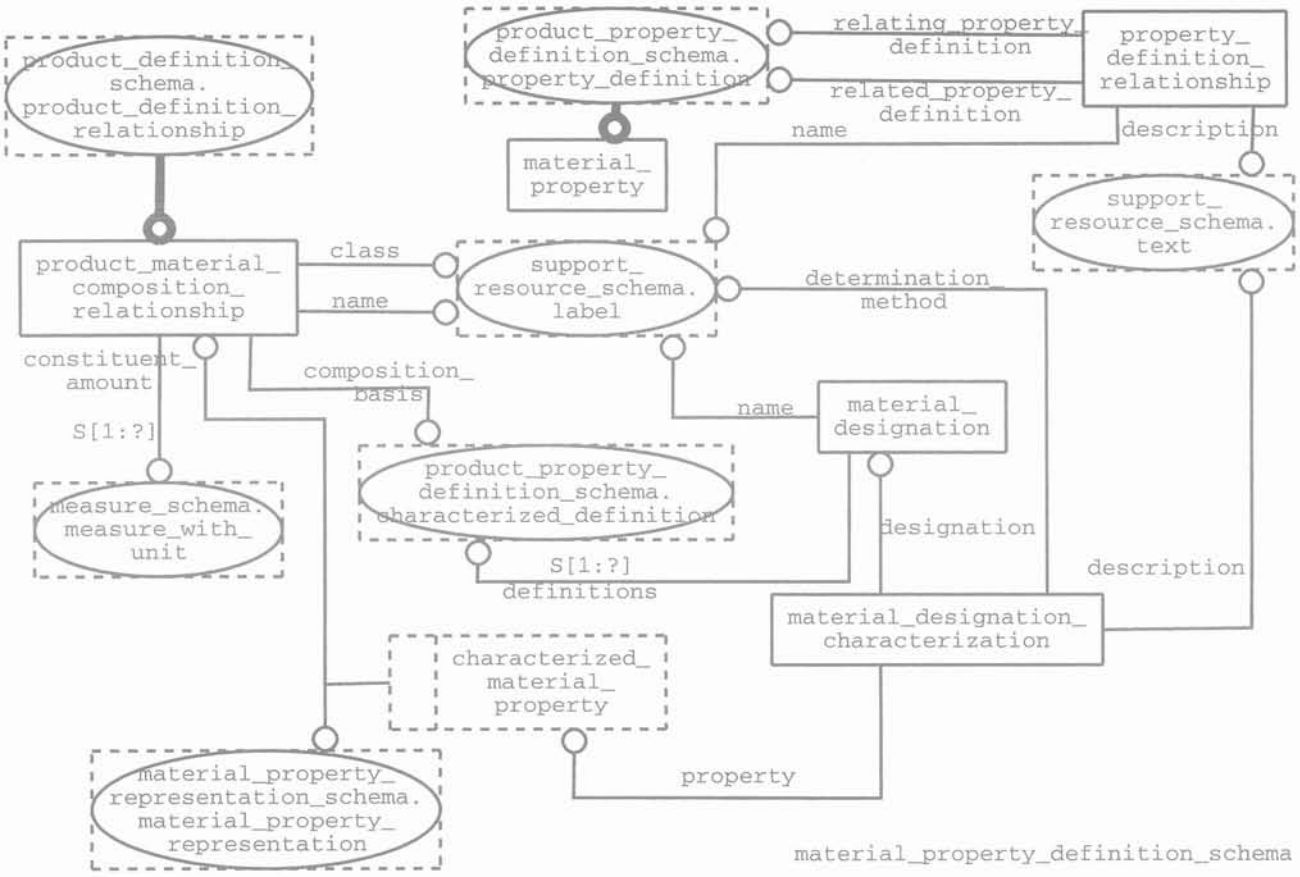


FIG. 2.12 – material_property_definition_schema

5.2.2 material_property_representation_schema

Ce *schéma* contient la structure de base pour toutes les représentations de propriété de matériau dont il faut dériver dans un *Protocole d'Application* : l'*entité* material_property_representation. Il impose également la description des données de l'environnement dans lequel la propriété est définie. Ainsi, l'une des trois structures définies est la suivante :

```
ENTITY material_property_representation
  SUBTYPE OF (property_definition_representation);
  dependent_environment : data_environment;
END_ENTITY;
```

Le diagramme 2.13 montre la représentation Express-G de ce *schéma*.

5.2.3 qualified_measure_schema

Ce *schéma* est une description générique des données utilisées pour instancier des mesures. La Figure 2.14 est le diagramme Express-G complet de ce *schéma*.

Il caractérise par exemple l'incertitude des mesures de la manière suivante :

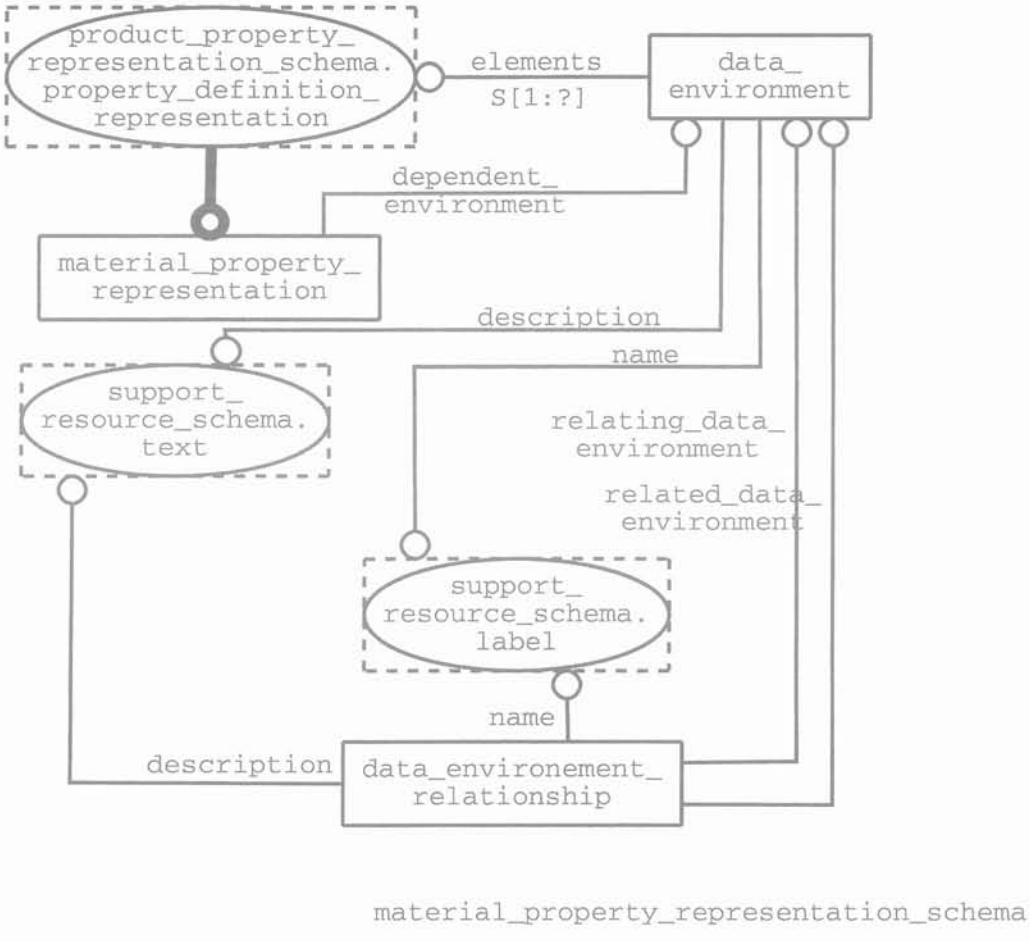


FIG. 2.13 – material_property_representation_schema

```
ENTITY uncertainty_qualifier
  SUPERTYPE OF (ONEOF
  (standard_uncertainty,
   qualitative_uncertainty));
  measure_name : label;
  description  : text;
END_ENTITY;
```

5.2.4 Utilisation dans un *Protocole d'Application*

Cette *partie* très générique sera utilisée dans un *Protocole d'Application* par dérivation et utilisation de contraintes (where rules). Par exemple, dans le *Protocole d'Application* 209 (Composite and Metallic Structural Analysis and Design), la densité surfacique formattée pour la méthode des éléments finis sera définie par l'*entité* fea_area_density qui dérive de l'*entité* fea_material_property_representation qui dérive elle-même de l'*entité* material_property_representati qui elle provient de la *partie* 45, comme représenté ci-dessous en Express :

```
ENTITY fea_material_property_representation
SUBTYPE OF (material_property_representation);
WHERE
wr1: SIZEOF(QUERY
            ( item <* SELF\property_definition_representation.
              used_representation.items |
     (SIZEOF(['FEA_LINEAR_ELASTICITY',
         'FEA_MASS_DENSITY','FEA_AREA_DENSITY',
         'FEA_TANGENTIAL_COEFFICIENT_OF_LINEAR_THERMAL_EXPANSION',
         'FEA_SECANT_COEFFICIENT_OF_LINEAR_THERMAL_EXPANSION',
         'FEA_MOISTURE_ABSORPTION','FEA_SHELL_MEMBRANE_STIFFNESS',
         'FEA_SHELL_BENDING_STIFFNESS',
         'FEA_SHELL_MEMBRANE_BENDING_COUPLING_STIFFNESS',
         'FEA_SHELL_SHEAR_STIFFNESS'] * TYPEOF(item)) = 1) )) = 1;
END_ENTITY; -- fea_material_property_representation

ENTITY fea_area_density
SUBTYPE OF (fea_material_property_representation_item);
   fea_constant : scalar;
END_ENTITY; -- fea_area_density
```

5.3 Géométrie et topologie

La *partie* 42 : Geometric and Topological Representation [40] contient les structures de définition de données géométriques et topologiques. Elle appartient à la *classe* des Ressources d'Information Intégrée, Ressources Génériques. C'est donc une *partie* également très générique. Mais à la différence de la *partie* précédente, elle décrit complètement la plupart des formes nécessaires à la simulation.

Elle contient trois *schémas* :
- geometry_schema
- topology_schema
- geometric_model_schema

5.3.1 geometry_schema

Ce *schéma* donne la définition :
- des points, vecteurs, courbes et surfaces paramétrées
- des volumes finis avec paramétrisation interne
- des transformations
- des points définis par rapport à une courbe ou une surface
- des coniques et surfaces élémentaires
- des courbes définies à partir d'une surface de référence
- des courbes, surfaces, volumes splines
- des points, courbes et surfaces replicas

- des offsets de courbes et surfaces
- des intersections de courbes

Le diagramme Express-G 2.15 montre les différentes représentations possibles d'un point.

5.3.2 topology_schema

Ce *schéma* donne la définition :

- des *entités* fondamentales de la topologie : sommet, arête et face avec chacun un sous-type particulier pour les associer à la géométrie (point, courbe, surface respectivement)
- des collections de ces *entités* de bases pour former les structures topologiques chemin, boucle, coque et des contraintes pour assurer l'intégrité de ces structures
- de l'orientation de ces *entités* topologiques

Le diagramme Express-G 2.16 montre les différentes représentations possibles d'une coque.

5.3.3 geometric_model_schema

- la forme géométrique précise d'objets solides tridimensionnels
- des modèles CSG (Constructive Solid Geometry)
- des modèles CSG dans un espace 2D
- des primitives CSG
- la création de modèles solides par des opérations d'extrusion
- des modèles B-rep (Boundary representation)
- des contraintes pour assurer l'intégrité des modèles
- des modèles de surface
- des modèles en "fil de fer" wireframe
- des ensembles de géométries
- la création d'un modèle solide par duplication dans un nouvel espace

Le diagramme Express-G 2.17 montre la représentation d'un tétraèdre

5.4 Eléments finis

La *partie* 104 : Finite Element Analysis de STEP concerne les éléments finis. Le document complet (ref. [43]), mais non encore définitif, peut-être consulté. Bien que n'ayant pas encore le statut de norme, la *partie* 104 a atteint un stade avancé de développement. Cette section présente sa structure actuelle, divisée en 4 *schémas* :

- structural_response_definition_schema
- structural_response_representation_schema
- finite_element_analysis_control_and_result_schema
- fea_scalar_vector_tensor_schema

5.4.1 Schéma structural_response_definition_schema

Ce *schéma* contient les informations générales du modèle (représentation administrative).

5.4.2 Schéma structural_response_representation_schema

Ce *schéma* donne les définitions détaillées de ce qui forme le modèle FEA :
- les noeuds
- les éléments
- les matériaux
- les propriétés des éléments

Il donne également les hypothèses valables pour la totalité du document (telles que l'unicité d'un modèle FEA, la nécessité d'un système géométrique de coordonnées, ...). Les *types* définis dans ce *schéma* sont utilisés également dans les schémas qui suivent. On remarque que certains *types* tels que les matrices de rigidité, de masse, de frottements fluides, de torsion, de traction, d'efforts de cisaillement sont spécifiques à la mécanique.

Les *entités* définis sont divisés en groupes :
- le modèle FEA : 2D, 3D, lien entre le modèle et sa représentation
- les systèmes de coordonnées : orientation, coordonnées sphériques et cylindriques
- la représentation des noeuds : les différents types de noeuds autorisés
- la représentation des éléments
- les systèmes de coordonnées utilisés dans les éléments : paramètres, coordonnées relatives dans un élément
- l'intégration dans une matrice
- localisation des éléments : paramètres, coordonnées relatives dans un volume
- propriétés des matériaux dans une FEA : définition des matériaux.
- propriétés des éléments : informations sur les éléments 2D et 1D. On considère que les surfaces et les courbes sont des volumes particuliers. Ils nécessitent donc des propriétés supplémentaires, non mentionnées dans la définition de la géométrie
- groupes : ensembles pour, par exemple, affecter une couleur

Les *fonctions* affectées à ce *schéma* permettent de spécifier les données génériques issues de la *classe* Ressources d'Information Intégrées (*sous-classe* Ressources Génériques) pour leur utilisation dans le cadre de la modélisation par les éléments finis. Elles vérifient, par exemple, qu'un volume contient bien le nombre de noeuds nécessaires. Ce *schéma* est essentiel pour représenter un problème modélisé par les éléments finis.

5.4.3 Schéma finite_element_analysis_control_and_result_schema

Ce *schéma* concerne l'utilisation d'un modèle FEA et les résultats donnés par cette utilisation. C'est dans ce *schéma* que sont mentionnés les *types* concernant les conditions aux limites, les valeurs rentrées par l'utilisateur, les vecteurs de variables pour les forces, les moments, les déplacements.

Comme pour le *schéma* précédent, les *entités* peuvent être regroupées en sous-ensembles :
- contrôle d'analyse : description des opérations effectuées sur le modèle FEA pour obtenir la réponse finale. Par exemple, l'*entité* linear_constraint_ equation_element permet le

stockage de la matrice b pour la résolution de l'équation matricielle Ax=b.
- résultats d'analyse : réponse aux opérations décrites par les *entités* précédentes, donnée par une application. Toute application utilisant un modèle FEA doit pouvoir être reliée à ces *entités*.
- état du modèle : état du modèle à un instant donné.
- définition d'un état : si la définition contient des valeurs, elle permet de donner les variables du modèle. Sinon, elle fait appel à une sortie pour que l'application associée calcule ces variables.

Les *fonctions* permettent de vérifier que les informations sont conformes aux *entités* du *schéma*.

5.4.4 Schéma fea_scalar_vector_tensor_schema

Ce *schéma* permet de définir les *entités* de base pour les entrées et sorties d'un modèle FEA : scalaire, vecteur, matrice. Les *types* qui y sont définis donnent la forme de matrices, de vecteurs de toute dimension avec leur propriétés de symétrie. Ils sont pour la plupart utiles dans une analyse mécanique des éléments finis. Par exemple, il est défini une matrice fea_iso_orthotropic_symmetric_ tensor4_3d pour permettre d'instancier la matrice d'élasticité.

5.4.5 Constatations

Il est à remarquer que la *partie* 104 n'est pas encore une norme mais une proposition de norme bien avancée dans le processus de normalisation.

Cette étude de la *partie* 104 que nous avons voulu approfondir a permis de constater tout d'abord que la méthode des éléments finis décrite dans cette *partie* correspond, dans son principe, à celle généralement admise dans toutes les disciplines (mécanique des structures, mécanique des fluides, thermique, électromagnétique) : un modèle est un ensemble d'éléments ou de groupe d'éléments, un élément est un ensemble de noeuds, un noeud peut être lié à un point, les éléments ainsi que les noeuds peuvent être regroupés en une seule *entité*. Cependant de nombreux *entités* et *types* sont spécifiques à la mécanique des structures : cette *partie* n'est donc pas réellement générique malgré son appartenance à la *classe* Ressources d'Information Intégrées. Par exemple, les variables sont limitées à celles utilisées en mécanique des structures (translation, rotation, frottements fluides,...). En particulier, la plupart des objets relatifs à la méthode des éléments finis en électromagnétisme, décrits précédemment, n'apparaissent pas.

6 Conclusion

Ce chapitre a présenté quelques principes de la méthodologie STEP, la construction des différentes *parties*, les langages et les notations utilisés dans le cadre de la norme. STEP ayant été développée avant tout pour permettre des échanges de données, les concepts les plus importants sont la possibilité de faire évoluer les modèles pour intégrer d'autres domaines et la persistance des modèles existants. Ainsi chaque discipline fait référence à une *partie* de STEP mais les structures ne sont pas dupliquées, favorisant ainsi les échanges et l'interopérabilité des

systèmes informatiques.

La norme STEP possède de très nombreux avantages. Elle fournit une vue intégrée des données de produits et utilise la plupart des facilités des méthodes orientées objets. C'est en plus le seul standard international pour l'échange et la représentation des modèles de données produits. Elle permet également de séparer les méthodes d'implantation du contenu de l'information et ainsi de définir plusieurs vues spécifiques d'application. Elle permet enfin pour les entreprises de situer une information dans le cycle de vie d'un produit, à savoir comment et quand cette information est utilisée et ainsi définir les besoins d'une application particulière.

Par contre, il est à constater que STEP est très vaste donc pose des problèmes d'interopérabilité entre les modèles et donc leurs applications.

Nous avons ainsi fait l'analyse des *parties* de STEP qui vont servir dans le domaines de la simulation en électromagnétisme. La *partie* concernant la géométrie est très complète. La *partie* concernant les matériaux est beaucoup trop générique et devra donner lieu à de nombreuses autres structures pour pouvoir décrire les propriétés électriques et magnétiques des objets analysés. La *partie* concernant les éléments finis est assez complète également mais reste générique pour la méthode des éléments finis : il faut d'autres structures pour l'adapter à la vue électromagnétique.

Ce sont ces principes, langages, notations et structures génériques que nous utiliserons pour décrire le modèle et mode d'échange proposés dans le chapitre suivant.

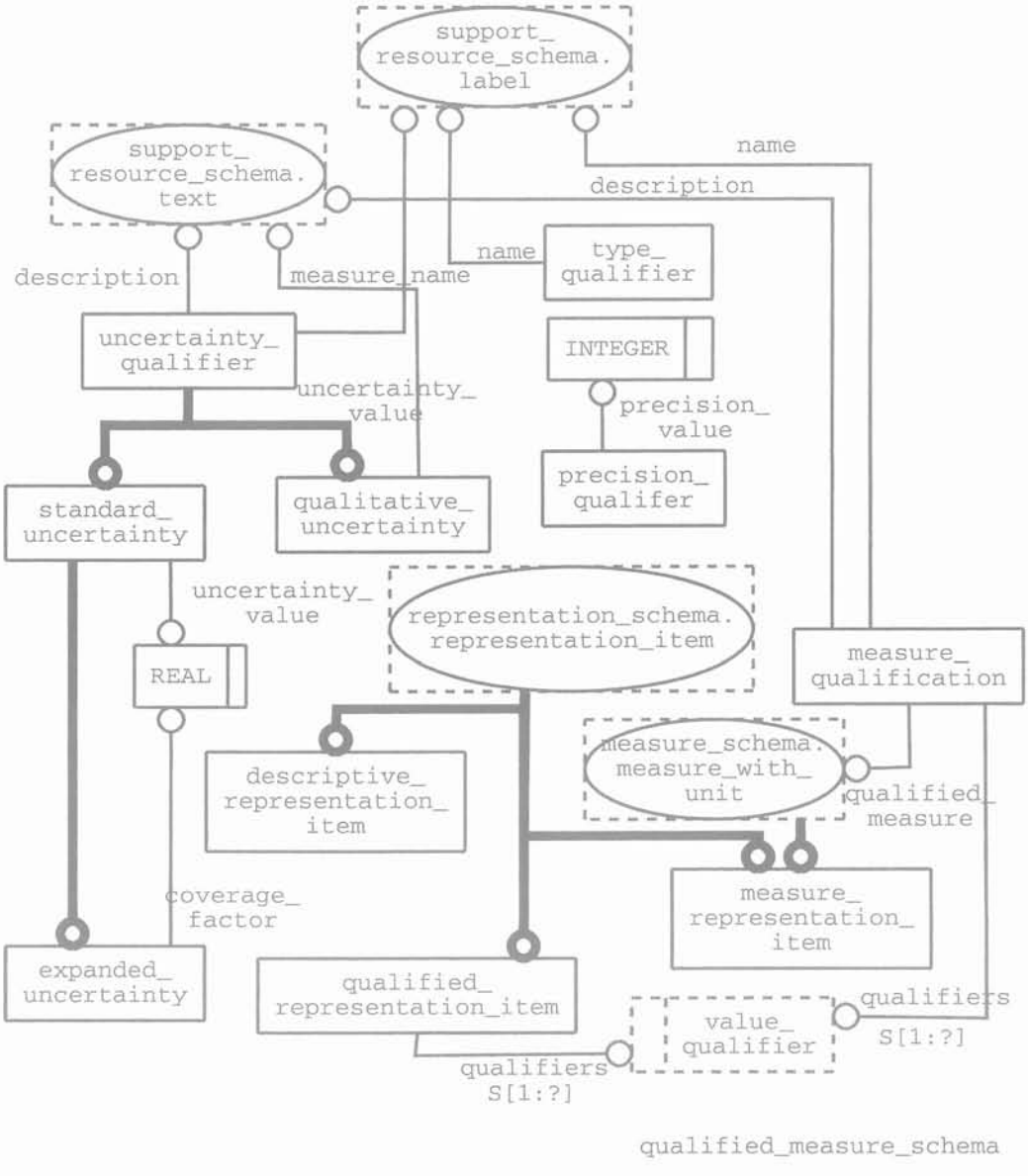


FIG. 2.14 – qualified_measure_schema

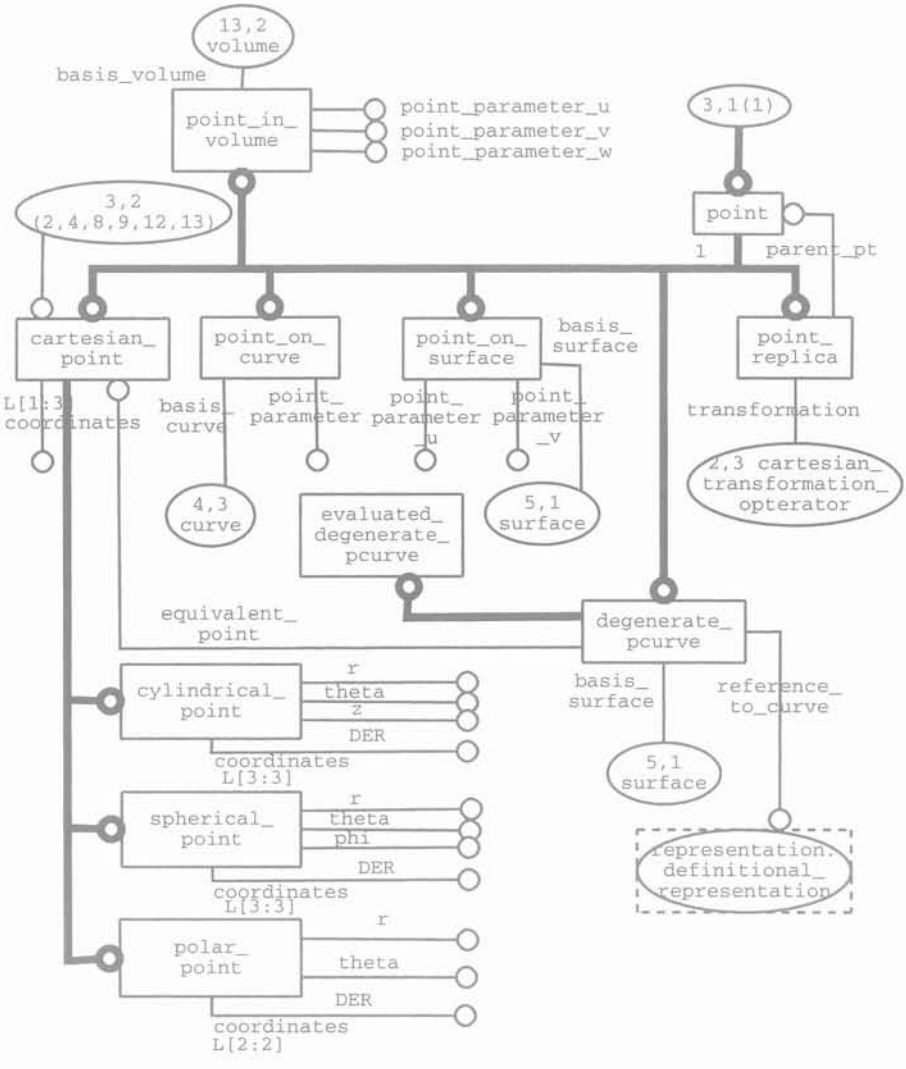


FIG. 2.15 – Représentations d'un point dans STEP [40]

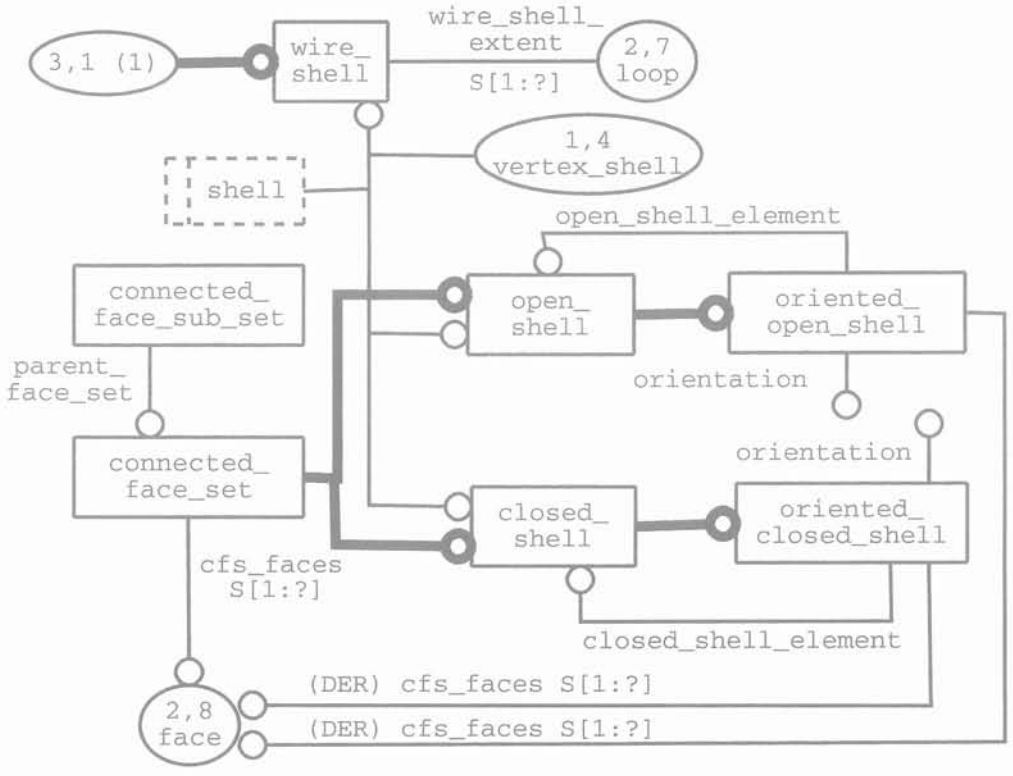


FIG. 2.16 – Représentations d'une coque dans STEP [40]

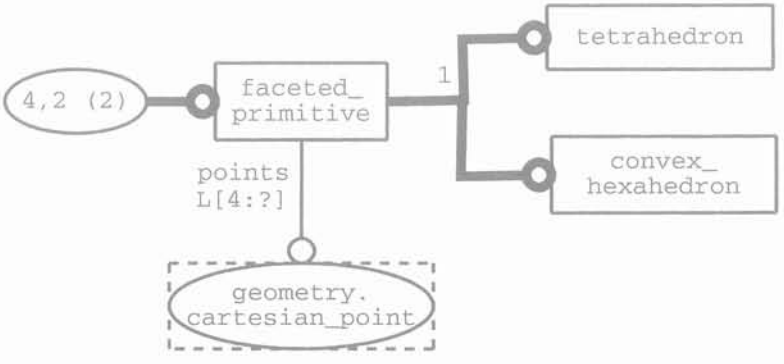


FIG. 2.17 – Représentation d'un tétraèdre dans STEP [40]

Chapitre III

Un *Protocole d'Application* pour la Magnétostatique

Le chapitre précédent a montré la structure et la méthodologie STEP. Cette méthodologie est utilisée pour échanger des données et est actuellement en train de remplacer toutes les normes existantes d'échange de données. Ce qui est remarquable en particulier est qu'elle dispose déjà de structures génériques pour la géométrie, la topologie, la désignation des matériaux, les mesures, les unités et les éléments finis.

Ce chapitre décrit le modèle de données que nous proposons pour décrire la simulation en Magnétostatique. Nous insisterons sur l'utilisation de la méthodologie STEP et des structures de données déjà définies dans la norme, telles que la géométrie. Nous voulons montrer qu'on peut structurer de manière précise les données nécessaires à l'analyse en Magnétostatique dans le but d'utiliser un maximum des connaissances provenant d'autres domaines.

La première section est une introduction à l'analyse en Electromagnétisme en général, son but et de ses contraintes. C'est un rappel très succint des lois physiques dans ce domaine. La deuxième section et la troisième section décrivent pourquoi et comment on simule ces lois dans un système informatique, d'abord en Electromagnétisme puis en Magnétostatique. La quatrième section est la définition des *unités de fonctionalités* (voir Chapitre II, section 4.1.4) que nous avons identifiés. C'est le découpage macroscopique par fonction des structures de données décrites. La cinquième section décrit le modèle ARM. C'est une description détaillée des *entités* EXPRESS proposés ainsi que leurs relations d'héritage ou d'attribut tel qu'il est conçu dans le domaine de la simulation en Magnétostatique. Ces *entités* ont été rassemblées dans un schéma que nous avons nommé STEPMSA (STEP MagnetoStatic Analysis). La sixième section présente, dans le processus de développement d'un *Protocole d'Application* les perspectives envisagées pour ce modèle ARM. La dernière section est une conclusion sur le modèle réalisé.

1 L'analyse en l'Electromagnétisme

D'après [9],

> *L'objet fondamental de l'électromagnétisme est de décrire les interactions qui s'exercent à l'intérieur d'un système de particules chargées.*

L'électromagnétisme définit ainsi un être physique (E,B) appelé *champ électromagnétique.* Le but de l'analyse en électromagnétisme est de calculer ce champ à partir de la distribution de charges et de courants qui le crée, distribution caractérisée par son repère R, sa densité de charge et sa densité de courants. Ce champ peut-être calculé à partir de sa source grâce à quatre relations locales appelées équations de Maxwell.

Les quatre relations sont :

$$\begin{aligned} \mathbf{rot}\,\vec{E} &= -\frac{\partial \vec{B}}{\partial t} \\ \mathbf{rot}\,\vec{H} &= \vec{j} + \frac{\partial \vec{D}}{\partial t} \\ div\,\vec{B} &= 0 \\ div\,\vec{D} &= \rho \end{aligned}$$

Avec :

$$\begin{aligned} \vec{B} &= \mu\vec{H} + \vec{B_r} \\ \vec{D} &= \varepsilon\vec{E} \\ \vec{j} &= \sigma\vec{E} \end{aligned}$$

$\vec{E}$: champ électrique
$\vec{D}$: induction électrique
$\vec{H}$: densité de courant
$\vec{B}$: induction magnétique
$\vec{j}$: densité de courant
ρ : densité de charge
$\vec{B_r}$: induction rémanente
μ : perméabilité magnétique
ε : permittivité électrique
σ : conductibilité électrique

2 La simulation en Electromagnétisme

Dans les cas concrets, la géométrie, les matériaux, les sources de champ existants, les contraintes imposés au domaine de l'espace dont on veut faire l'étude électromagnétique rendent très complexe, voire impossible la résolution analytique des équations présentés précédemment.

Par contre, si au lieu de chercher une expression exacte du champ électromagnétique, on en cherche une approximation, en tout point du domaine étudié, le problème peut être résolu de

manière numérique.

Les méthodes numériques pour déterminer le champ électromagnétique dans un domaine de l'espace sont pour la plupart bien au point. L'une des méthodes les plus anciennes et les plus connues est la méthode des éléments finis. Cette méthode se base sur une division de l'espace en sous-domaines appelés éléments finis. La précision du champ calculé dépendra par exemple de la qualité et de la quantité de sous-domaines qu'on traite. Cette division de l'espace permet ainsi d'effectuer des calculs discrets et l'utilisation d'outils informatiques. Depuis sa création, de très nombreux travaux [6], [7], [25] ont permis de mettre au point les techniques informatiques de cette méthode. D'autres méthodes plus récentes font également l'objet d'études poussées. C'est le cas de la méthode des intégrales de frontière [15], la méthode des éléments diffus [13], [22]...

Le but de la simulation en électromagnétisme est ainsi de déterminer le champ électromagnétique dans un domaine donné de l'espace par des méthodes numériques.

3 La simulation en Magnétostatique

La Magnétostatique est un sous-domaine de l'Electromagnétisme. D'après [9],

> *On appelle magnétostatique la branche de l'électromagnétisme qui étudie les champs magnétiques créés par des courants permanents.*

Le problème le plus général qui sera étudié sera donc de déterminer le champ magnétostatique B dont la source est une distribution de courants permanents.

Ce problème peut, sous certaines hypothèses, être étudié analytiquement. Dans notre cas, l'étude se fera de manière numérique. On ne tiendra donc pas compte des méthodes analytiques pures.

Le domaine de la simulation en Magnétostatique comprend ainsi à la fois les lois de la magnétostatique mais également les méthodes numériques qui permettent de mettre en oeuvre ces lois. Il comprend donc :

- la description géométrique 2D et 3D d'objets
- la description des matériaux qui composent ces objets
- le maillage
- la résolution numérique des équations de la magnétostatique

4 UoF : Units of Functionality

Chaque *entité* d'un protocole d'application appartient à une *unité de fonctionalité* qui définit son rôle dans le modèle. Dans notre cas, nous avons identifié six Uof décrits dans les sous-sections suivantes.

4.1 administrative_data

Permet de nommer et de décrire les données du produit dont on fait l'analyse électromagnétique, les opérations pour arriver aux résultats.

4.2 material_data

Permet de décrire les données de matériau nécessaires pour la résolution d'un problème en magnétostatique.

4.3 geometry_data

Permet de donner les informations de type géométriques.

4.4 topology_data

Permet de donner les informations de type topologiques.

4.5 sources_data

Permet de décrire les sources de champs magnétiques.

4.6 formulation_data

Permet de décrire les formulations servant à résoudre les équations de la magnétostatique.

5 Le schema STEPMSA : ARM

L'ARM est l'expression du modèle tel que le conçoivent les spécialistes du domaine étudié. C'est ce niveau de modèle que nous souhaitons spécifier pour le domaine de la magnétostatique dans le schéma que nous avons nommé STEPMSA (Step MagnetoStatic Analysis). Nous ne décrirons pas ici les données concernant la configuration, la géométrie, la topologie, le maillage qui sont les mêmes que celles décrites respectivement dans les spécifications [39], [40], [43]. Le chapitre précédent a également donné un résumé et une analyse de ces *parties*. Nous détaillerons, dans les sous-sections suivantes, les données concernant la magnétostatique, la description physique du problème adaptée à la simulation : matériaux, variables, contraintes, sources. Nous avons choisi de ne donner ici que la représentation graphique du modèle avec éventuellement un exemple en EXPRESS. La représentation EXPRESS complète est donnée en Annexe I.

5.1 Réels et complexes

Les modèles de matériaux et de variables électromagnétiques peuvent être représentés sous forme réelle ou complexe.

L'*entité* math_scalar (voir la Figure 3.1) désigne à la fois un réel et un complexe.

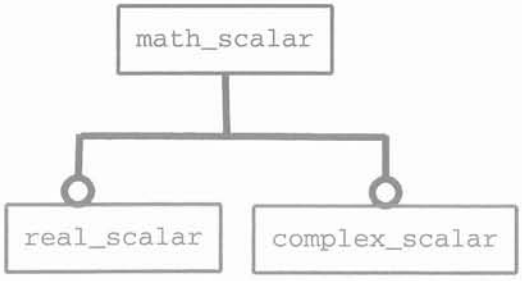


FIG. 3.1 – Réels et Complexes

5.2 Modèles de propriété de matériaux

Un modèle de propriété permet de définir la relation entre deux ou plusieurs variables. La relation peut être analytique, tabulée ou définie par sous-programme. Définir un modèle de propriétés permet de traiter les matériaux numériquement car il est insérable directement dans le code. Les modèles décrits ci-dessous sont ceux utilisés dans les logiciels Flux2D et Flux3D ([44]). Nous avons structuré ces données de telle sorte qu'elles soient compatibles avec le reste de l'ARM mais en respectant cependant la sémantique des concepts tels que les modèles splines, linéaires ou gaussiens.

Le schéma Express-G de la Figure 3.3 donne la description des différents modèles de propriétés connus. Il faut remarquer que nous donnons ici tous les modèles de propriétés utilisés dans un logiciel comme Flux3D, indépendamment des hypotèses magnétiques, électriques ou thermiques : certaines propriétés de matériau ont un sens uniquement en magnétisme (permeabilité), d'autres uniquement en élecricité (résistivité) ou en thermique (coefficient de convection).

L'*entité* la plus abstraite est l'*entité* fea_material_property_model qui sera décrite de la manière suivante :

```
ENTITY fea_material_property_model
SUPERTYPE OF (ONEOF (constant,
                     line,
                     exponential,
                     simple_analytical_saturation,
                     adjustable_analytical_saturation,
                     spline_saturation,
                     parabola_plus_line,
                     constant_plus_gaussian,
                     exponential_plus_gaussian,
                     constant_mul_exponential,
                     simple_analytical_saturation_mul_exponential,
                     adjustable_analytical_saturation_mul_exponential,
                     node_values,
                     user_defined));

END_ENTITY;
```

Les *entités* dérivées sont structurées de manière à pouvoir contenir toutes les données nécessaires à la description du modèle. Par exemple, l'*entité* exponential_model représente un modèle de type exponentiel tel que montré dans la Figure 3.2. La propriété peut être décrite par une courbe exponentielle dont la valeur en zero est "zero_value", la constante de temps "tau" et la valeur de la limite à l'infini "constant".

Elle sera représentée en EXPRESS de la manière suivante :

```
ENTITY exponential_model
SUBTYPE OF (fea_material_property_model);
   zero_value : math_scalar;
   tau : math_scalar;
   constant : math_scalar;
END_ENTITY;
```

Ce modèle est utilisé par exemple pour décrire une variation exponentielle du potentiel V en fonction de la température :

$$V = V_0 * exp^{-T/tau} + const$$

Un exemple d'instantiation de cette propriété dans un fichier neutre STEP peut être :

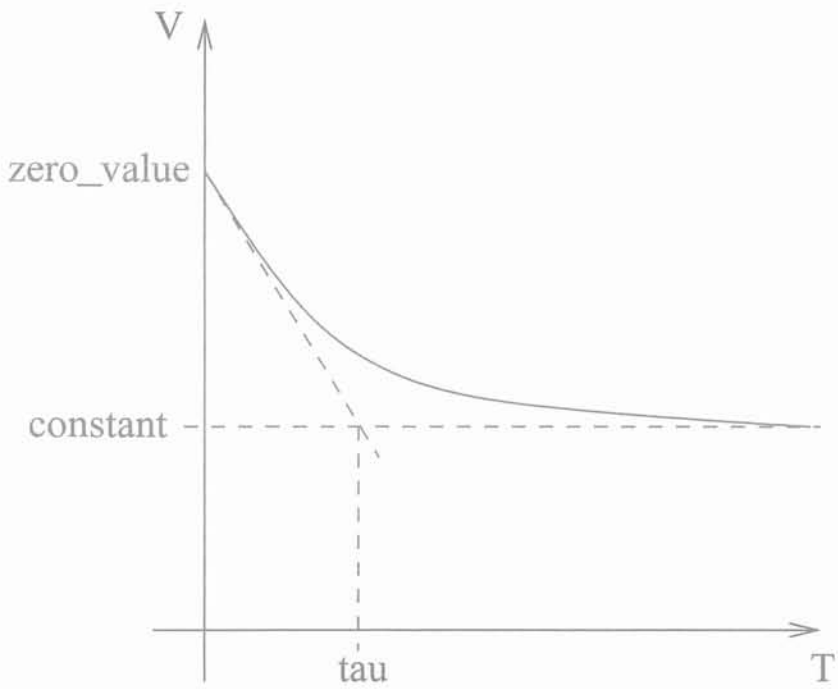


FIG. 3.2 – Modèle de propriétés : exponentielle

```
#19 = real_scalar(1.0);
#20 = real_scalar(1.0);
#21 = real_scalar(1.0);
#22 = exponential_model('V', #19, #20, #21);
```

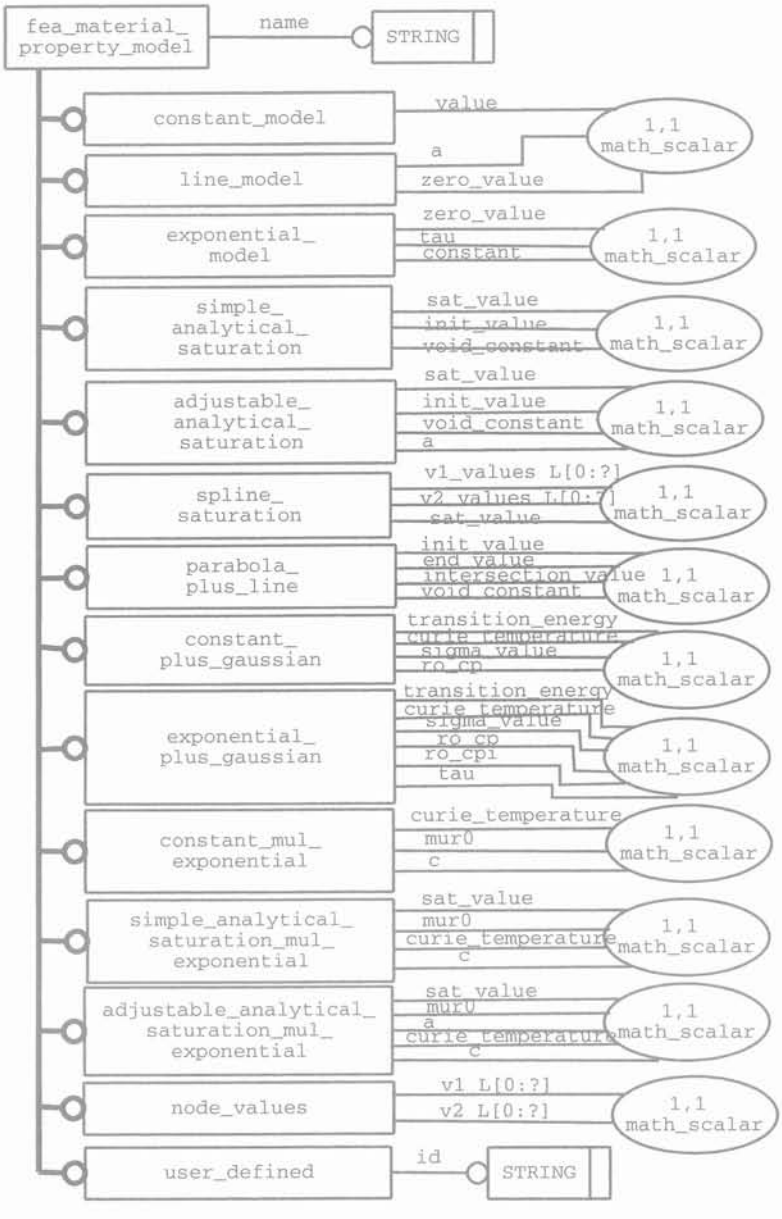


FIG. 3.3 – Modèle de propriétés

5.3 Propriétés de matériaux

Les propriétés d'un matériau en électromagnétisme sont de deux types : les propriétés électriques et les propriétés magnétiques. Pour prévoir l'inclusion des propriétés électriques et éventuellement thermiques, on ajoute une *entité* supplémentaire dérivée de la principale *entité* de description des propriétés, description adaptée pour la simulation par éléments finis.

Une propriété de matériau est toujours définie avec son modèle de propriétés. Dans notre *schema*, ce lien est représenté par l'attribut *model*.

La Figure 3.4 montre les liens de dérivation et d'attributs entre notre modèle de données et les *entités* qui existent déjà dans STEP en ce qui concerne l'affectation des matériaux à chaque élément fini. Dans cette Figure, pour simplifier la représentation, nous n'avons pas détaillé toutes les *entités* de la partie 104 (éléments finis), partie 45 (matériaux) et partie 41 (représentation). On pourra se référer à [43], [41] et [39] pour le modèle complet.

Le lien avec l'ensemble des autres propriétés (mécaniques, thermiques,...) du matériau est fait par dérivation de l'*entité* fea_material_property_representation_item. En effet, suivant le modèle élément finis (défini dans la *partie* 104) les propriétés sont affectées à chaque élément par l'*entité* element_material qui contient un *attribut* "properties" qui est un ensemble de "material_property_representation"s qui sont eux-mêmes dérivés de l'*entité* "property_definition_representation" donc font références à une *entité* "representation" qui contient un *attribut* "items" qui est un ensemble de "representation_item" dont dérive l'*entité* "fea_material_property_representation_item" dont nous faisons dériver notre *entité* "fea_material_magnetic_property_representation_item". La Figure 3.4 résume de manière très simplifiée les relations que nous venons de décrire.

Ainsi, la principale classe pour les propriétés magnétiques seront décrites dans notre modèle de données de la manière suivante :

```
ENTITY fea_material_magnetic_property_representation_item
SUBTYPE OF (fea_material_property_representation_item)
SUPERTYPE OF (ONEOF (permeability,
                     real_coercitif_magnetic_field,
                     complex_current_density,
                     real_current_density,
                     permeability,
                     reluctivity
                     ));
   model: fea_material_property_model;
END_ENTITY;
```

Par exemple, un matériau avec une perméabilité relative de 1000 et donc une perméabilité absolue de $1000 * 4 * PI * 10^{-7} H.m^{-1}$ sera décrit avec la séquence suivante dans un fichier neutre

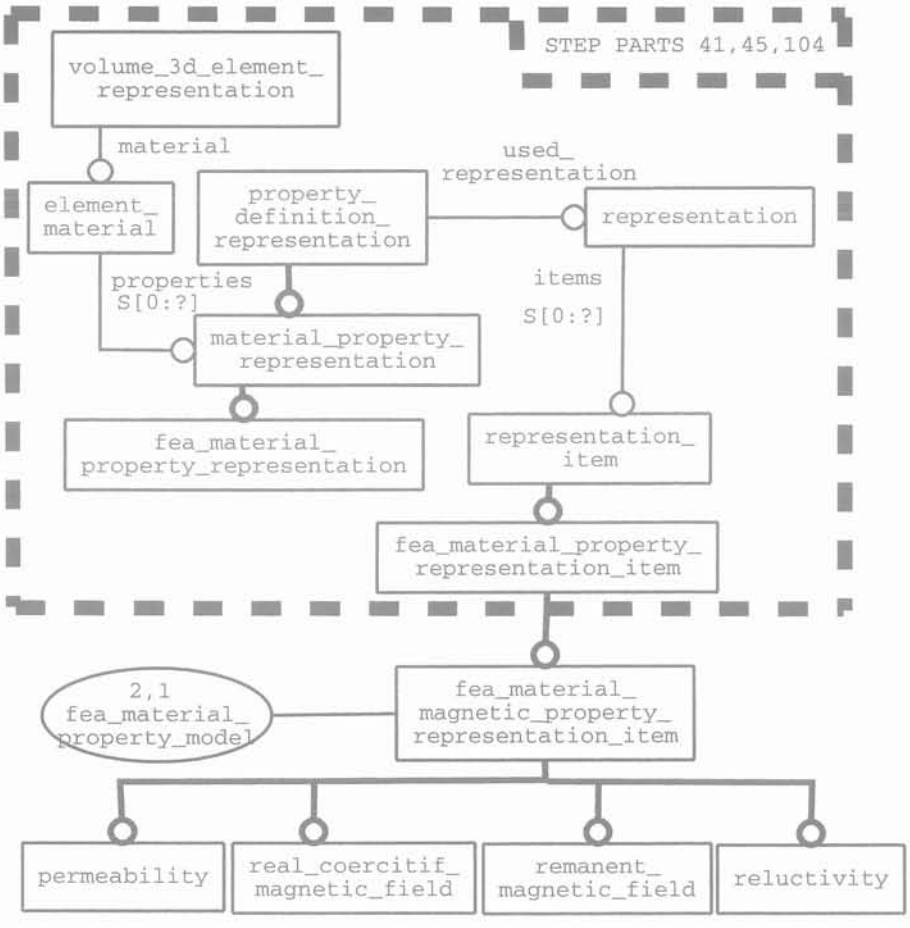


FIG. 3.4 – Propriétés de matériaux

STEP :

```
#1749 = REAL_SCALAR(0.001256);
#1750 = CONSTANT_MODEL('muconst',#1749);
#1751 = PERMEABILITY(#1750);
#1752 = REPRESENTATION($, (#1751), $);
#1753 = FEA_MATERIAL_PROPERTY_REPRESENTATION($, #1752, $);
#1754 = ELEMENT_MATERIAL('iron', 'iron with constant permeability', (#1753));
```

5.4 Variables et opérateurs

En électromagnétisme, les opérateurs de dérivation de vecteurs ou de scalaires rotationel, gradient, divergence, laplacien scalaire et laplacien vectoriel sont souvent utilisés. Pour pouvoir exprimer les relations entre certaines variables, il est nécessaire de leur donner une structure. La Figure 3.5 donne la représentation Express-G de ces structures.

Les principales variables utilisés lors de la résolution d'un problème en magnétostatique sont de deux types :

- les variables vectorielles. La plus utilisée en simulation magnétostatique par éléments finis est encore maintenant le potentiel vecteur magnétique $\overrightarrow{A}$.
- les variables scalaires. La plus utilisée en simulation magnétostatique par éléments finis est le potentiel scalaire magnétique ϕ.

Pour chacun de ces deux types de variables, en plus des variables les plus utilisées actuellement, nous avons ajouté les *entités* derived_vector_variables et derived_scalar_variable qui représentent respectivement les variables vectorielles dérivées et les variables scalaires dérivées (voir Figure 3.5).

Les structures de données pour les variables sont résumées dans Figure 3.5.

5.5 Formulations

Une formulation est un ensemble d'équations déduites des équations de Maxwell et équivalentes à celles ci soit de manière générale soit avec des conditions particulières. Son intérêt est tout d'abord de réduire au minimum le nombre de variables manipulées et donc la taille des calculs à effectuer. Un autre intérêt des formulations est également de stabiliser numériquement le calcul effectué.

Toutes les formulations nécessitent la même quantité d'information, à savoir :
- un nom pour l'idenfication
- une description
- une variable qui sera la variable d'état de la formulation
- un domaine de validité qui est un espace mathématique
- une liste des types de contraintes autorisées
- une liste des variables auxquelles la formulation a accès
- une liste des propriétés de matériau dont elle a besoin

La représentation Express de l'*entité* sera ainsi la suivante :

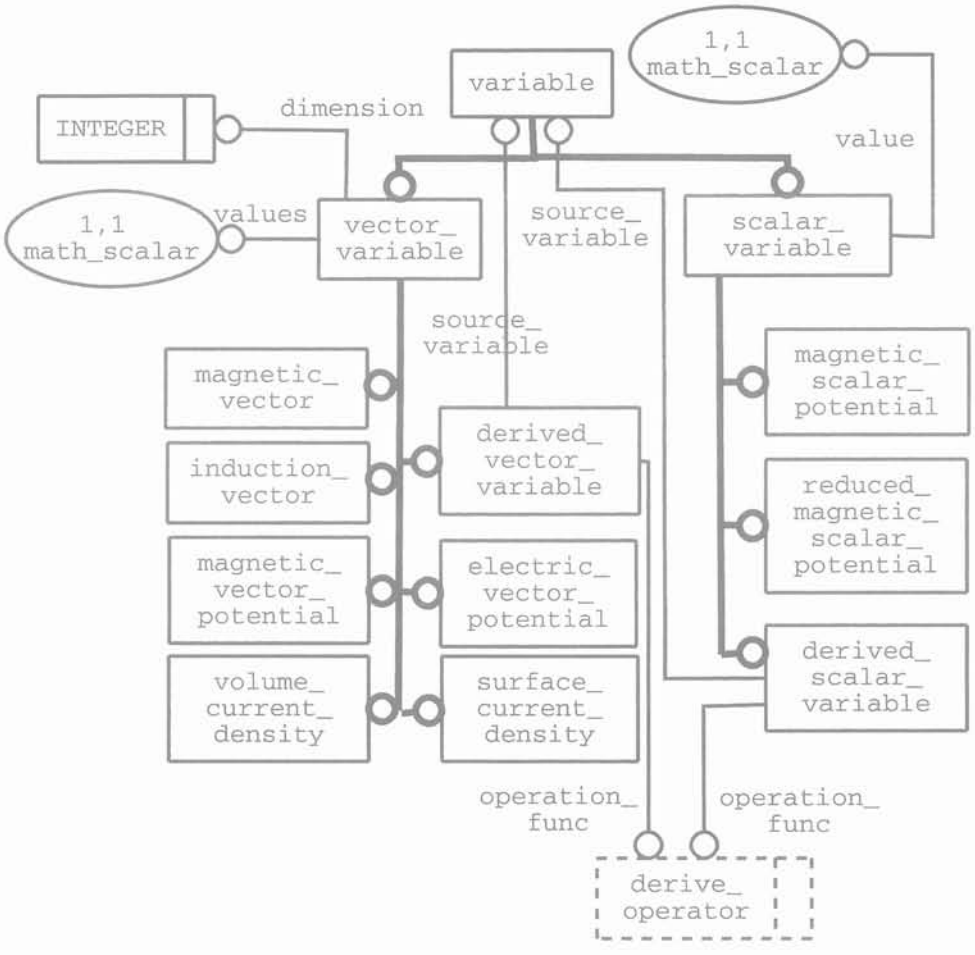


FIG. 3.5 – Variables

```
ENTITY formulation
SUPERTYPE OF (ONEOF(magnetostatic_formulation))
   name : label;
   description : text;
   state_variable : variable;
   validity_domain : maths_space;
   allowed_constraints : SET [0:?] OF constraint;
   accessible_variables : SET [0:?] OF variable;
   needed_material_properties : SET [0:?] OF material_property;
END_ENTITY;
```

Une des formulations les plus connues en magnétostatique est la formulation "potentiel vecteur magnétique" ([16]) puisqu'elle ne fait intervenir qu'une variable : le potentiel vecteur magnétique $\overrightarrow{A}$:

$$rot(\nu rot\,\overrightarrow{A} = \overrightarrow{J})$$

Sa représentation Express contiendra une règle pour imposer une variable vecteur potentiel

magnétique ou une dérivée de cette variable, comme représentée ci-dessous :

```
ENTITY magnetostatic_3d_vector_potential
SUBTYPE OF (magnetostatic_formulation);
WHERE
  WR1: 'SELF.state_variable'
       IN TYPEOF (MAGNETIC_VECTOR_POTENTIAL));
END_ENTITY;
```

5.6 Régions

Une région physique est une partie du domaine géométrique étudié dont tous les éléments ont les mêmes propriétés physiques et à qui on affecte les mêmes formulations. La Figure 3.6 en donne la représentation Express-G.

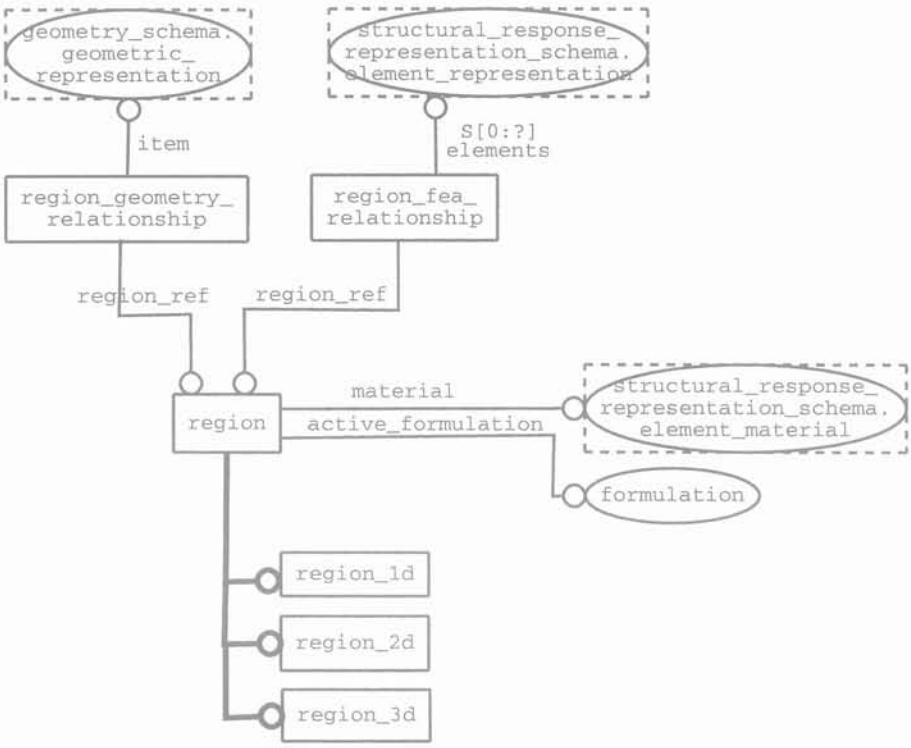


FIG. 3.6 – Regions

5.7 Contraintes

Une contrainte est une relation imposée à une variable d'état pour tenir compte des effets d'interaction du domaine de l'espace étudié avec le reste de l'espace.

La relation entre la contrainte et l'*entité* géométrique sur laquelle elle s'applique est donnée par l'*entité* constraint_geometry_relationship.

La relation entre la contrainte et les *entités* éléments finis sur lesquelles elle s'applique est donnée par l'*entité* constraint_fea_relationship.

Parmi les contraintes, peut être citée la symétrie qui est une contrainte à la fois géométrique et physique : la géométrie et la physique (sources de champ par exemple) sont les mêmes de chaque côté de l'*entité* géométrique sur laquelle est appliquée la contrainte.

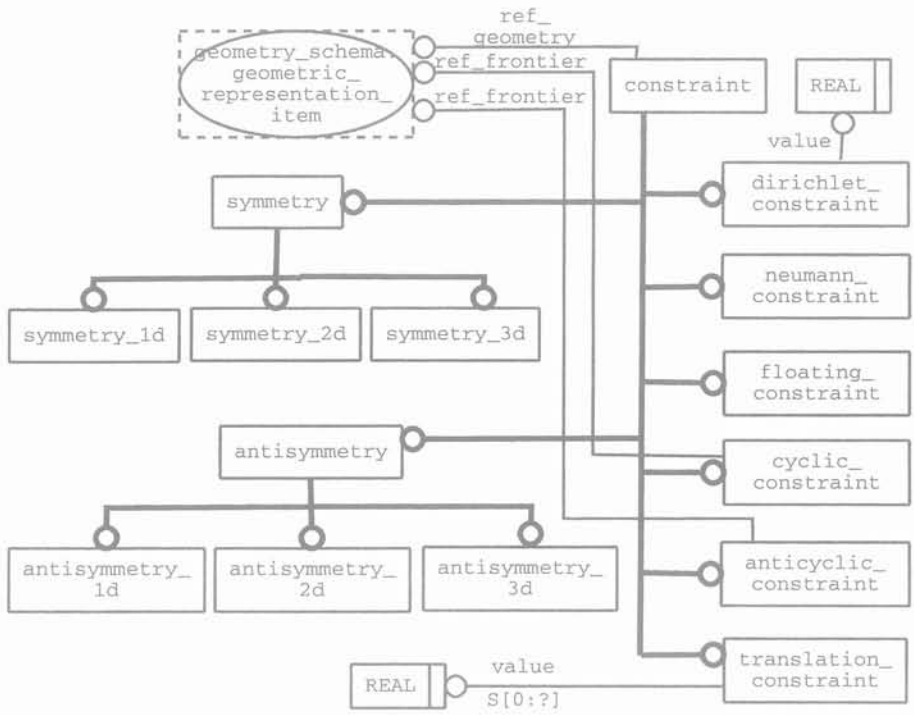


FIG. 3.7 – Contraintes

5.8 Sources de champs

Une source de champ produit de manière constante un champ magnétique, électrique ou électromagnétique. Elle influence tout ou une partie du domaine étudié mais n'est jamais perturbée par celui-ci.

On distingue ainsi les sources de champ magnétiques (courants, aimants,...), les sources de champ électriques (charges), les sources de chaleur, ...

Dans le cadre de notre modèle électromagnétique, nous ne détaillerons que les sources de champ magnétiques mais nous laissons la possibilité d'ajouter des structures pour les autres sources en ajoutant entre l'*entité* field_source et l'énumération des sources de champ magné-

tiques, l'*entité* magnetic_field_source. Ainsi pour avoir un modèle de données plus général, on pourra ajouter les *entités* electric_field_source et thermic_field_source.

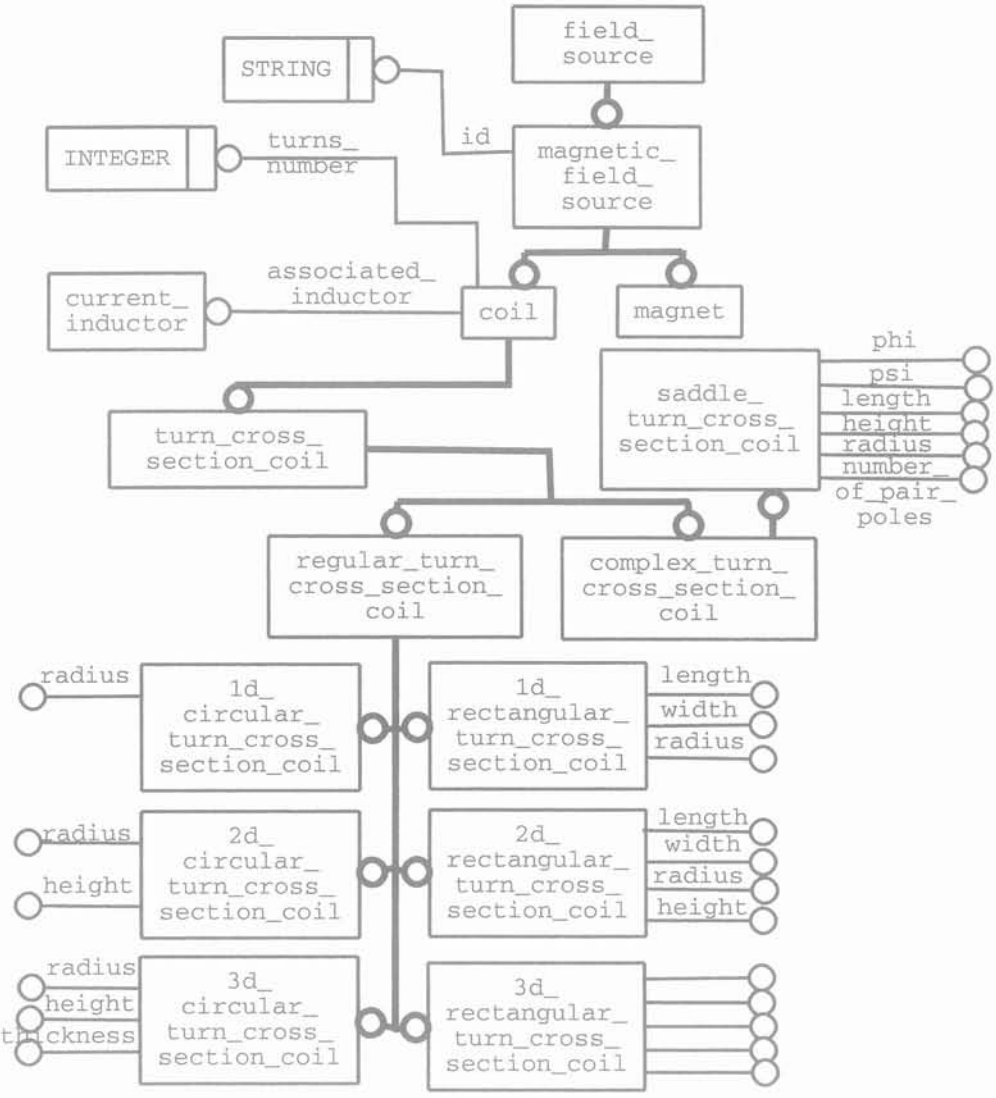


FIG. 3.8 – Sources de champs magnétiques

La Figure 3.8 donne la représentation Express-G des structures décrivant les sources de champs magnétiques.

6 Le schéma STEPMSA : AIM

Le modèle AIM (Application Interpreted Model) est le résultat de la correspondance de l'ARM (Application Reference Model) avec les *entités* des *ressources intégrées* de STEP.

L'ARM est le modèle proposé par les spécialistes du domaine étudié. L'AIM est le modèle proposé par les spécialistes de la modélisation STEP. Dans ce travail, nous nous positionnons du coté des spécialistes de la simulation en électromagnétisme et donc ne proposons qu'un modèle adapté au langage de ce domaine. Comme éventuelle suite de ce travail, il pourra être envisagé de faire l'intégration de ce modèle ARM pour obtenir le modèle AIM. La principale difficulté est qu'il faut connaître de manière très approfondie les structures existantes des *ressources d'informataion intégrées* de STEP et des méthodes de modélisation STEP. Ce travail d'intégration ne peut donc se faire qu'en collaboration avec des spécialistes de la modélisation STEP.

7 Conclusion

Ce chapitre a présenté le modèle de données que nous proposons pour définir la simulation magnétostatique. Ce modèle doit permettre de décrire complètement un problème en magnétostatique utilisé pour les dispositifs électrotechniques. Dans un premier temps, nous avons décrit les modèles de matériau, la manière de représenter les propriétés physiques (principalement magnétiques) des matériaux utilisés en électrotechnique. Puis nous avons inclus la géométrie et la topologie proposée dans les parties génériques de STEP, c'est à dire les parties 41, 42, 43, 44, 45 et 104. Ces parties génériques permettent effectivement de décrire toutes les parties non physiques du problèmes. Notre apport a donc concerné principalement la description physique : matériaux, conditions aux limites, sources, contraintes, formulations. La description en Express de notre contribution se trouve en Annexe I.

Chapitre IV

Implanter STEP : un Environnement de Développement

Le chapitre précédent a présenté le modèle de données que nous avons proposé pour la simulation en magnétostatique. Mais comme tous les protocoles d'application STEP, ce modèle est complexe et contient de grandes quantités de structures (plus de 900 *entités* et plus de 200 *types*) qu'il faut pouvoir exploiter informatiquement.

Dans ce chapitre nous tenterons de mettre au point des méthodes d'implantation d'un *Protocole d'Application* STEP dans un logiciel. Après une étude des différentes méthodes proposées par la norme et des méthodes couramment utilisées dans le domaine de la compilation de langages, nous décrirons les applications que nous avons utilisées et celles que nous avons développées.

La première section décrit les *parties* de STEP qui spécifient l'implantation logicielle de la norme. Ces spécifications proposant différentes méthodes d'implantation, nous avons choisi de mettre en oeuvre celle qui permet une bonne réutilisabilité et une mise à jour simple ainsi qu'un langage d'application : Java. Après une étude des outils existants, présentée dans la section deux, nous avons constaté que l'implantation en Java était alors trop peu étudiée. C'est pourquoi nous avons mené une étude pratique des différents éléments logiciels nécessaires pour développer un *Protocole d'Application* et pour ensuite la mise en oeuvre de ce Protocole. Ce qui nous a conduits à nous intéresser aux méthodes de compilation/génération de code. Ce sont les résultats de nos réflexions sur ces méthodes et leur mise en oeuvre logicielle que nous décrirons dans la troisième section. La section quatre décrit les outils que nous avons utilisés et ceux que nous avons réalisés à la fois pour développer et pour mettre en oeuvre un *Protocole d'Application* STEP. La dernière section est une conclusion sur la maquette que nous avons obtenue.

1 Spécification pour l'implantation logicielle de STEP

Une des spécificités de la méthodologie de modélisation de données STEP est de fournir des méthodes de mise en oeuvre logicielle des différents protocoles d'application. Ces méthodes sont décrites dans la classe "Implementation Methods" qui comprend les *parties* 21 à 29 (voir l'Annexe B pour l'intitulé de ces *parties*). Cette section est une étude de quelques unes de ces *parties*. La première sous-section résume la *partie* générique des interfaces d'accès aux données. La sous-section suivante montre comment cette *partie* générique est traduite dans différents langage avec une attention plus particulière pour le langage Java.

1.1 ISO 10303-22 : Standard Data Access Interface

La *partie* 22 : Standard Data Access Interface de STEP décrit de manière générique l'interface d'échanges de données STEP. C'est à la fois une description des structures nécessaires pour lire un modèle Express et une spécification de toutes les fonctions que doivent implanter les applications souhaitant manipuler une base de données contenant des instances de ce modèle Express. En se référant au document [36], on peut voir que la *partie* 22 inclut les points suivants :

- Accès et manipulation d'instances d'*entités* décrites en utilisant le langage de description de données Express
- Accès simultané à différents dépôts de données par une seule application
- Possibilité de regrouper les opérations décrites dans des structures qui peuvent ensuite être sauvées ou annulées
- Accès aux données qui décrivent les données (méta-représentation). Cette méta-représentation peut alors être manipulées par une application
- Possibilité de lancer la validation des contraintes décrites dans le modèle Express
- Aide pour gérer les relations de dépendance entre les instances d'*entités*
- Possibilité de décrire des collections logiques d'instances d'*entités* pour définir des ensembles sur lesquels les références sont autorisées
- Possibilité de décrire des collections logiques d'instances d'*entités* sur lesquelles les règles globales sont validées
- Aide pour l'utilisation de données créées dans le contexte d'un autre schéma

La *partie* 22 est elle même décrite en Express et en Express-G. Pour plus de détail, on pourra se référer à [36]. Pour en donner un rapide résumé, la *partie* 22 comprend les quatre schémas Express suivants :

- sdai_dictionary_schema : c'est la description en Express du langage Express. Par exemple, une *entité* sera définie par l'*entité* Express entity_definition
- sdai_session_schema : c'est la description des structures de données nécessaires pour gérer une session d'accès à des données STEP. Ce schéma décrit par exemple le type access_type qui est une ENUMERATION de read_only ou read_write
- sdai_population_schema : c'est la description des structures servant à l'organisation, la création, la gestion d'instances d'*entités* Express
- sdai_parameter_data_schema : c'est la description des structures qui sont passées en paramètre ou manipulées dans cette interface SDAI. Il n'est pas nécessaire d'implanter

ce schéma. Il sert pour la description des opérations et définitions du SDAI

A partir de cette structure générique des interfaces d'accès aux données STEP, la norme en définit l'implantation dans la plupart des langages informatiques connus. Cette implantation est décrite dans la sous-section suivante.

1.2 Implantation dans un langage donné

Les *parties* 23 à 29 de STEP sont les traductions de la *partie* 22 décrite dans la section précédente dans un langage informatique donné. Par exemple la *partie* 23 est la traduction en C++, la *partie* 27 celle en Java. Le document donné en Annexe B (Step On A Page *implementation methods*) montre la totalité des traductions vers les langages actuellement disponibles et leur stade de normalisation dans STEP.

Ces *parties* spécifient la traduction de l'interface générique d'accès aux données (*partie* 22) dans un langage particulier. Pour C++, les classes spécifiées sont des classes abstraites qu'il faut étendre. Ainsi, si deux applications utilisent, lors d'accès aux données, ces classes, elles seront compatibles entre elles même si leurs implantations sont différentes. En ce qui concerne Java, ce sont des interfaces qui sont spécifiées. En Java les interfaces sont l'équivalent des classes abstraites en C++ avec des contraintes supplémentaires : les méthodes ne peuvent pas avoir de corps, les variables ne peuvent être que des variables de classe constantes (publiques statiques finales).

Il y a toujours deux manières d'implanter ces *parties* : avec ou sans utilisation de classes de la méta-représentation. Lorsqu'on utilise la méta-représentation, on fait une implantation en "late binding". Les instances ne sont que des instances de la méta-représentation. Lorsqu'on ne se sert pas de la méta-représentation, on fait du "early binding". Les instances sont alors des instances de classes utilisatrices. Par exemple, si on prend l'*entité* Express suivante :

```
ENTITY point;
  x : REAL;
  y : REAL;
END_ENTITY;
```

et si on échange le fichier suivant :

```
#10=point(23.0,3.0);
```

Dans l'implantation "late binding" en Java, on doit disposer de :
- l'interface EEntity_definition qui représente une *entité* Express
- la classe CEntity_definition qui étend cette interface
- une instance de cette classe qui a pour variable de classe une instance de la classe attribut dont le nom est x et dont le domaine est un double

Dans l'implantation "early binding" en Java, on doit disposer de :
- l'interface EPoint qui représente le point
- la classe CPoint qui implémente cette interface
- une instance de CPoint qui contient comme variable de classe une variable x de valeur 23.0 et une variable y de valeur 3.0

1.3 Choix de la méthode et du langage de mise en oeuvre

Le choix que nous avons fait est d'implanter une combinaison des deux méthodes : nous pouvons ainsi disposer, à chaque session d'utilisation, à la fois de la méta-représentation et des classes utilisatrices provenant du "early binding".

Un autre choix nécessaire pour l'implantation est celui du langage. Nous avons choisi le langage Java. Le langage Java est apparu assez récemment (1995) et n'a donc pas encore atteint le niveau de maturité de langages tels que C++ mais il a acquis à présent une bonne stabilité. Il a des avantages certains par rapport à tous les langages existants. Le premier de ces avantages est sa structure purement orientée objet : contrairement à C++, il oblige à s'interresser à l'aspect structurel de l'application développée car il ne permet pas de mettre au même niveau objets et fonctions. Le principal avantage de Java reste néanmoins sa portabilité : c'est un langage indépendant du système d'exploitation. A cela s'ajoutent : la possibilité d'utilisation du parallélisme (multithreading), la possibilité de créer rapidement un environnement graphique convivial (fenêtres, souris, ..), l'existence d'une aide facile à consulter (format HTML), la gestion automatique de la mémoire et une relative sécurité pour les applications en réseaux.

Le principal défaut de Java pour les applications nécessitant des calculs complexes est sa lenteur d'exécution. Ce défaut vient du fait que c'est un langage interprété et devrait être corrigé si on utilise des compilateurs binaires au lieu du compilateur bytecode standard.

Dans le cadre de cette étude, nous nous interressons à l'aspect structurel des applications de calcul, sans chercher réellement à augmenter la vitesse de calcul ni à mettre en oeuvre des méthodes numériques optimales. Java convient donc bien pour la mise en oeuvre.

2 Outils existants

Il existe dans le commerce des outils permettant de créer un Protocole d'application STEP et de l'implanter dans les langages C++ et Java.

On peut citer par exemple :
- en C++ : EPM Express Data Manager [28], Step-tools Express Developper [29], Winstep [30]
- en Java : LKSoft J-SDAI [31]

L'implantation en "late binding" ne nécessite que le codage d'un nombre limité de classes : celles de la méta-représentation. Par contre, l'implantation en "early binding" nécessite le codage de toutes les *entités* décrites dans le *Protocole d'Application* que l'on veut utiliser. En tenant compte du fait que chaque *entité* STEP doit donner lieu à au moins 3 classes Java (une interface, une classe d'implémentation, une classe d'agrégation de classes du même type), nous avons calculé qu'il y avait plus de 2000 classes à coder. Cette opération peut-être faite "à la main" pour un Protocole d'Application donné mais devra être reproduite si on veut implanter un autre Protocole d'Application pour un autre domaine.

C'est pourquoi il nous a paru plus avantageux d'étudier comment on peut faire la traduction automatique Express vers Java, tout en respectant les interfaces spécifiées dans la *partie* 27 : Java language Binding. Les outils de génération de classes C++ à partir d'Express existent mais pour le langage Java, ils sont encore à l'état de prototypes et peu accessibles puisque la *partie* 27 de STEP n'est qu'à l'état de proposition, c'est à dire qu'elle n'est pas encore une norme à part entière. L'étude et la réalisation d'un générateur de code Express vers Java est décrite dans la section suivante.

3 Génération Automatique de Code

Le but de cette section est de montrer qu'il est possible de minimiser la quantité de code à écrire pour implanter un modèle de données écrit en EXPRESS tout en respectant les spécifications données dans STEP.

Nous présenterons en premier lieu les différentes techniques et outils les plus connus dans le domaine de la compilation. Puis nous justifierons notre choix de l'outil Java Compiler Compiler. Enfin nous présenterons l'application ExpToJava que nous avons développée dans le but de générer du code source Java à partir d'un modèle de données STEP décrit en EXPRESS.

3.1 Langages de programmation et langages de description

Les langages comme C, C++ sont des langages de haut niveau qui permettent plus facilement au programmeur de spécifier les actions que doit accomplir la machine. Pour pouvoir les traduire en langage machine, on utilise des compilateurs qui permettent de transformer des lignes écrites dans ces langages en langage binaire.

Les langages comme Java sont interprétés : le compilateur de Java ne transforme pas les lignes de code en langage binaire mais en un langage appelé "byte code" qui sera à son tour interprété par un programme : la machine virtuelle ou Java VM. C'est cette machine virtuelle qui exécute les commandes demandées. L'intérêt de cette technologie est que le "byte code" généré est alors commun à tous les systèmes d'exploitation. La différence vient alors des différentes implantations de machines virtuelles dans chaque système d'exploitation.

Enfin, un dernier type de langage est le langage de description. EXPRESS en est un exemple. Un langage de description sert principalement à créer des librairies qui à leur tour pourront

être utilisées par des programmes. EXPRESS est adapté pour la description de structures de données : il utilise les notions d'héritage, de collection, de référence qui donnent une structure suffisamment complète pour décrire des données complexes. Dans le cas d'EXPRESS, la nouveauté du langage est une difficulté supplémentaire lorsqu'on souhaite l'utiliser : il y a peu de compilateurs accessibles. Pour pouvoir l'utiliser, il est nécessaire de le traduire vers un autre langage : c'est le rôle d'un générateur de code.

La génération de code permet de traduire un langage en un autre langage. En général, le premier langage est un langage peu courant et le deuxième langage est un langage informatique parmi les plus utilisés (C la plupart du temps).

Dans notre cas, nous souhaiterions automatiser la traduction d'un modèle Express vers le langage Java. Pour cela, nous utilisons la même technique que celle utilisée dans les compilateurs pour faire de la génération de code. En effet, la lecture d'un langage de description tel que Express peut-être optimisée par ces méthodes. Ainsi, nous lirons un ou des fichiers Express selon des techniques de compilation connues dans le domaine de la programmation : une analyse lexicale contrôlée par une analyse syntaxique et sémantique. C'est ce processus que nous détaillerons dans la sous-section suivante.

3.2 Analyse d'un langage

L'analyse d'un langage se fait en trois *parties* : une analyse lexicale, une analyse syntaxique et une analyse sémantique. Chacune des étapes nécessite des résultats de l'étape précédente.

La première étape est l'analyse lexicale. Elle permet d'identifier des unités lexicales (token) ou plus simplement des mots, d'éliminer les caractères de séparation (espaces, retour chariot,...), de différentier les directives de compilation des éléments du langage. Par exemple, les chaînes de caractères "class" et "512" seront des tokens pour le langage C++.

La deuxième étape est l'analyse syntaxique. Elle met en place les règles de grammaire en indiquant tous les cas possibles de placement des unités lexicales données par la première analyse. Chaque description de placement est une règle de production. C'est la définition de la structure du langage. Par exemple, un analyseur syntaxique (ou parseur) du langage C saura comment sont construites les expressions, les instructions, les déclarations de variables, les appels de fonctions,...

La dernière étape est l'analyse sémantique. Cette phase est un contrôle de types pour vérifier le sens de la structure. Un analyseur sémantique interdira par exemple l'affectation de variables de type différents. Par exemple :

```
int i;
String j;
i=j;
```

est une entrée qui sera acceptée par les analyseurs lexical et syntaxique mais sera refusée par l'analyseur sémantique.

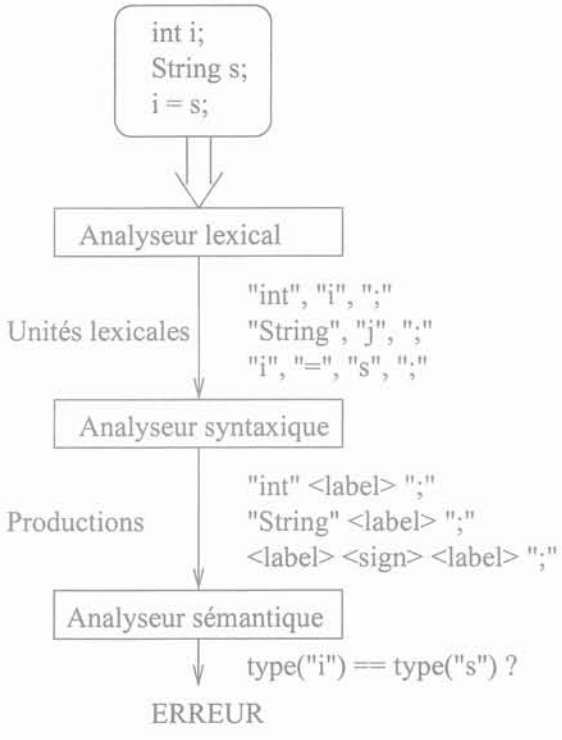


FIG. 4.1 – Etapes d'analyse d'un langage

En résumé (voir Figure 4.1), il faut donc un analyseur lexical ou scanner qui rassemble les lettres pour en faire des unités lexicales. Ce scanner transmet au fur et à mesure de la lecture les unités lexicales lues à un analyseur syntaxique. L'analyseur syntaxique tente alors de faire coïncider la séquence d'unités lexicales avec des productions syntaxiques suivant les règles qui lui ont été prédéfinies. L'analyse syntaxique peut générer un arbre dont chaque production est un noeud. Après ou au fur et à mesure de l'analyse syntaxique, un analyseur sémantique se charge de vérifier la cohérence du contenu de ce qui est lu.

Pour pouvoir ensuite intégrer ces données dans des programmes plus évolués, il faut les traduire dans un langage informatique : c'est la production de code.

3.3 Production de code

Il s'agit de produire les instructions du langage source dans le langage cible. En général, les instructions seront générées pour une machine abstraite (virtuelle). Puis ensuite on fera la traduction de ces instructions en des instructions directement exécutables par la machine réelle sur laquelle on veut que le compilateur s'exécute. Ainsi, le portage du compilateur sera facilité, car la traduction en code cible virtuel sera faite une fois pour toutes, indépendamment de la machine cible réelle. Il ne reste plus ensuite qu'à étudier les problèmes spécifiques à la machine cible, et non plus les problèmes de reconnaissance du programme.

Dans notre cas, le langage cible est Java qui est déjà un langage interprété. Nous nous arrêterons donc à la génération d'instructions. Le compilateur Java fera le reste, c'est à dire la

compilation vers le "byte code" compréhensible par la machine virtuelle de Java.

3.4 Outils de création de compilateurs

Les outils qui permettent de créer des analyseurs lexicaux et syntaxiques en langage source C existent depuis très longtemps : ce sont les outils Lex (Lexical Analyser) et Yacc (Yet Another Compiler Compiler). Nous en donnerons la description dans la première *partie* de cette sous section. Un outil un peu plus récent, JavaCC, permet de générer à la fois un analyseur lexical et syntaxique.

3.4.1 Lex et Yacc

Lex et Yacc sont des outils de création de compilateurs. Lex permet de faire l'analyse lexicale d'un langage donné et Yacc permet de faire une analyse syntaxique. A titre d'exemple, nous donnons dans l'annexe E les codes Lex et Yacc pour faire une calculatrice. Ces outils, un peu complexes à appréhender ont été utilisés pour générer de nombreux compilateurs.

Lex est un utilitaire d'Unix, son équivalent Flex est un utilitaire de GNU. Lex et Flex acceptent en entrée des spécifications d'unités lexicales sous forme de définitions régulières et produit un programme écrit dans un langage de haut niveau (le langage C) qui, une fois compilé, reconnaît ces unités lexicales (ce programme est donc un analyseur lexical). Le fichier de spécifications Lex contient des expressions régulières suivies d'actions (règles de traduction). L'exécutable obtenu lit le texte d'entrée caractère par caractère jusqu'à ce qu'il trouve le plus long préfixe du texte d'entrée qui corresponde à l'une des expressions régulières. Dans le cas où plusieurs règles sont possibles, c'est la première règle rencontrée (de haut en bas) qui l'emporte. Il exécute alors l'action correspondante. Dans le cas où aucune règle ne peut être sélectionnée, l'action par défaut consiste à copier le caractère du fichier d'entrée en sortie.

Les analyseurs syntaxiques sont des outils qui construisent automatiquement une table d'analyse à partir d'une grammaire donnée. Yacc est un utilitaire d'Unix, Bison est son équivalent sous GNU. Yacc et Bison acceptent en entrée la description d'un langage sous forme de règles de productions et produit un programme écrit dans un langage de haut niveau (le langage C) qui, une fois compilé, reconnaît des phrases de ce langage (ce programme est donc un analyseur syntaxique). Yacc signifie Yet Another Compiler Compiler, c'est à dire encore un compilateur de compilateur. Cela n'est pas tout à fait exact, Yacc et Bison seuls ne permettent pas d'écrire un compilateur, il faut rajouter une analyse lexicale (à l'aide de Lex ou Flex par exemple) ainsi que des actions sémantiques pour l'analyse sémantique et la génération de code.

Nous avons tenté de créer un compilateur Express vers C++ avec les outils Flex et Bison dans un projet d'étudiants. Le résultat donne un compilateur dont le code source est peu modifiable et finalement peu facile à entretenir du fait de la complexité des notations Flex et Bison. Mais les premiers compilateurs d'Express vers C [27] ont été créés avec ces outils et servent pour l'implantation de STEP en C et C++.

3.4.2 JavaCC

Java Compiler Compiler [33] est un outil qui permet également de générer des analyseurs lexicaux et syntaxiques (parseurs). Le code source généré est en Java et non en C comme c'est le cas pour Lex et Yacc. Le langage utilisé pour décrire les fichiers d'entrée de JavaCC est un langage qui ressemble fortement à l'EBNF (Extended Backus-Naur Form).

Par exemple, un morceau de la description JavaCC d'EXPRESS sera :

```
/********** PRODUCTION 197 **********/
void entity_head() : {}
{
        <ENTITY> entity_id() (subsuper())? <SEMICOLON>
}
```

Le fichier en Annexe D montre à titre d'exemple une calculatrice faite en JavaCC.

Les parseurs générés par JavaCC [33] sont des parseurs récursifs descendant (contrairement à ceux générés par Yacc qui sont récursifs montants). Ils permettent de faire l'analyse lexicale et syntaxique de grammaires LL(k). Le premier L signifie Left to Right scanning. Le deuxième L signifie Left most derivation. k est le nombre de tokens lus "en avance". Par exemple, une grammaire qui dit "A : a et B : aa" n'est pas LL car il y a ambiguïté entre l'état A et l'état B quel que soit le nombre de tokens lus en avance.

La grammaire du langage Express données par l'ISO dans la *partie* 11 de STEP [34] est une grammaire LL(2). Ce qui nous autorise à utiliser JavaCC.

3.5 Procédé de traduction d'Express vers Java

L'application ExpToJava que nous avons développée crée un ensemble de classes Java à partir d'un modèle écrit en EXPRESS. Les classes générées reflètent alors les structures d'héritage du modèle EXPRESS.

Son but principal est d'aider au développement d'applications utilisant des données STEP. Le code généré sera donc destiné à être utilisé avec ou sans modifications.

Le scanner et analyseur syntaxique écrits en JavaCC (express.jj) a été mis au point en collaboration avec un étudiant de PSU (PennState University). Le code source est donné en Annexe F. Une utilisation simple de l'analyseur syntaxique produit est de vérifier si un modèle écrit en Express est "légal", c'est à dire conforme à la grammaire Express donnée par l'ISO dans la *partie* 11 de STEP [34].

A cet analyseur, nous avons ajouté l'analyseur sémantique pour pouvoir instancier une méta-représentation du modèle Express lu. Il s'agit entre autre de connecter la lecture à la

création d'objets de la méta-représentation d'Express. La librairie des classes de cette méta représentation est un ensemble d'environ 140 classes contenues dans le package SDAI.lang. A ces classes s'ajoutent également une dizaine d'autres classes pour gérer l'entrée sortie de fichiers. L'analyse sémantique est faite en deux fois : lecture simple puis cohérence entre les références. L'analyse sémantique est une *partie* assez longue de la réalisation du compilateur : sa prise en compte double le volume de code de l'analyseur lexical et syntaxique.

Les fichiers sources JavaCC (.jj) sont ensuite compilés par l'application JavaCC en fichiers sources Java. Ce sont alors les sources Java de l'analyseur sémantique. L'application finale est obtenue après compilation des sources Java générés et de la librairie de la méta-représentation d'Express.

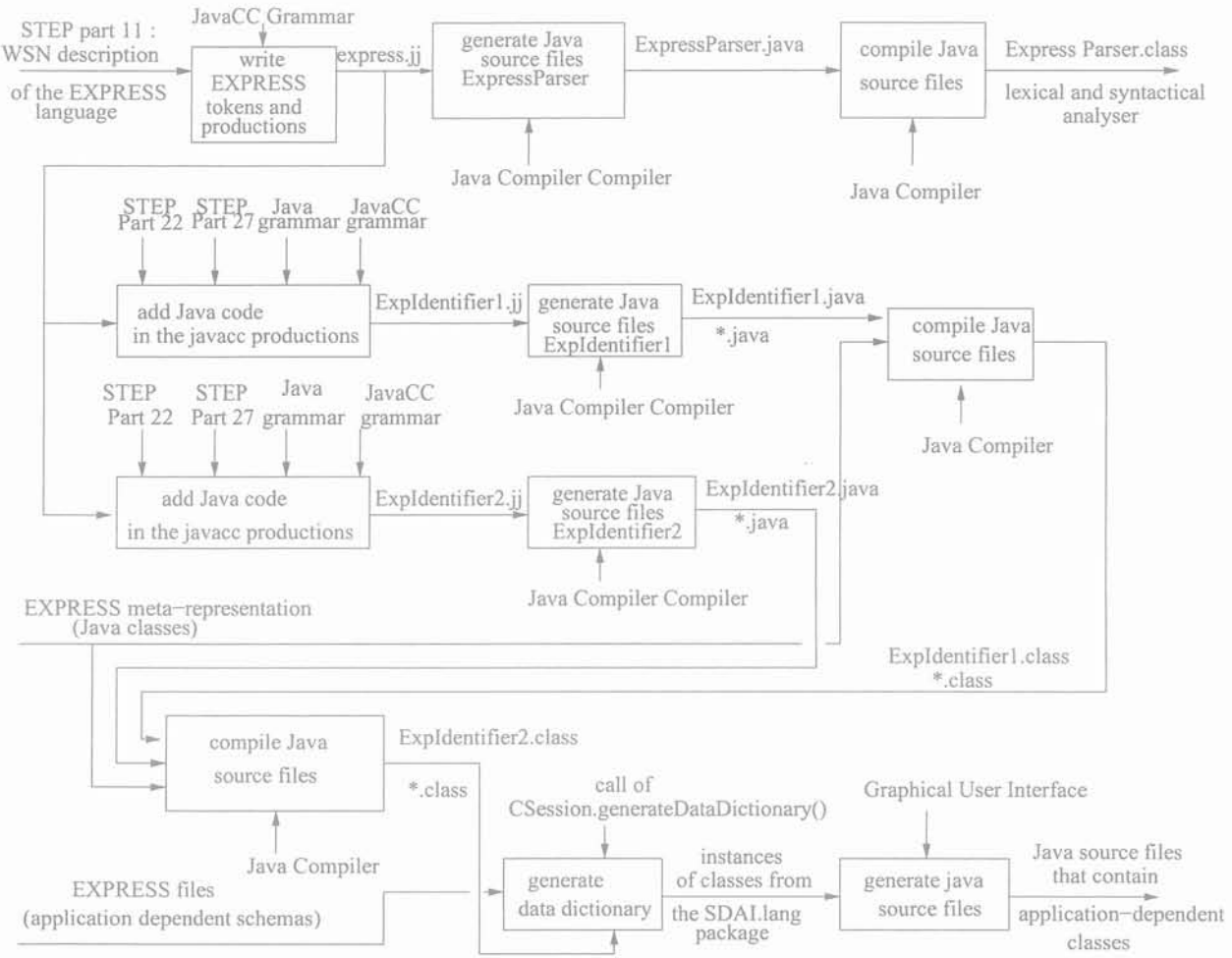


FIG. 4.2 – Un compilateur Express vers Java : processus

Le processus de développement de ce parseur est résumé dans la figure 4.2. La notation utilisée dans cette figure est semblable à SADT :

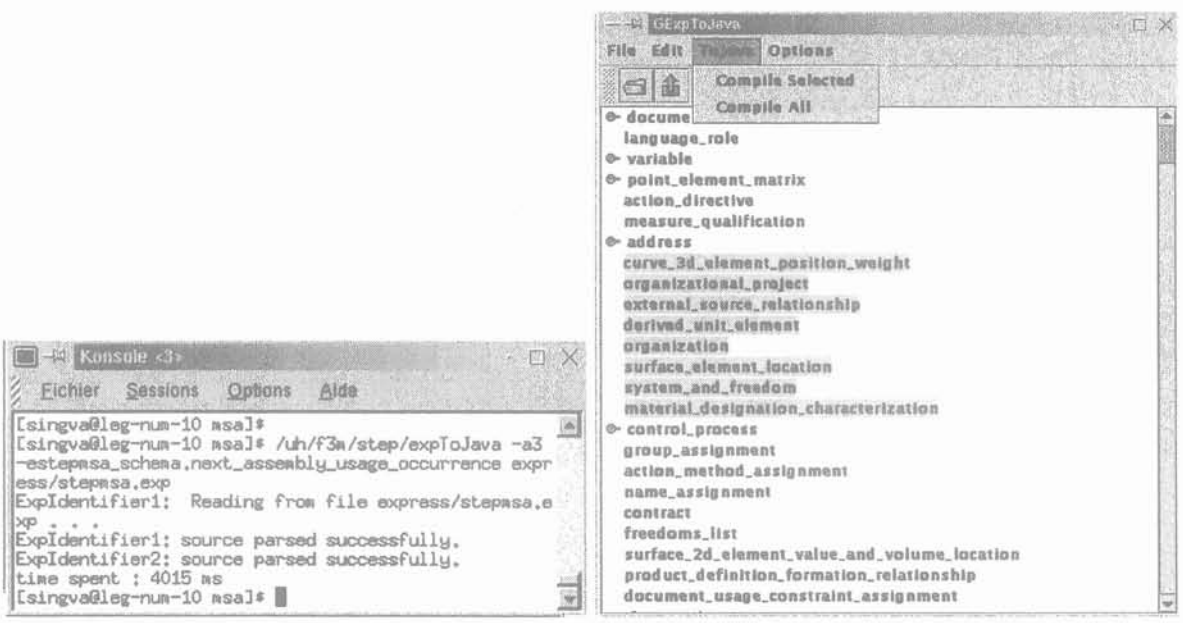
- les activités sont représentées par des boites,

- les flux de données par des flèches,
- les données "du haut" sont des controles,
- les données "du bas" sont les mécanismes ou outils utilisés.
- les données "de gauche" sont les entrées,
- les données "de droite" sont les sorties.

On remarquera que ce processus nécessite deux parseurs. En effet, la gestion de cohérence ne peut être faite totalement en une seule lecture : certaines références sont faites sur des *entités* postérieures à celles déjà lues. Ainsi, le premier parseur instancie des classes de la métareprésentation "vides" de toute données et le deuxième parseur "remplit" ces instances en créant les liens de références.

3.6 Interface graphique pour la compilation Express vers Java

La sous section précédente a présenté le processus de développement d'un compilateur. Ce compilateur peut être utilisé en ligne avec des options de compilation (tel que le nom de l'*entité* que l'on veut compiler) pour une utilisation moins consommatrice de ressources, c'est l'application ExpressToJava (voir Figure 4.3 (a)) ou avec une interface graphique, c'est l'application GExpressToJava (voir Figure 4.3 (b)).



(a) en ligne (b) avec IHM

FIG. 4.3 – Compilateurs

L'intérêt d'une interface graphique est que l'utilisateur (développeur de l'application utilisant un nouveau protocole STEP) peut visualiser en une fois le modèle qui l'intéresse, faire des recherche sur les *entités* (par nom).

3.7 Exemple de code généré

En utilisant le générateur de code précédent à partir de la déclaration en EXPRESS suivante :

```
ENTITY cartesian_point
SUBTYPE OF (point);
  coordinates: SET[2:3] OF REAL;
END_ENTITY;
```

on obtient les fichiers Java suivants :
Interface ECartesian_point :

```
package SDAI.SStepmsa_schema;
import SDAI.lang.*;
public interface ECartesian_point extends EPoint {
   // methods for attribute: coordinates
   public A_double getCoordinates() throws SdaiException;
   public A_double createCoordinates() throws SdaiException;
   public boolean testCoordinates() throws SdaiException;
   public void unsetCoordinates() throws SdaiException;
}
```

Classe CCartesian_point :

```
package SDAI.SStepmsa_schema;
import SDAI.lang.*;

public class CCartesian_point extends CPoint implements ECartesian_point {

   //------ field and methods for attribute: coordinates

   protected A_double coordinates;
   protected boolean coordinatesSet;

   public void setCoordinates(Object value)
          throws SdaiException
   {
      this.coordinates = (A_double)value;
      coordinatesSet = true;
   }

   [...]
}
```

Aggrégats ACartesian_point :

```
package SDAI.SStepmsa_schema;
import SDAI.lang.*;

/**
 * The ACartesian_point class shall represent an aggregate of the SDAI
 * entity Cartesian_point
 * specified in ISO 10303-22:6.4.13.
 */
public class ACartesian_point implements Aggregate {

  protected ECartesian_point[] elementData;
  protected int      elementCount;
  protected int      capacityIncrement;

  //---------- constructors -----------------------------------------------

  public ACartesian_point()
  {
     super(10);
  }

  //---------- needed methods ---------------------------------------------

  public void increaseCapacity(int increment)
  {
      int newCapacity =
          increment > 0 ? (elementData.length+increment) :
                          2*elementData.length;
      ECartesian_point[] newElementData = new ECartesian_point[newCapacity];
      System.arraycopy(elementData, 0, newElementData, 0, elementCount);
      elementData = newElementData;
  }

[...]

}
```

3.8 Ajout de code utilisateur et mise à jour

Le code généré n'est pas suffisant pour créer une application capable de résoudre les équations de la physique : il faut encore ajouter les méthodes numériques. Ces méthodes ne sont bien évidemment pas générées automatiquement : c'est le travail du programmeur.

Ainsi dans un premier temps, le modèle Java est généré par le générateur de code Express vers Java et dans un deuxième temps, le programmeur utilise les classes générées et y ajoute du code supplémentaire pour l'analyse numérique.

Cependant, il est parfois nécessaire de changer le modèle de données Express après ajout de code dans les fichiers générés. Il faut alors garantir qu'une nouvelle génération de code n'effacera pas le code ajouté. Une mise à jour partielle du modèle doit tenir compte des modifications apportées à une classe.

Pour la mise à jour, nous avons identifié différentes méthodes.

La première est la génération automatique de classes abstraites ou interfaces (au sens Java) uniquement (voir Figure 4.4). Le problème avec cette méthode est que la génération de code n'aide en rien le programmeur, mis à part l'obligation de suivre la structure donnée en interface.

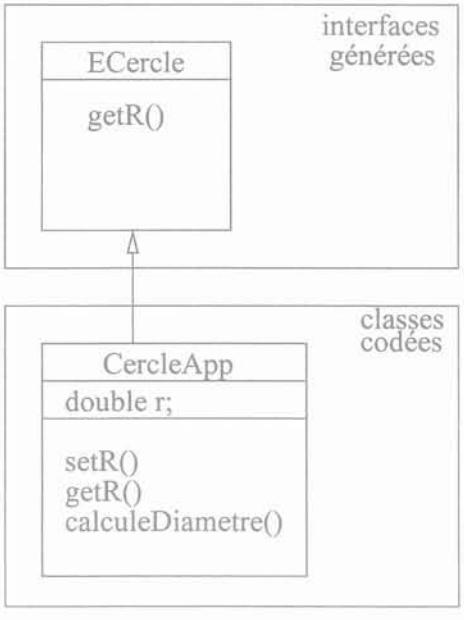


FIG. 4.4 – Implémentation d'interfaces générées

Une autre méthode est l'encapsulation des classes générées. Ce sont des classes qui sont générées mais on en interdit la modification. Pour l'ajout des méthodes numériques, on se sert des propriétés d'héritage/dérivation du langage cible. Ainsi, si par exemple (voir Figure 4.5), on génère la classe Cercle, il faudra créer une classe CercleApp qui dérive de Cercle mais à laquelle on ajoute par exemple la méthode de calcule de diamètre calculeD(). La classe dérivée aura alors toutes les propriétés de la classe générée avec en plus les méthodes numériques ajoutées.

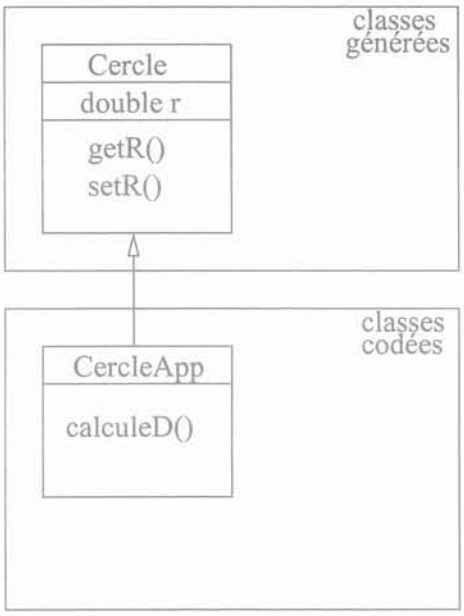


FIG. 4.5 – Encapsulation des classes générées

Le problème avec cette méthode est le multihéritage : le modèle Express a une structure d'héritage assez complexe. La figure 4.6 montre le cas où la méthode d'encapsulation des classes générées impose l'utilisation d'héritage multiple alors que le modèle Express n'utilise qu'un simple héritage. Dans cet exemple, SubClass dérive de SuperClass dans le modèle généré. SuperClassApp est l'implantation de SuperClass donc en dérive. SubClassApp est l'implantation de SubClass donc en dérive. Or, il faudrait également que SubClassApp dérive de SuperClassApp pour que le modèle codé soit lui aussi cohérent. Le modèle codé doit donc utiliser un multihéritage alors que le modèle généré n'en utilise pas. Java n'autorise le multihéritage que pour les interfaces dont on ne peut coder le corps des méthodes. En Java, le multihéritage des classes, même abstraites, n'est pas admis.

Une troisième méthode est l'utilisation de balises signalant au compilateur que cette zone doit être transféré intégralement dans le nouveau modèle généré. Pour cela, le code ajouté dans un fichier java généré doit être balisé de telle sorte qu'une modification du modèle Express puisse garantir la conservation du code ajouté. Pour cela, on génère, en plus des données du modèle, des zones dans lesquelles le programmeur peut ajouter du code et on interdit l'écriture en dehors de ces zones dans un éditeur spécialisé.

Certains éditeurs peuvent empêcher l'écriture dans une portion particulière du fichier. C'est le cas de l'outil XEmacs si on lui ajoute le module [26]. Avec ce module, l'éditeur de texte XEmacs n'autorise l'écriture que dans les zones situées entre les lignes marquées par :

```
//---- Start editable code block: import statement
[...]
```

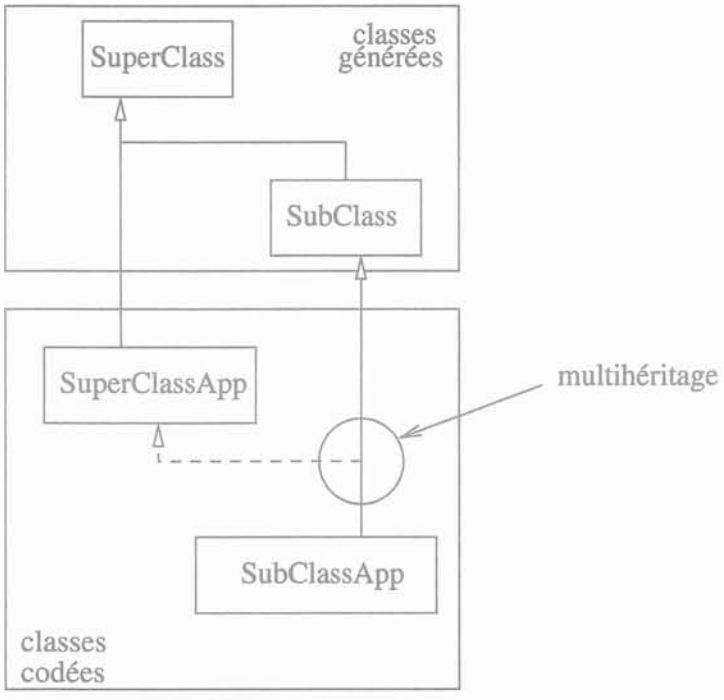


FIG. 4.6 – Encapsulation des classes générées

```
//---- End editable code block: import statement
```

La Figure 4.7 montre un exemple de fichier généré avec une zone de mise à jour. L'écriture dans les zones générées peut toujours être autorisée à nouveau par la mise à vraie d'une option de l'éditeur. Toute nouvelle génération de code effacera alors les modifications apportées dans la zone générée.

4 Outils pour la mise en oeuvre d'un *Protocole d'Application*

Le générateur de code décrit précédemment permet de mettre en oeuvre de manière semi-automatique un quelconque modèle Express. Or un *Protocole d'Application* dans le cas de STEP est un ensemble assez rigide de modèle Express et de règles de mise en oeuvre et de test de conformité. Un générateur n'est donc pas suffisant pour une complète implantation de STEP.

Cette section décrit les autres éléments logiciels utilisés dans le cas de la mise en oeuvre d'un *Protocole d'Application* STEP en général. La première sous-section indique les documents de base dont il faut disposer. La deuxième sous-section décrit les modules que nous avons développés, soit en Java, soit à partir d'outils de logiciels libres GNU. La dernière sous-section

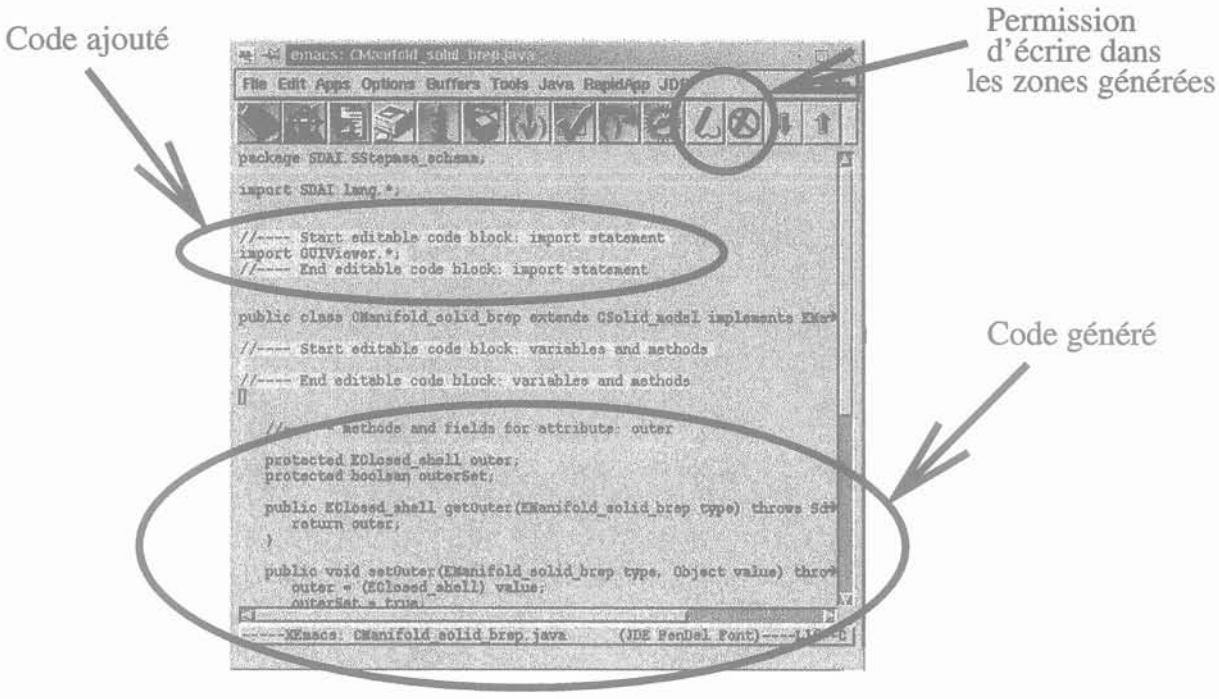


FIG. 4.7 – Code généré avec mise à jour

donne les spécifications pour un environnement complet de développement et de mise en oeuvre d'un modèle STEP.

4.1 Documents fournis par l'ISO

4.1.1 Description détaillée du *Protocole d'Application*

Il faut en tout premier lieu la description en texte clair du protocole d'application. C'est essentiel pour bien comprendre ce que représentent les structures. Par exemple, on peut trouver dans la *partie* 42 (Geometry and Topology Representation) la description d'une surface B-spline. Si on n'est pas familier avec le domaine de la géométrie, il peut être difficile de "deviner" à partir uniquement de la structure ce que représente l'entier "u_degree" ! Ce document, édité par l'ISO, peut-être considéré comme le contrat qui spécifie :

- le cadre exact du domaine couvert dans les échanges avec des précisions sur les domaines non couverts
- toutes les structures de données qui peuvent être utilisées dans les échanges
- les tests qu'il faut passer en lecture/écriture/échanges pour assurer la conformité des outils logiciels utilisés

4.1.2 Fichier Express du modèle proposé

C'est un fichier qui décrit le modèle Express du *Protocole d'Application*. Pour ce fichier, il y a deux possibilités : le format court et le format long. La différence entre ces deux formats est dans l'importation implicite ou explicite d'*entités* provenant d'autres schéma. Dans le format court, ces *entités* sont référencées par le mot clé "REFERENCE" ou "USE" alors que dans le format long, elles sont recopiées directement dans le schéma courant.

On choisit le format court lorsque l'on dispose déjà de *parties* implémentées. Il est avantageux de coder les *Ressources Génériques* puisque tous les *Protocoles d'Applications* les utilisent. Mais parfois, et c'est notre cas, il est nécessaire également d'implanter les *parties* appartenant aux *Ressources d'Application Intégrées* (*partie* 104 sur les éléments finis).

On choisit le format long lorsqu'on ne veut développer qu'un seul *Protocole d'Application*. En effet, avec ce format, on dispose de toutes les structures nécessaires et suffisantes.

4.2 Outils logiciels

4.2.1 Librairie pour la méta représentation d'Express

Les structures décrites en Express ont elles-mêmes une méta-représentation en Express dans la *partie* 22 et dans un langage donné dans les différentes *parties* de mise en oeuvre (*parties* 23 pour C++, 27 pour Java par exemple). En ce qui concerne le langage Java, c'est une librairie d'environ 140 interfaces dont les spécifications sont données dans la *partie* 27 [42].

Les spécifications en Java ne donnent que les interfaces (au sens Java). L'implantation devra donc implémenter ces interfaces dans des classes.

Par exemple, la *partie* 27 donne les spécifications suivantes pour la méta-représentation d'une *entité* Express :

```
package SDAI.lang;

public interface EEntity_definition extends ENamed_type {
  AEntity_definition getSupertypes() throws SdaiException;
  boolean getComplex() throws SdaiException;
  boolean getInstantiable() throws SdaiException;
  boolean getIndependent() throws SdaiException;
  AAttribute getAttributes() throws SdaiException;
  AUniqueness_rule getUniqueness_rules() throws SdaiException;
  AGlobal_rule getGlobal_rules() throws SdaiException;

  boolean isSubtypeOf(EEntity_definition CompType) throws SdaiException;
  boolean isSdaiSubtypeOf(EEntity_definition CompType) throws SdaiException;
  boolean isDomainEquivalentWith(EEntity_definition CompType)
          throws SdaiException;
}
```

Notre implantation de la méta-représentation sera la classe CEntity_definition qui implémentera (au sens Java) cette interface, dont nous donnons une partie du code ci-dessous :

```
package SDAI.lang;

import java.util.Vector;
import java.util.Enumeration;
[...]

public class CEntity_definition
       extends CNamed_type
       implements EEntity_definition
{
    public AEntity_definition supertypes;
    protected boolean complex;
    protected boolean instantiable;
    [...]
    public CEntity_definition(String name)
    {
      super(name);
      supertypes = new AEntity_definition();
      complex      = false;
      instantiable = true;
      independent  = true;

      expAttrs = new Vector();
      derAttrs = new Vector();
      invAttrs = new Vector();
      subtypes = new Vector();
    }
    [...]
    public boolean isSubtypeOf(EEntity_definition compType)
           throws SdaiException
    {
      if (compType==this || supertypes.isMember(compType))
            return true;
      SdaiIterator it = supertypes.createIterator();
      while(it.next())
      {
        EEntity_definition current = supertypes.getCurrentMember(it);
        if (current.isSubtypeOf(compType))
            return true;
      }
      return false;
    }
    [...]
}
```

Cette implantation contient également des méthodes pour la génération automatique de code. En effet, l'analyseur sémantique est chargé d'instancier cette classe et ce sont ces instances qui contiennent les données lues sur les fichiers Express. Il y a par exemple les méthodes :

```
public void writeJavaInterface(File path)
public void writeJavaClass(File path)
public void writeJavaAggregate(int level, File path, File directory)
```

Enfin, notons que cette implantation a été faite à partir de [42] qui est une version provisoire de la norme. Depuis, cette version a été mise à jour : changement de certains noms (ApplicationInstance en App_inst), création de nouveaux packages (package SDAI.Runtime) et réarrangement des classes mentionnées précédemment dans les nouveaux packages créés.

En conclusion, nous pouvons dire que cette méthode pour programmer la méta-représentation en Java est possible et n'est pas trop différente pour l'implantation en C++ ou dans un autre langage supporté par STEP. Mais l'implantation en Java que nous avons fournie dans ce travail devra être modifiée pour suivre les évolutions à la fois de la norme et du langage Java.

4.2.2 Constructeurs de fichiers neutres STEP avec IHM

Parce que le langage EXPRESS dispose d'une méta-représentation, on peut associer à chacune de ses structures un composant graphique particulier pour pouvoir créer des formulaires interactifs qui permettent de collecter des données et de créer des fichiers neutres STEP correspondant.

Pour mettre en oeuvre cette interface graphique de type formulaire, nous avons créé une librairie de classes de composants graphiques "neutres" qui fait le lien entre les structures EXPRESS et les structures Java.

La correspondance entre les classes "neutres" et les structures EXPRESS se fait avec la table suivante :

EXPRESS	classes "neutres"
entity	NDialogBoxDescriptor
type nommé	suivant le type
type simples	NDialogItem (combo_box, text field, choice... suivant le type)

Cela donne des formulaires graphiques créés automatiquement par lecture directe du fichier de structure EXPRESS.

Par exemple, la Figure 4.8 montre le formulaire créé pour la structure EXPRESS suivante :

```
ENTITY point;
  x: REAL;
  y: REAL;
  z: REAL;
END_ENTITY;
```

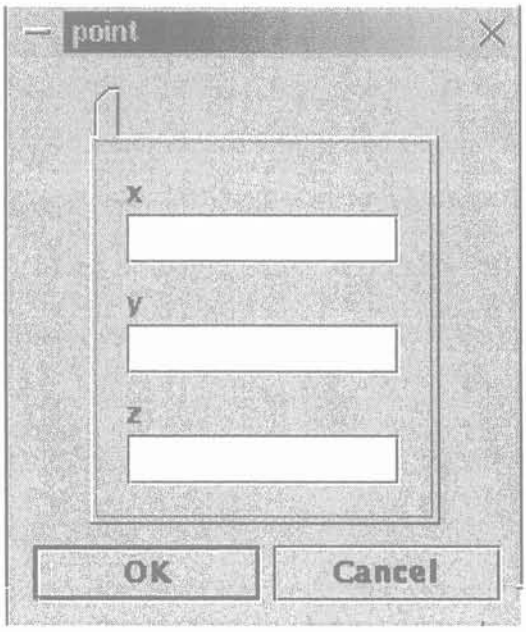


FIG. 4.8 – Formulaire créé automatiquement pour l'*entité* point

Une structure plus complexe telle que l'*entité* b_spline_surface_with_knots (surface b-spline avec points de contrôle) définie dans la *partie* géométrie (42) de STEP donne le formulaire montré dans la Figure 4.9.

```
ENTITY b_spline_surface_with_knots
   SUBTYPE OF (b_spline_surface);
   u_multiplicities  : LIST [2:?] OF INTEGER;
   v_multiplicities  : LIST [2:?] OF INTEGER;
   u_knots           : LIST [2:?] OF parameter_value;
   v_knots           : LIST [2:?] OF parameter_value;
   knot_spec         : knot_type;
END_ENTITY;
```

Ces formulaires sont remplis par l'utilisateur et l'application peut alors générer des fichiers neutres STEP contenant les données à stocker.

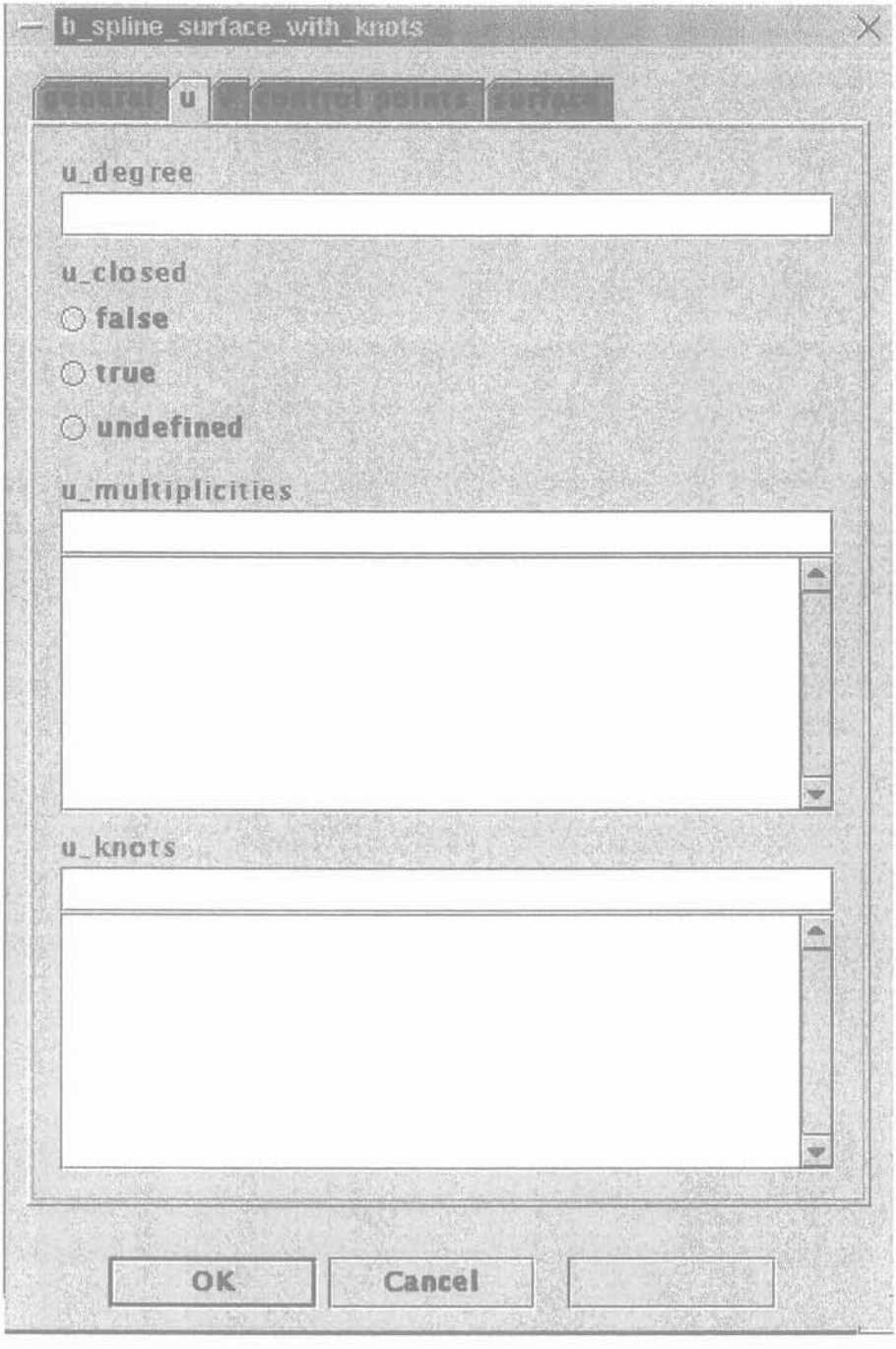


FIG. 4.9 – Formulaire créé automatiquement pour l'*entité* b_spline_surface_with_knots

4.2.3 Un éditeur pour le langage Express

Pour éditer un fichier Express, un simple éditeur de texte suffit. Mais pour faciliter à la fois la lecture et le développement d'un modèle en Express, on peut utiliser un éditeur plus évolué.

Nous avons développé un "mode Express" pour l'éditeur de GNU XEmacs, dont le code source en LISP est donné en Annexe G. Ce mode permet de décrire les mots clés du langage Express pour pouvoir les mettre en évidence sur un éditeur XEmacs.

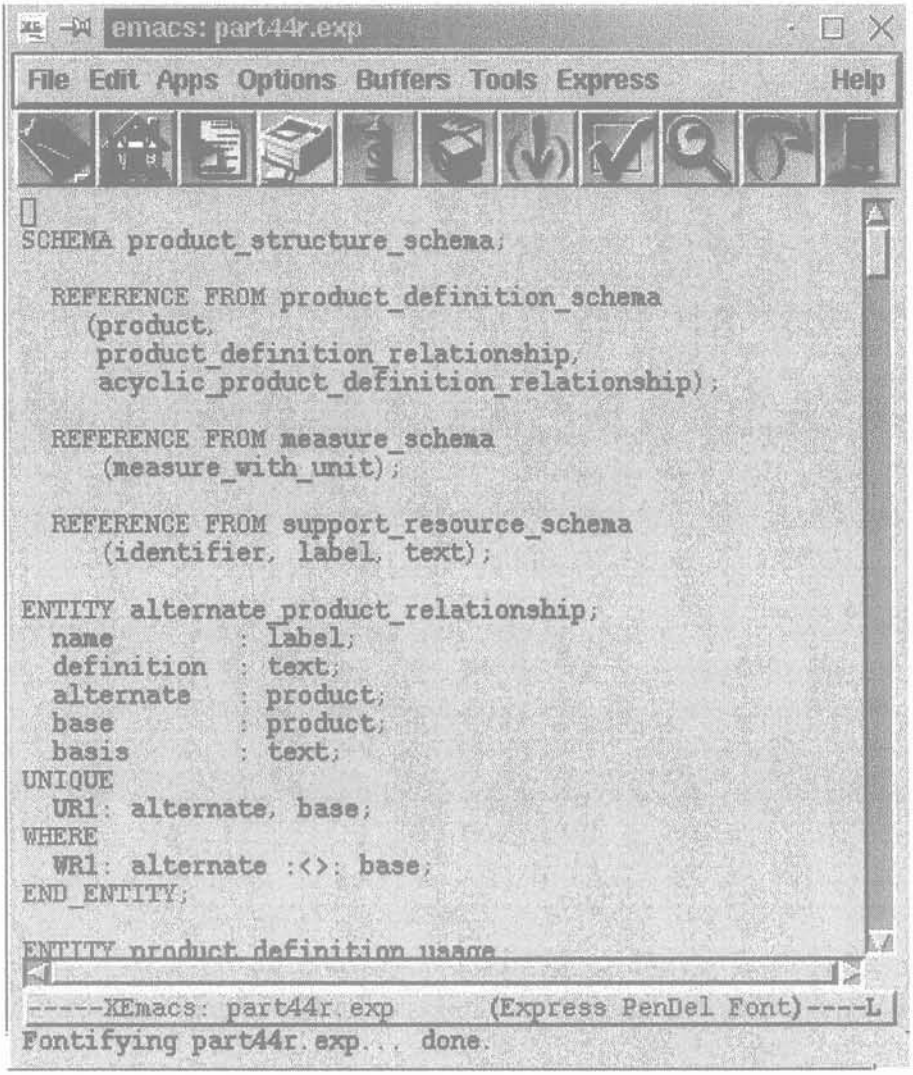


FIG. 4.10 – Un mode Express pour XEmacs

4.2.4 Un éditeur pour le langage Express-G

Le langage Express-G est graphique. L'éditeur doit donc permettre de dessiner à la fois les *entités* et les relations entre eux (dérivation, appartenance,...)

Nous avons développé un "module Express-G" pour l'application GNU Dia, dont le code source en XML est donné dans l'annexe H. Ce module permet d'utiliser les fonctionnalités de Dia en ce qui concerne les connexions entre les *entités* mais permet également de dessiner des diagrammes Express-G avec les notations standards. La figure 4.11 montre un exemple de cette application.

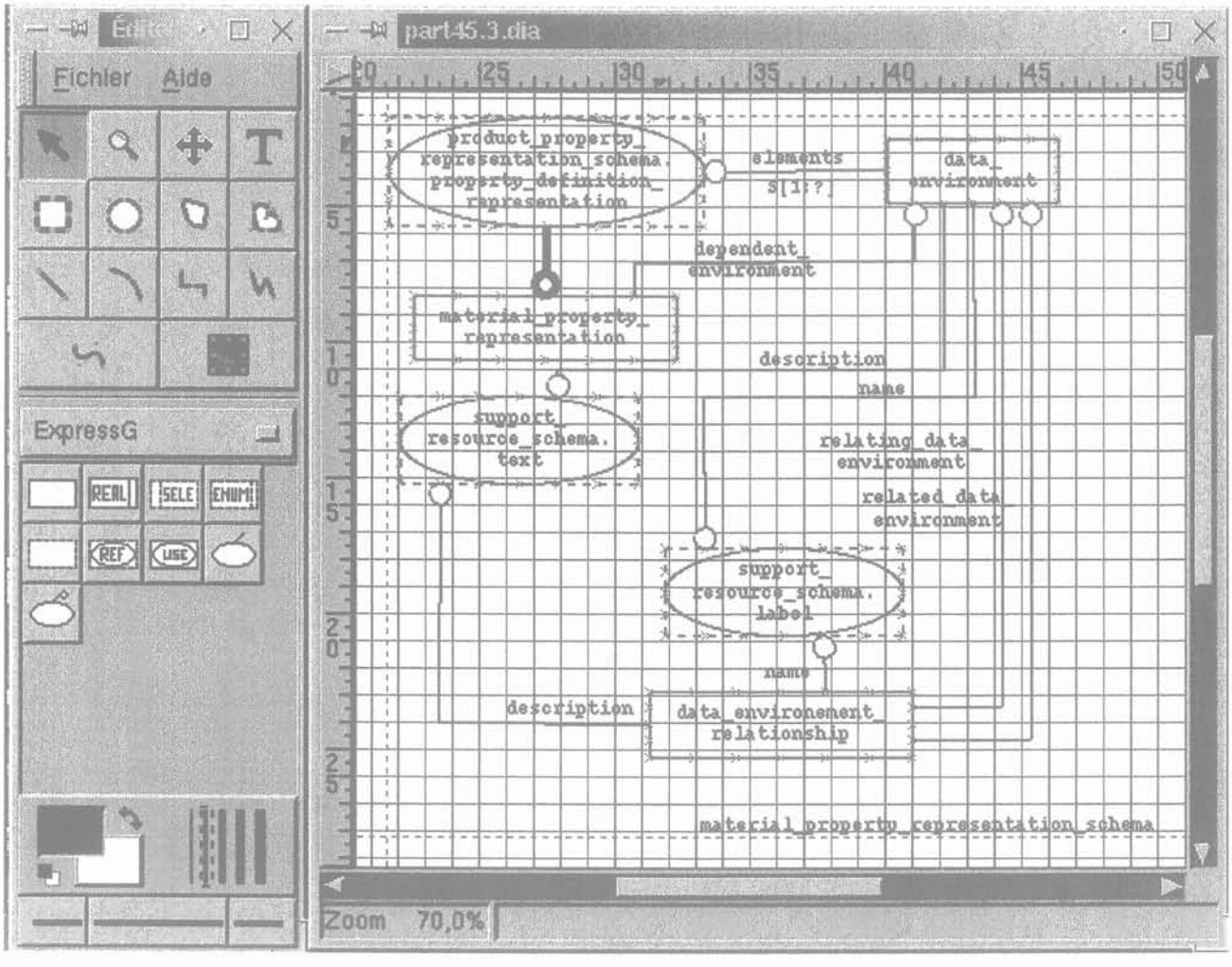


FIG. 4.11 – Un module Express-G pour Dia

4.3 Vers un environnement unique de développement

Nous avons la plupart du temps choisi des outils séparés les uns avec les autres pour développer et mettre en oeuvre un modèle STEP. Cependant à terme, il serait beaucoup plus efficace de disposer d'un seul environnement capable de contenir tous les modèles Express, Express-G, Java, C++, etc. La navigation entre ces modèles serait alors beaucoup plus efficace.

Un tel environnement serait alors tel qu'il remplisse les rôles des outils que nous avons décrit précédemment mais également qu'il prenne en charge les interactions entre les différents

modèles.

La figure 4.12 donne une idée de ce que pourrait être un tel environnement.

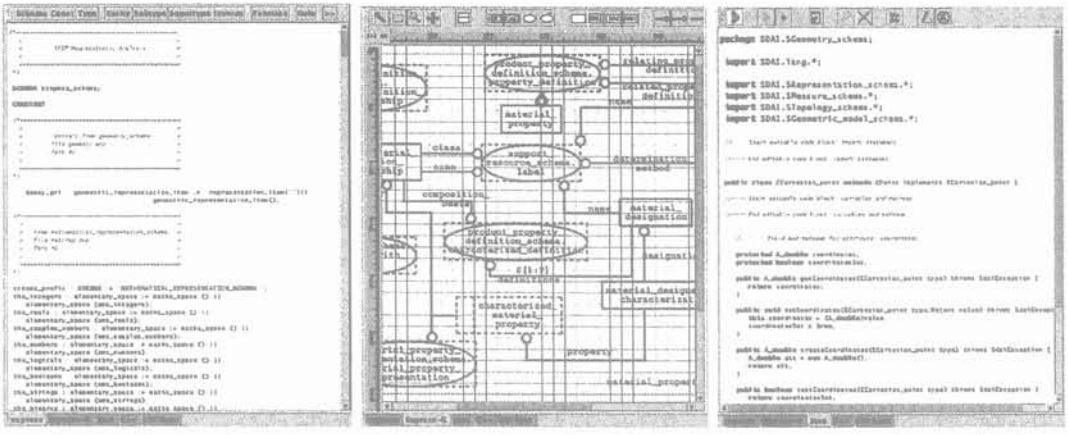
FIG. 4.12 – Un environnement pour développer un modèle Express

4.3.1 Editeurs

Il y a trois sortes d'éditeurs identifiés (Figure 4.13). Tout d'abord, il faut des éditeurs de texte formatés : un pour le langage Express, un pour chacun des langages d'implémentation (Java, C++, C, XML, etc.). Puis il faut des éditeurs graphiques (Express-G et IDEF0, éventuellement UML, SADT, etc.) ;

4.3.2 Navigation

En ce qui concerne la navigation dans un modèle et entre les modèles, nous identifions trois sortes d'arbres (voir Figure 4.14). En premier, il faut un arbre qui répertorie les instances du



(a) Express (b) Express-G (c) Source

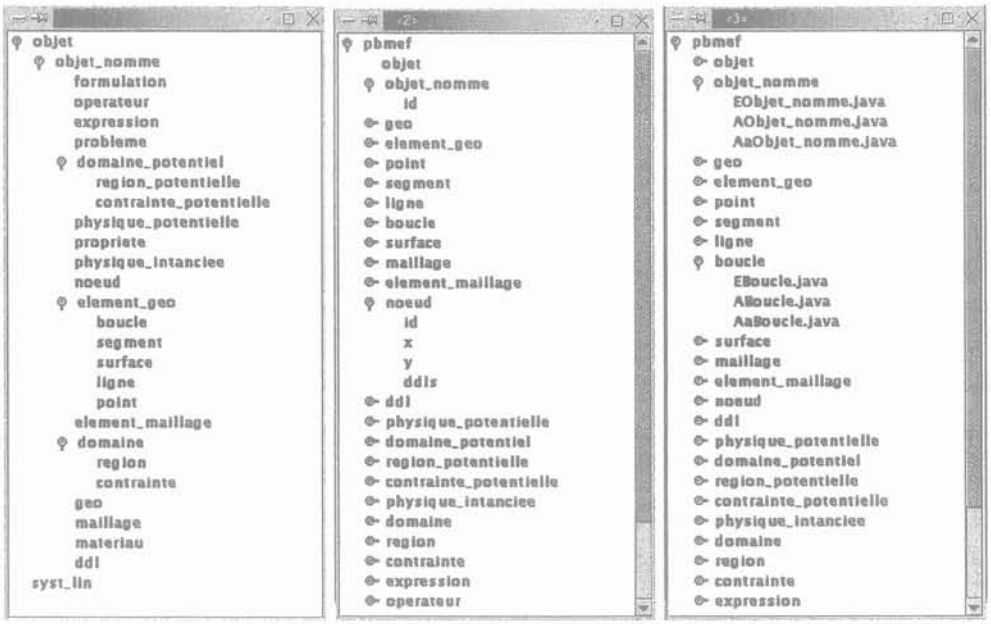
FIG. 4.13 – Editeurs

méta modèle STEP par ordre d'héritage : les noeuds de base représentent des *entités* qui ne dérivent d'aucune autre *entité* connue du modèle. Les noeuds d'ordre supérieurs sont rangés suivant leur niveau de dérivation par rapport aux noeuds de base (Figure 4.14a). Puis, il faut un arbre qui décrit la structure exacte du modèle Express. Cet arbre servira notamment à apporter des informations pour l'interface graphique de représentation des *entités* (Figure 4.14b). Enfin, il faut un arbre qui indique les fichiers générés à partir du modèle STEP pour indiquer leur provenance (Figure 4.14c).

4.3.3 Interaction entre les différents modèles

Dès qu'un modèle est modifié, les autres modèles doivent suivre. L'environnement dispose donc d'une seule base de données, qui peut être éventuellement répartie.

La base de données étant unique, les différentes applications peuvent régulièrement se mettre à jour : la modification d'une instance est répercutée automatiquement aux autres applications.



(a) Héritage (b) Structure (c) Fichiers générés

FIG. 4.14 – Arbres

5 Conclusion

L'implantation d'un *Protocole d'Application* STEP est en partie spécifiée par la norme. Cette spécification concerne en particulier les interfaces d'accès aux données sous forme générique (Express), de classes abstraites (C++) ou d'interfaces (Java). Mais l'implantation de ces interfaces en classes est laissée à la charge des développeurs d'applications. Dans ce chapitre, nous avons présenté une étude de cette mise en oeuvre en nous plaçant du côté des développeurs chargés de créer des applications pour l'utilisation logicielle de Protocoles d'Application. Nous avons développé un générateur de code capable de générer des fichiers sources en Java à partir de la description EXPRESS du *Protocole d'Application*. Ce générateur de code permettra de développer des librairies en Java à partir de modèles STEP. Nous avons également programmé des outils de manipulation de fichiers EXPRESS et EXPRESS-G ainsi que des des outils de création de fichiers neutres STEP avec une interface graphique automatiquement créée à partir du modèle EXPRESS.

Chapitre V

Application

L'environnement présenté dans le chapitre précédent permet de développer des librairies d'accès aux données STEP en respectant la méthodologie proposée par la norme. Ces librairies générées automatiquement permettent d'assurer une mise à jour simplifiée pour le suivi des évolutions des modèles de données existants et la prise en compte de nouveaux modèles de données.

Ce chapitre montre comment nous nous sommes servis de ces librairies générées pour créer une application d'analyse en magnétostatique. La maquette de l'application que nous proposons met en valeur entre autre les différents modules indispensables pour la simulation en électromagnétisme. C'est donc une maquette divisée en plusieurs modules qui représentent chacun une étapes de l'analyse en électromagnétisme.

La première section décrit la base de données destinée à recevoir et stocker les données de l'objet analysé. La seconde section décrit le gestionnaire de base de données chargé d'accéder à ces données stockées et de les transmettre au module demandeur. La section 3 décrit un visualiseur 3D qui peut accéder à la base de donnése et en montrer son image 3D. La section 4 permet de décrire la physique du problème : contraintes, conditions aux limites, matériaux. La section 5 décrit le solveur qui permet de résoudre le problème ainsi décrit. La dernière section est une conclusion sur les possibilités, les perspectives, les impossibilité de notre démarche de conception d'applications.

1 Principes

Comme nous le résumons dans la figure 5.1, un logiciel d'analyse par éléments finis en électromagnétisme peut être divisé en cinq modules : géométrie, maillage, description de la physique, résolution, exploitation des résultats.

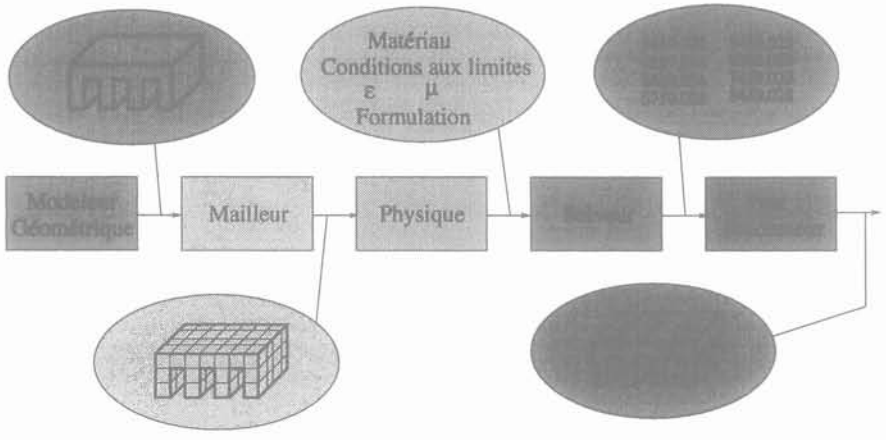


FIG. 5.1 – Modules de la simulation par éléments finis en électromagnétisme

Pour notre maquette, nous n'avons pas développé tous les modules : nous avons choisi d'utiliser autant que possible des applications existantes. La figure 5.2 résume les différents modules que nous utilisons ou développons ainsi que les formats d'échanges entre ces différents modules.

Ainsi, le module description de la géométrie est assuré par le prologiciel ProEngineer. Il permet de décrire une géométrie volumique de manière efficace. Le fichier au format STEP généré est conforme au Protocole d'Application 203. La géométrie décrite dans ce fichier est du type Boundary Representation.

Le module maillage sera assuré par le logiciel Ideas. Il y a deux intérêts à l'utilisation d'Ideas. Le premier est qu'il peut importer un fichier STEP conforme au Protocole d'Application 203. Le deuxième intérêt est que le fichier au format UNV (Ideas Universal File) est assez facilement exploitable.

Le module description de la physique est complètement développé en Java. Il est capable de lire des fichiers géométriques STEP, du maillage généré par Ideas, et d'ajouter à toutes ces données le contenu de la physique du problème, à savoir : répartir des régions, des matériaux, des contraintes.

Le module solveur est également développé en Java. Il permet la résolution du problème et la sortie des résultats en format STEP également.

Le module exploitation des résultats est également développé entièrement en Java avec utilisation de la librairie Java3D.

En plus de ces modules classiques, nous ajoutons la base de données, le gestionnaire de base de données, différents viewers de fichiers neutres STEP (géométrie, maillage)

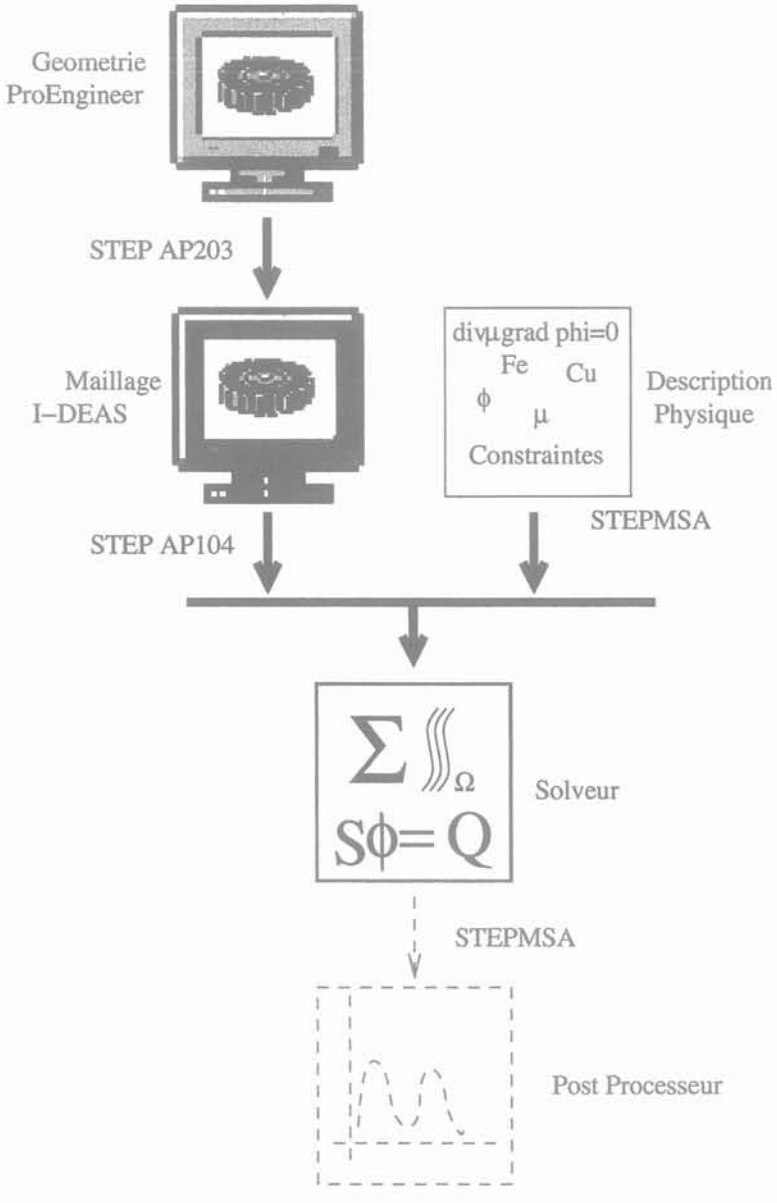


FIG. 5.2 – Modules et formats d'échanges

2 Base de Données

La base de données est destinée à recevoir les données que l'on veut garder et auxquelles on veut avoir accès rapidement.

La génération et la lecture de ces fichiers est assurée par les logiciels extérieurs si besoin est (modeleurs géométriques uniquement pour le moment) et par le gestionnaire de base de données.

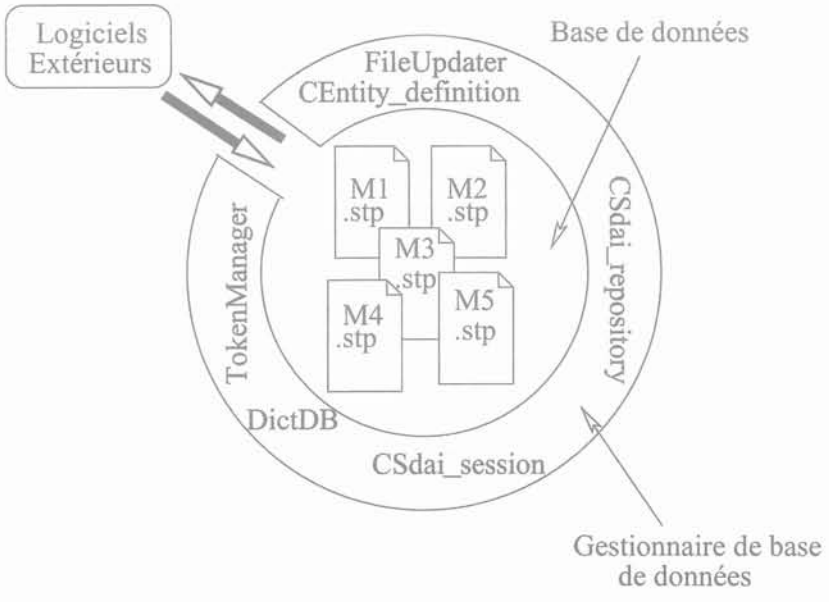


FIG. 5.3 – Base de données et Gestionnaire de base de données

3 Gestionnaire de base de données

Le gestionnaire de base de données est l'interface entre la base de données et les autres modules. Il doit être capable de lire et écrire les fichiers au format neutre STEP et de fournir les informations requises par tous les autres modules. Il dispose donc de nombreuses fonctions de stockage, de recherche et de tri.

Les classes qui le composent sont (voir Figure 5.3) :
- les classes qui implémentent la méta-représentation d'Express : CEntity_definition, CType_definition, CInteger_type, ...
- les classes qui implémentent le schéma sdai_session de la *partie* 22 de STEP (voir Section 1.1 du Chapitre 4) : CSdai_session, CSdai_repository, ...
- les classes de lecture/écriture/mise à jour de fichiers EXPRESS et de fichiers neutres : parseurs générés par JavaCC (ExpIdentifier, TokenManager,...), FileUpdater)
- les classes de stockage de la méta représentation : DictDB,...

Les modules internes (viewers, mailleurs, solveurs) ne peuvent accéder à la base de données que par l'intermédiaire d'une de ces classes. Par contre les modules externes (modeleurs géométriques, ici ProEngineer et Ideas) y ont accès directement par lecture/écriture contrôlées par l'utilisateur,

4 Viewers

Une certitude qu'admet STEP est que tous les produits ont une représentation géométrique. Les premiers outils développés étaient donc des visualiseurs 2D ou 3D pour l'implantation de l'AP 203. Un visualiseur doit être capable de lire un fichier neutre STEP et d'en représenter graphiquement les éléments.

Le visualiseur que nous avons développé utilise la librairie Java3D. Le principe en est simple : on développe une "package" java pour faire l'interface entre les entités géométriques de STEP (cartesian_point, surface, ...) et les classes de la librairie Java3D (Point, ...). Ce package contient des classes d'accès aux fichiers neutres et des classes contenant les données de STEP.

La transformation des données STEP en instances de classes Java3d se fait après lecture du fichier neutre STEP par la base de données et demande par le vieweur à travers le gestionnaire de la base de données.

Nous utilisons l'approche objet pour transformer les données STEP lues en instances de classe Java. Chaque objet instancié par la lecture du fichier neutre STEP est capable de donner sa propre traduction en données lisibles par Java3d à travers une méthode qui dépend de la classe dont l'instance est issue.

Par exemple, une instance de la classe Brep_solid_... permet d'avoir un tableau des données de ses composants grâce à la méthode getData(). Cette méthode fait une descente récursive dans les éléments de cette instance et de toutes celles qu'elle référence, jusqu'à la source des données qui donne finalement les informations fondamentales, notamment les coordonnées, rayons, etc.

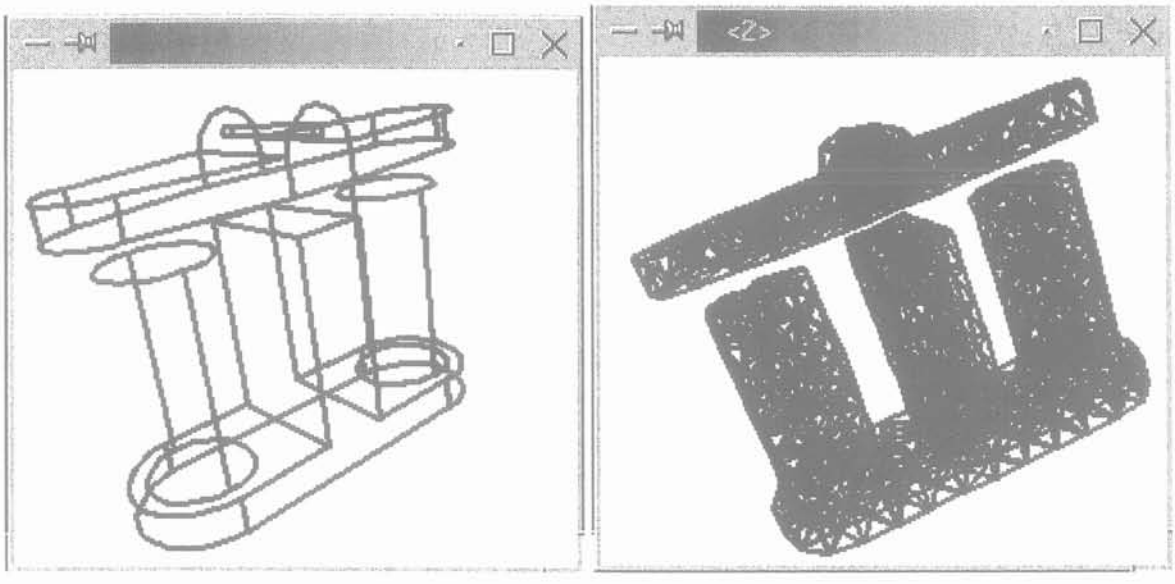
On utilise ainsi l'héritage/dérivation, le polymorphisme, et l'encapsulation des données.

En plus du viewer géométrique, nous avons développé un viewer de maillage qui permet de visualiser les principales structures de la *partie* éléments finis de STEP (*partie* 104). Les instances visualisées (éléments, noeuds, groupes d'éléments et de noeuds) peuvent être le résultat de l'importation d'un autre format de fichier converti en instances de structures STEP. Pour notre application, le maillage provient du logiciel Ideas par l'intermédiaire d'un fichier universel (.unv) et converti en "instances STEP" par une classe Java (que nous avons appelée IdeasImporter).

La Figure 5.4 montre la visualisation "wireframe" d'une géométrie créée par ProEngineer et la visualisation du maillage STEP obtenu par l'importation du fichier universel Ideas correspondant et la traduction vers des structures STEP.

5 Descripteur de physique

La description géométrique n'est pas suffisante pour faire une analyse en électromagnétisme. Il est nécessaire également de connaître la physique du problème, c'est à dire non seulement la



(a) Géométrie (b) Maillage

FIG. 5.4 – Visualiseurs

composition des matériaux et donc leur propriétés magnétiques mais également les conditions extérieures : est-ce qu'il y a des sources de champs magnétiques/électriques, est-ce qu'on tient compte des courants de pertes, y a-t-il des sources de chaleur, des symétries qui permettraient de simplifier le problème et donc de le résoudre plus rapidement...

Le descripteur de physique est entièrement développé en Java. Il accède à la base de données par l'intermédiaire du gestionnaire, en extrait la géométrie, et permet à l'utilisateur d'affecter des propriétés physiques telles que les matériaux, les contraintes du problème, les sources de champ. C'est ce module qui nous permettra de vérifier si notre modèle EXPRESS pour la magnétostatique est complet, c'est à dire si les données que nous avons structurées sont nécessaires et suffisantes pour définir complètement le problème physique..

6 Solveur

Le solveur contient les méthodes numériques de résolution des équations de la physique. Dans notre cas, le solveur est un solveur par éléments finis.

Il prend la description géométrique et physique puis les noeuds et éléments finis et procède à l'intégration et à la résolution des systèmes linaires obtenus.

Le résultat est ensuite stocké dans des fichiers au format neutre STEP par l'intermédiaire du gestionnaire de base de données.

7 Exploitation des résultats

Pour le moment, nous n'avons développé que la visualisation sous forme de dégradé des résultats obtenus par le solveur. C'est une forme qui convient bien dans le cas des résolutions utilisant la formulation potentiel scalaire magnétique linéaire.

8 Problèmes lors du développement

En général, pour créer une application utilisant un modèle de données STEP, le problème est de bien comprendre la signification des modèles. Par exemple, le modèle géométrique est très complet : il décrit à la fois les modèles CSG (Constructive Solid Geometry), BRep (Boundary Representation), extrudé en 1D, 2D et 3D (voir la *partie* géométrie de STEP [40] pour les détails). Or pour le moment, les logiciels de calcul utilisent pour la majorité le modèle BRep. En suivant notre méthode de génération de classes, nous avons la structure en Java des modèles CSG et sommes capable d'instancier, par exemple, une intersection de deux ellipsoïdes mais il faut en plus pouvoir coder le calcul des points de cette intersection pour ensuite les visualiser graphiquement. La difficulté et peut-être également l'intérêt de STEP vient du fait que, même en connaissant la structure de données d'un produit, il faut des compétences supplémentaires pour coder les méthodes numériques.

9 Tests

9.1 Protocole d'Application AP203 : Config and Control Design

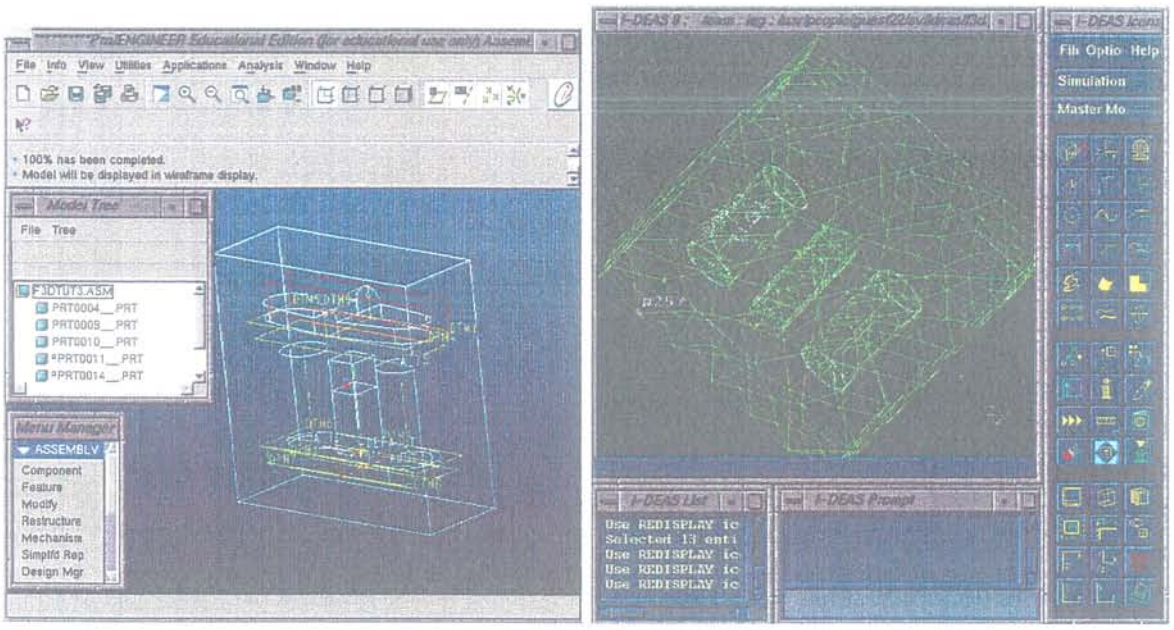
Nous avons testé le générateur de code sur l'AP203 pour pouvoir importer et visualiser des fichiers STEP en géométrie provenant de ProEngineer (Figure 5.5a) et Ideas (Figure 5.5b). La librairie de classes Java générée complétée par des méthodes numériques du type discrétisation des B_spline et associée à la librairie Java3D a donné le vieweur présenté dans la section 4.

9.2 Cube

Pour tester le solveur sur une géométrie simple, nous avons fait des essais sur un cube. La géométrie est dessinée sous ProEngineer, le maillage est fait par Ideas par importation de la géométrie STEP et le descripteur de physique et le solveur sont ceux décrits dans les sections précédentes.

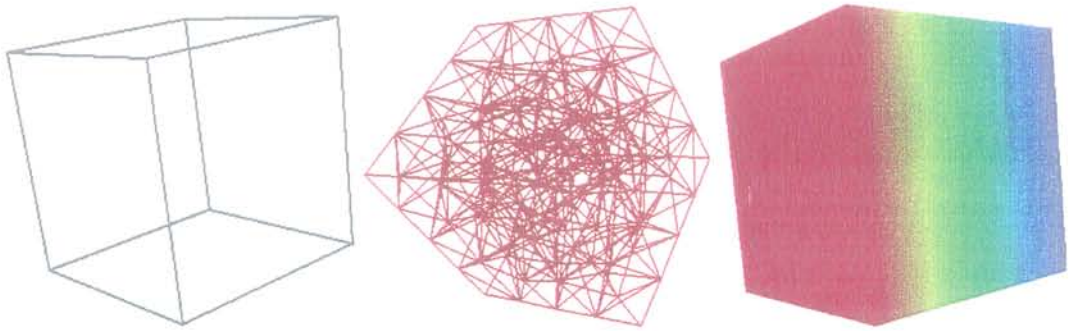
Le problème est une portion cubique de l'espace dont la face supérieure est portée au potentiel magnétique 0 et la face inférieure au potentiel magnétique 100.

On obtient les résultats montrés sur la Figure 5.6



(a) ProEngineer (b) Ideas

FIG. 5.5 – Les prologiciels ProEngineer et Ideas



(a) Géométrie (b) Maillage (c) Résultat

FIG. 5.6 – Cube soumis à une différence de potentiel magnétique

10 Conclusion

Ce chapitre a présenté une application du modèle que nous avons développé. Cette application est destinée à faire des calculs en magnétostatique à partir d'une géométrie STEP et d'un maillage Ideas. L'intérêt d'une telle application est d'utiliser des applications existantes. Des modeleurs tels que les prologiciels ProEngineer et Ideas sont les meilleurs pour traiter des géométries complexes et permettent de faire des calculs d'intersection de volumes, de surfaces, en 2D ou 3D.

Puis il s'agit de développer les interfaces quand elles n'existent pas déjà. Le développement d'interfaces est moins coûteux en temps de programmation que le développement des applications. Le seul risque de cette méthode est la perte d'information. Mais si le format neutre utilisé permet du côté du récepteur de retrouver totalement la quantité d'information émise, ce système est avantageux. L'utilisation du format STEP entre ProEngineer et Ideas a permis de retrouver sans erreur notable la géométrie échangée pour un cas assez complexe (contacteur défini avec des splines, des cylindres, des extrusions,...) et paraît donc fiable.

Les modules telles que le descripteur de physique et le solveur ont été entièrement développées mais utilisables uniquement dans des cas simples de Magnétostatique. Pour la suite de ce travail, la mise en oeuvre de méthodes numériques plus puissantes (traitement matrices creuses, optimisation des contraintes,...) permettrait, sans modifier la méthode ni les structures existantes, de traiter des cas plus complexes.

Conclusion Générale

Les échanges de données normalisés entre systèmes de Conception Assistée par Ordinateur existent depuis plus d'une vingtaine d'année pour des données de type géométrie. Mais pour d'autres types de données, dont les résultats numériques, ils ne sont encore que des standards de fait, c'est à dire imposés par un système ou une entreprise dominante. C'est le cas par exemple format Universel Ideas pour les données de type maillage.

Depuis assez peu de temps (pour l'échelle de l'Organisation Internationale de Normalisation, c'est environ 10 ans) il existe une norme encore en développement qui a l'ambition de permettre des échanges de données normalisés pour des données d'autres types dont bien sûr la géométrie mais aussi la configuration, la représentation d'un produit, l'analyse en cinétique, etc... C'est la norme ISO 10303, connue sous le nom de STEP ou Standard for The Exchange of Product model data.

Cette norme est à la fois un langage, une méthodologie, une base de connaissance pour modéliser et échanger des données d'un produit tout au long de son cycle de vie. C'est une entreprise très ambitieuse qui se veut applicable à tous les métiers par lesquels passe un produit depuis sa conception jusqu'à sa réalisation et son service après-vente.

Ce travail s'inscrit dans le domaine de la modélisation et des échanges de données pour la simulation en électromagnétisme.

Après avoir constaté que dans notre domaine, la simulation en électromagnétisme, il existe des besoins d'échanger de données et donc de les modéliser, nous avons proposé la solution STEP qui est actuellement la plus prometteuse puisque la plus récente et celle annoncée par les plus grandes industries comme celle qu'ils adopteront (la NASA a annoncé récemment qu'elle adoptait STEP comme le format privilégié pour les échanges avec ses collaborateurs).

Puis nous avons travaillé sur une expérience à la fois théorique et pratique de la modélisation de données en suivant la méthodologie STEP à travers la description d'un modèle de données pour la magnétostatique et à travers la mise en oeuvre informatique de ce modèle.

Ce mémoire contient donc une étude des différentes méthodes de modélisation de données essentiellement pour éclaircir quelques termes et noms de standards parmi les plus récents, une description de la norme STEP et des méthodes qu'elle utilise, une proposition de modèle de données STEP pour la simulation en magnétostatique et la description des réalisations informatiques que nous avons effectuées.

L'un des principaux buts de ce travail a été de couvrir l'existant de la méthodologie STEP, ses méthodes d'implantation informatique, sa structure, ses développements futurs. Il s'agissait de la comprendre de manière approfondie et donc pratique pour être capable d'intégrer un projet en marche depuis plus de 10 ans.

Un des aspects essentiels de notre recherche a donc été de faire une expérience pratique de cette méthode de modélisation que nous voulons intégrer dans le domaine de la simulation en électromagnétisme. D'où la nécessité de couvrir des domaines qui nous sont peu familiers tels que la génération de code.

Cette expérience nous a permis de mesurer la réelle ampleur de la norme STEP. C'est un projet d'une ambition sans égale puisqu'il vise à couvrir tous les aspects sur les échanges de données, toutes les industries et tous les domaines. Nous avons testé la complexité de sa mise en oeuvre à travers des réalisations informatiques.

Nous avons constaté de manière pragmatique que l'entrée d'un nouveau domaine dans cet projet n'est pas gratuite et nécessite beaucoup de temps d'apprentissage et de collaboration avec tous les acteurs du nouveau domaine mais aussi avec des spécialistes de la modélisation.

A travers ce travail, nous avons voulu couvrir de manière nécessairement superficielle tous les aspects de la norme ISO 10303 pour son intégration dans la communauté du calcul de champs électromagnétiques et pour l'intégration de l'analyse en électromagnétisme dans ce projet.

Il nous paraît indiscutable qu'il reste encore beaucoup de travaux à faire de la part de la communauté du calcul de l'électromagnétisme dans la formalisation des données nécessaires à notre domaine mais également, de la part de STEP pour faciliter l'intégration de personnes expertes dans leur métier mais encore novices dans la modélisation de données produits.

Perspectives

Pour la suite de ce travail, il est nécessaire d'approfondir trois points très différents.

Le premier est de manière évidente l'extension de notre formalisation du modèle de donnée en magnétostatique en un modèle de données pour tout le domaine de l'électromagnétisme. L'électromagnétisme est un domaine également très complexe mais dont les lois (les équations de Maxwell) sont parmi les mieux connues de la physique. Les principales difficultés de sa formalisation sera peut-être pour l'aspect des interactions avec d'autres domaines (couplages).

Un autre point à approfondir est l'aspect réalisation logicielle. L'utilisation de méthodes efficaces pour la résolution de problèmes complexes est courante dans tous les logiciels de calcul en électromagnétisme. Nous pensons que l'intégration de ces méthodes nécessitera beaucoup de programmation mais ne présentera que peu de difficultés structurelles. Par contre cette intégration de méthodes numériques performantes peut également poser un problème de choix entre la rapidité de calcul et la clarté de la structure de l'application.

Le dernier point qu'il faudrait aborder est le lancement d'une proposition d'un nouveau Protocole d'Application dans la norme ISO 10303. Cela ne peut se faire qu'avec l'appui des industries de l'électrotechnique (fabricants de machines électriques, sociétés de développement de logiciels de calcul de champs électromagnétique,...).

Liste des Publications

Ma S., Maréchal Y., Coulomb J-L.,
"Evaluation of the STEP Standard in modeling electromagnetic phenomena", proceedings of the IGTE'98 conference, 1998.

Ma S., Maréchal Y., Coulomb J-L.,
"STEP standard's evaluation for modeling in electromagnetism", The International Journal for Computation and Mathematics in Electrical and Electronic Engineering, Vol. 18, No. 3, 1999)

Ma S., Maréchal Y., Coulomb J-L.,
"Finite Element Analysis in the STEP Standard", proceedings of the CEFC2K conference (2000)

Ma S., Maréchal Y., Coulomb J-L.,
"Methodology for an implémentation of the STEP Standard : a Java Prototype", published in the International Journal "Advances in Engineering Software", Vol. 32, pp.15-19, 2001

Ma S., Maréchal Y., Coulomb J-L.,
"A Finite Element 3D Magnetostatic Solver using STEP data", proceedings of the Compumag'2001 conference (2001)

Bibliographie

[1] Anderl R, Wasmer A, "Integration of Product Life Cycle Views on the Basis of a Shared Conceptual Information Model" Chap 2 pp 47-58, Information Infrastructure Systems for Manufacturing. Chapmann & Hall, London 1997

[2] Arbouy, S. "STEP concepts fondamentaux", *AFNOR*, Paris, 1994.

[3] Grady Booch, "Object-oriented Design with Applications", *The Benjamin/Cummings Publishing Company*, (1991)

[4] Med Bouazza, le langage EXPRESS, Hermes, Paris (1995)

[5] Brun J., "Modèle produit : les conditions de cohérence et leur évolution au cours des processus de conception" Revue internationale de CFAO et d'Infographie, vol.12 no 5 pp 513-529, 1997

[6] Jean-Louis Coulomb, "Analyse tridimensionnelle des champs électriques et magnétiques par la méthode des éléments finis", *Thèse de doctorat, Institut National Polytechnique de Grenoble*, 1981

[7] Dhatt, G. and Touzot, G., "Une présentation de la méthode des éléments finis", Maloine, Collection Université de Compiègne, 1984

[8] Dular P., Legros W., Nicolet A., "Coupling of Local and Global Quantities in Various Finite Element Formulations and its Application to Electrostatics, Magnetostatics and Magnetodynamics", IEEE Transaction on Magnetics, Vol. 34, No. 5, Septembre 1998

[9] Hubert Gié, "Electromagnétisme", Sciences physiques, Tec&Doc, 1991

[10] Ghodous P., "Modélisation Intégrée de Données de Produit et de Processus de Conception", *Thèse de doctorat, Université Claude Bernard-Lyon 1*, 1996

[11] Ghodous P., Vandorpe D., "A Systematic Approach for Product and Process Data Modeling Based on the STEP Standard", Computer-Aided Civil and Infrastructure Engineering, Vol. 13, pp. 189-205, 1998

[12] Henrotte F., Meys B., Hedia H., Dular P., Legros W., "Finite Element Modelling with Transformation Techniques", IEEE Transaction on Magnetics, Vol. 35, No. 3, Mai 1999

[13] Christophe Hérault, "Vers une simulation sans maillage des phénomènes électromagnétique", *Thèse de doctorat, Institut National Polytechnique de Grenoble*, 2000

[14] Ivar Jacobson, "Génie Logiciel Orienté Objet", *Addison-Westley France*, (1993)

[15] Laurent Krähenbühl, "La méthode des équations intégrales de frontière pour la résolution des problèmes de potentiel en électrotechnique et sa formulation axisymétrique", *Thèse de doctorat, Ecole Centrale de Lyon*, 1983

[16] Huu Tuan Luong "Amélioration de la formulation en potentiel scalaire magnétique et généralisation au couplage entre équations de champ et de circuit électrique", *Thèse de doctorat, Institut National Polytechnique de Grenoble*, 1997

[17] Michel Lai, "UML, La notation unifiée de modélisation objet", *Dunod*, (1998)

[18] Liang J., Shah J.J, D'Souza R., Urban S.D, Ayyaswamy K., Harter E., Bluhm T. "Synthesis of consolidated data schema for engineering analysis from multiple STEP Application Protocol", Computer-Aided Design, Vol. 31, pp. 429-447, 1999

[19] Männistö T., Peltonen H., Martio A., Sulonen R., "Modelling generic product structures in STEP", Computer-Aided Design, Vol. 30, No. 14, pp. 1111-1118, 1998

[20] Maréchal Y., "Modélisation des Phénomènes Magnétostatiques avec Terme de Transport. Application aux Ralentisseurs Electromagnétiques", *Thèse de doctorat, Institut National Polytechnique de Grenoble*, 1991

[21] Hafedh Mili, François Pachet, "Exchanging Models Between Tools Supporting A Different Number of Modeling Layers", *OOPSLA'98*, (1998)

[22] Nayroles B., Touzot G., Villon P., "Generalizing the Finite Element Method : Diffuse Approximation and Diffuse Elements", *Computational Mechanics, 10, pp 307-318*, 1992.

[23] James Rumbaugh, "Modélisation et Conception orientées objet", *Mason et Prentice Hall*, (1995)

[24] D. Schenk & P. Wilson *Information Modeling the EXPRESS Way,* Oxford University Press, ISBN 0-19-508714-3 1994

[25] Zienkiewicz O.C, "The Finite Element Method", *Mc Graw hill*, New York, 1989

[26] http ://www.sgi.com/developers/devtools/languages/rapidapp.html

[27] http ://www.nist.gov/sc4/tools/nist.

[28] http ://www.epmtech.jotne.com/

[29] http ://www.steptools.com/products/stdev/

[30] http ://www.bauv.unibw-muenchen.de/WinSTEP/

[31] http ://www.lksoft.com/

[32] http ://www.mel.nist.gov/sc5/soap/

[33] http ://www.metamata.com/javacc/index.html

[34] ISO 10303, *Industrial automation systems and integration Product data representation and exchange -*
"Part 11 : The EXPRESS language reference manual" 1996

[35] ISO 10303, *Industrial automation systems and integration Product data representation and exchange -*
"Part 21 : Clear-Text Encoding Exchange" 1996

[36] ISO 10303, *Industrial automation systems and integration Product data representation and exchange -*
"Part 22 : Standard Data Access Interface" 1996

[37] ISO 10303, *Industrial automation systems and integration Product data representation and exchange -*
"Part 25 : EXPRESS to OMG XMI binding" 1999

[38] ISO 10303, *Industrial automation systems and integration Product data representation and exchange -*
"Part 28 : XML representation of EXPRESS" 1999

[39] ISO 10303, *Industrial automation systems and integration Product data representation and exchange -*
"Part 41 : Fundamentals of product description and support" 1999

[40] ISO 10303, *Industrial automation systems and integration Product data representation and exchange -*
"Part 42 : Geometric and topological representation" 1996

[41] ISO 10303, *Industrial automation systems and integration Product data representation and exchange -*
"Part 45 : Materials" 1996

[42] ISO 10303, *Industrial automation systems and integration Product data representation and exchange -*
"Part 27 : Java Language Binding" 1999

[43] Hunten, K. A. *Industrial Automation Systems and Integration - Product Data Representation and Exchange - Part 104 : Integrated application resources : Finite element analysis*, Committee Draft of ISO 10303-104 (document ISO/TC184/SC4/WG3/N447). 12 September 1995

[44] Flux2D, Flux3D sont des logiciels édités par la société CEDRAT.

[45] ProEngineer est un logiciel de MathWorks

[46] IDEAS est un logiciel de SDRC

Annexe A

Glossaire

Analyseur syntaxique Pour un compilateur, application chargée de rassembler les unités lexicales pour en faire des productions. Souvent l'analyseur syntaxique est reliée à la sortie d'un scanner ou Analyseur lexical.

Analyseur lexical Voir Scanner.

Analyseur sémantique Pour un compilateur, application chargée de vérifier la cohérence et le typage des productions. Souvent l'analyseur sémantique est associé à un analyseur syntaxique..

Attribut Membre d'une classe, d'une structure. Par exemple, "coordinates" est un attribut de la structure EXPRESS "point" définie par :

```
ENTITY point;
coordinates : SET[2 :3] OF REAL;
END_ENTITY;
```

Base de Données Ensemble de fichiers contenant des données. Elle peut être également un ensemble d'instances contenant des données qu'il faut sauvegarder.

BRep Boundary Représentation. Manière de représenter une géométrie à partir de ses frontières : un solide est un ensemble de faces qui sont des ensembles de courbes qui sont des ensembles de points.

Byte Code ou Bytecode Le format compilé pour des programmes Java. Une fois qu'un programme Java a été converti en bytecode, il peut être transféré àă travers un réseau et exécuté par la machine virtuelle Java Virtual (VM). Les fichiers Bytecode ont généralement une extension .class et sont les mêmes d'une plateforme à une autre..

CAO Conception Assistée par Ordinateur. Ensemble des outils informatiques permettant de concevoir un produit (dessin, analyse,...).

Classe Pour la norme STEP, ensemble de *parties*. C'est le plus haut niveau de découpage effectué dans la norme : STEP est constituée de 7 *classes* contenant chacune une ou plusieurs *parties*. Par exemple, la *classe* ***ressources intégrées*** contient la *partie* 42 : *geometric and topological representation*. Attention à ne pas confondre avec la notion de classe de langages informatiques comme C++ et Java : une classe pour C++ et Java est la description d'un objet : *class* point.

Compilateur Application permettant de transformer un langage dit de haut niveau en langage machine.

Contrainte Dans la méthode des éléments finis, relation imposée à une variable d'état pour tenir compte des effets d'interaction du domaine de l'espace étudié avec le reste de l'espace.

CSG Constructive Solid Geometry. Manière de décrire une géométrie à partir d'éléments de base (pavé, sphère, cylindre, cone) et d'opérations booléennes entre ces éléments.

Dérivation Spécialisation d'un objet. Exemple : soit l'objet *véhicule*. L'objet *voiture* dérive de l'objet *véhicule* car la *voiture* est un *véhicule* particulier (qui a quatre roues).

Donnée de définition La donnée de définition d'un produit est commune à tous les produits qui ont une même dénomination. La puissance est une donnée de définition d'une machine, un numéro de série n'est pas une donnée de définition : deux machines peuvent avoir la même puissance mais pas le même numéro de série. STEP ne normalise que les données de définition d'un produit.

EBNF Extended Backus-Naur Formalism. Langage de description d'unités lexicales et de productions. Exemple :

```
[1] mot : := consonne voyelle+ consonne
[2] consonne : := [^aeiouy]
[3] voyelle : := [aeiouy]
```

Entité Pour la norme STEP, objet appartenant à un *schéma* et contenant un ensemble d'*attributs* et de *règles*. C'est le seul objet instantiable d'un modèle EXPRESS. L'*entité* est désignée par le mot clé ENTITY dans le langage EXPRESS. Exemple : ENTITY point.

EXPRESS Langage de description orienté objet normalisé dans STEP.

Fichier neutre Fichier contenant des données pour les échanges. Exemple : si on définit dans STEP l'entité :

```
ENTITY point;
    x : REAL;
    y : REAL;
END_ENTITY;
```

alors un fichier neutre qui veut échanger deux points aura la forme :

```
#1 point(0,0)
#2 point(1,1)
```

Formulation Réécriture d'un ensemble d'équations dans le but de simplifier la résolution.

Générique Qui est à la base de tout.

Héritage Processus par lequel un objet s'approprie la définition et les propriétés d'un autre objet. Exemple : soit un objet appelé *rectangle* défini par deux dimensions *largeur* et *longueur*. Nous pouvons alors définir un *parallépipède* comme étant un *rectangle* auquel est ajoutée une troisième dimension, *hauteur*. Le *parallépipède* a obtenu par héritage ses attributs *largeur* et *longueur*. Lors de l'instanciation d'un *parallépipède*, les valeurs des trois attributs *largeur*, *longueur* et *hauteur* devront être spécifiés.

Implantation Mise en oeuvre d'un modèle, d'une classe.

Instance Objet. Les *points* de coordonnées (0, 1, 1) et (0, 0, 0) sont des instances de *point*.

Langage orienté objet Langage qui met en oeuvre des objets. C++, Java, Smalltalk, EXPRESS sont orientés objets. C, Pascal, Fortran ne sont pas orientés objets.

Maillage Division de l'espace en sous-domaines.

Meta langage langage utilisé pour décrire un langage. Exemple : EXPRESS est un méta langage car il existe des *schémas* écrits en EXPRESS pour la description du langage IDL (Interface Description Language), du langage C++ et C.

Méthode des Eléments Finis Méthode qui consiste à diviser l'espace continu en sous-domaines appelés éléments finis sur lesquelles les propriétés physiques sont simplifiées.

Modèle de propriété Pour la modélisation d'un matériau, courbe mathématique ou programme informatique ou tables de données permettant de représenter la propriété d'un matériau. Par exemple, une permeabilité peut être modélisée par une courbe de saturation en B(H).

Modélisation orientée objet Modélisation utilisant des objets.

Modeleur Géométrique Application permettant de créer une géométrie. ProEngineer, Ideas sont des modeleurs géométriques.

Objet Structure contenant un ensemble de données appelées attributs ou champs (suivant les langages) et un ensemble de méthodes pour manipuler ces champs. Exemple en Java : soit une description d'objet *personne*

```
class personne{
    public String nom;
    public int age;
    public void feterAnniversaire()
    {
    age=age+1;
    }
}
```

Les champs (ou attributs) de l'objet Pierre seront `nom` et `age` et pourront être affectés d'une valeur :

```
personne Pierre;
Pierre.nom="Pierre";
Pierre.age=2;
```

L'objet Pierre possède une méthode : `feterAnniversaire()`. Lors de l'appel à la méthode `feterAnniversaire()` de l'objet Pierre, on va incrémenter le champ `age` de 1. Ce champ prendra alors la valeur 3.

Package ou Paquetage Pour Java, c'est un ensemble de classes. Un package permet d'affiner la visibilité d'une classe, d'une variable ou d'une méthode par rapport aux autres. Par exemple, une

Partie Pour la norme STEP, ensemble de *schémas*. Par exemple la *partie* 22 de STEP contient les *schémas* ***dictionary_schema, parameter_data_schema, session_schema, population_schema***.

Produit Pour la norme STEP, "*un produit est tout ce qui est descriptible par STEP de telle sorte que cette description puisse se transformer en une représentation interprétable par l'homme et réciproquement que cette dernière puisse se transformer en une représentation interprétable par la machine sans perdre en intégrité*". (réf. [2])

Production Pour un compilateur, ensemble d'unités lexicales ordonnées suivant des règles spécifiques à chaque langage. Par exemple, pour le langage français courant, <sujet><verbe><compl est une production. Pour le langage Express "ENTITY"<label>";" est une production également.

Scanner Pour un compilateur, application chargée de reconnaitre et de séparer des unités lexicales.

Schéma Pour la norme STEP, ensemble d'*entités*, *types*, *règles*. Par exemple, on peut définir le *schéma* suivant :

```
SCHEMA labElec_grenoble;
TYPE equipe = ENUMERATION OF
    (Commande des systèmes,
    Electronique de puissance,
    Réseaux électriques
    Matériaux magnétiques
    Machines électriques
    Modélisation et CAO
    Conception et diagnostics intégrés);
END_TYPE;
ENTITY permanent;
    nom : STRING;
    equipe_de_recherche : SET [1 :?] OF equipe;
    fonction : STRING;
END_ENTITY;
RULE un_seul_directeur_de_laboratoire FOR (permanent);
WHERE
    SIZEOF(QUERY(x<* permanent | fonction='directeur de laboratoire')=1);
END_RULE;
END_SCHEMA;
```

Sémantique Pour un langage : signification d'un mot, d'une phrase.

Token Voir Unité lexicale.

Unité lexicale Pour les compilateurs, ensemble des caractères compris entre deux séparateurs. La plupart du temps, les séparateurs sont des espaces, retour à la ligne ou des tabulation. Mais ce n'est pas toujours le cas : parfois, on utilise les caractères "." ou "," etc.

Annexe B

Step On A Page

La page suivante montre le document Step On A Page fourni par [32].

Step On A Page est un résumé graphique décrivant la progression des différentes parties de STEP. L'état d'avancement de chaque partie est indiqué par une lettre. La signification de chaque lettre est donnée en bas à gauche de la page.

- "O" est le stade préliminaire
- "A" est le stade proposition
- "W" est le stade préparatoire
- "C" est le stade d'étude par un comité
- "E" est le stade d'enquête publique
- "F" est le stade d'approbation
- "I" est le stade de publication
- "X" signifie que la partie a été retirée

On peut considérer qu'une partie peut-être implémentée sans trop de risques de modification dès que la partie a atteint les niveaux E, F et I.

ISO TC184 SC4

STEP on a Page

ISO 10303

APPLICATION PROTOCOLS AND ASSOCIATED ABSTRACT-TEST SUITES

I 201 Explicit draughting [ATS 301 = X]
I 202 Associative draughting [C]
I 203 Configuration-controlled design (c2=I,a1=I)[C]
E 204 Mechanical design using boundary rep [C]
X 205 Mechanical design using surface rep [X]
X 206 Mechanical design using wireframe [X]
I 207 Sheet metal die planning and design [I]
X 208 Life-cycle product change process [X]
E 209 Composite & metal structural anal & related design[X]
I 210 Electronic assy, interconnection & packaging design [X]
X 211 Electronic P-C assy: test, diag, & remanuf[X]
I 212 Electrotechnical design and installation [C]
E 213 Num control (NC) process plans for mach'd parts [X]
F 214 Core data for automotive mech design processes [F]
W 215 Ship arrangement [X]
C 216 Ship moulded forms [X]
W 217 Ship piping [X]
W 218 Ship structures [W]
X 219 Dimension inspection [X]
O 220 Proc. plg, mfg, assy of layered electrical products [X]
C 221 Functional data & their schem rep for process plant [X]
X 222 Design-manuf for composite structures [X]
W 223 Exch of design & mfg product info for cast parts [X]
I 224 Mech parts def for p. plg using mach'n'g feat(e2=F) [I,W]
I 225 Building elements using explicit shape rep [C]
W 226 Ship mechanical systems [X]
@ 227 Plant spatial configuration(e2=W) [X]
X 228 Building services: HVAC [X]
X 229 Design & mfg product info for forged parts[X]
X 230 Building structural frame: steelwork [X]
C 231 Process-engineering data [W]
C 232 Technical data packaging: core info & exch [W]
W 233 Systems engineering data representation[A]
W 234 Ship operational logs, records, and messages[X]
W 235 Materials info for des and verif of products [X]
W 236 Furniture product and project data[W]
O Neutral optical-data-interchange format [O]
O Hi-level info plg model for prod l-c spt [O]
O Integ of l-c data for oil/gas production facility (ISO 15926)

INTEGRATED-INFORMATION RESOURCES

APPLICATION MODULES (Technical specifications)

D 1001 Appearance assignment
D 1002 Colour
D 1003 Curve appearance
D 1004 Elemental shape
D 1005 Elemental topological shape
D 1006 Foundation representation
D 1007 General surface appearance
D 1008 Layer assignment
D 1009 Shape appearance and layers

Legend: TS Status
0-10 =O=prop-->apvl for ballot
10-20=A=NP blt circ-->NP apvl
20-60=D=DTS dev-->reg as TS
>60 =T=TS Published

INTEGRATED-APPLICATION RESOURCES

I 101 Draughting (c1=I)
X 102 Ship structures
X 103 E/E connectivity
I 104 Finite element analysis
I 105 Kinematics (c1=I, c2=I)
W 106 Building core model
W 107 Engineering analysis Core ARM
W 108 Prmetizat'n&Constraints for expl geom prod mdls

INTEGRATED-GENERIC RESOURCES

I 41 Fund of prdct descr & spt (e2=E,c1=I)
I 42 Geom & top rep (c1c2=I. e2=I, c1=C. e3=C)
I 43 Repres specialization (e2=I,c1=I,c2=I)
I 44 Product struct confg (e2=I,c1=I)
I 45 Materials (c1=I)
I 46 Visual presentation (c1=I, c2=F)
I 47 Tolerances (c1=I)
X 48 Form features
I 49 Process structure & properties
E 50 Mathematical constructs

APPLICATION-INTERPRETED CONSTRUCTS

I 501 Edge-based wireframe
I 502 Shell-based wireframe
I 503 Geom-bounded 2D wireframe
I 504 Draughting annotation
I 505 Drawing structure & admin.
I 506 Draughting elements
F 507 Geom-bounded surface
F 508 Non-manifold surface
F 509 Manifold surface
I 510 Geom-bounded wireframe
@511 Topological-bounded surface
I 512 Faceted B-representation
I 513 Elementary B-rep
I 514 Advanced B-rep
I 515 Constructive solid geometry
X 516 Mechanical-design context
I 517 Mech-design geom presentation
E 518 Mech-design shaded presentation
I 519 Geometric tolerances (c1=I)
I 520 Assoc draughting elements

DESCRIPTION METHODS

I 1 Overview and fundamental principles (a1=O)
I 11 EXPRESS language ref man. (e2=E,c1=I,c2=C. e3=W(ISO 20303))
I 12 EXPRESS-I language ref man (Type 2 tech report, not a 10303 part)
X 13 Architecture and Methodology reference manual
C 14 EXPRESS X Language reference manual

IMPLEMENTATION METHODS

I 21 Clear-text encoding exch str (c1=I,e2=E,
I 22 Standard data access interface \ a1=X)
I 23 C++ language binding (to #22)
E 24 C language. binding (to #22)
W 25 EXPRESS to OMG XMI (to #22)
C 26 IDL language binding (to #22)
I 27 JAVA language binding (to #22)
D 28 XML rep for EXPRESS-driven data (DTS)
C 29 Ltwt Java binding (to #22)

CONFORMANCE TESTING METHODOLOGY & FRAMEWORK

I 31 General concepts
I 32 Requirements on testing labs and clients
X 33 Structure and use of abstract test suites
F 34 Abstract test methods for Part 21 implementation.
W 35 Abstract test methods for Part 22 implementation.

Legend: Part Status (E, F, I safe to implement)
0=O=Preliminary Stage (Proposal-->appr for NP ballot)
10=A=Proposal Stage (NP ballot circ-->NP approval)
20=W=Preparatory Stage (Wkg Draft devel.-->CD regis)
30=C =Committee Stage (CD circulation-->DIS regis)
40=E =Enquiry Stage (DIS circ.--FDIS registration)
50=F =Approval Stage (FDIS circ-->Int'l Std regis)
@=At ISO, approved for publication (ISO status 40.95 or 50.99)
60=I =Publication Stage (Int'l Std published)
98=X=Project withdrawn

jgnell, 89-Oct.-23; rev. 01-02-12. Origin: ISO 10303 Editing Committee. On-line: http://www.nist.gov/sc5/soap/

Annexe C

Sigles et Traduction

A

AFNOR	**	Association Française de NORmalisation
AIC	Application Interpreted Construct	**
AIM	Application Interpreted Model	modèle interprété de domaine d'application
AIT	Advanced Information Technologies	**
AMT	Advanced Manufacturing Technologies	**
ANSI	American National Standard Institute	**
AP	Application Protocol	protocole d'application
APPP	Application Protocol Project Proposal	
API	Application Programming Interface	interface de programmation d'application
ARM	Application Reference Model	modèle de référence de domaine d'application
ATM	Abstract Test Method	méthode d'essai abstrait
ATS	Abstract Test Suite	suites de tests abstraits

B

BN	**	Bureau de Normalisation
BREP	Boundary REPresentation	représentation par frontières
BSI	British Standards Institution	organisme national de normalisation du Royaume-Uni

C

CAD	Computer Aided Design	Conception Assistée par Ordinateur
CAM	Computer Aided Manufacturing	Conception et Fabrication Assistée par Ordinateur
CAx	Computer Aided something	quelque chose Assistée par Ordinateur
COM	Component Object Model	Modèle Objet par Composants
CD	Committee Draft	projet de comité
CDC	Committee Draft for Comments	**
CEE	**	Communauté Economique Européenne
CEI	**	Comité Electrotechnique Internationale
CEN	**	Comité Européen de Normalisation
CENELEC	**	Comité Européen de Normalisation ELECtrotechnique
CFAO	**	Conception et Fabrication Assistées par Ordinateur
CG		Commission Générale
CIM	Computer Integrated Manufacturing	**
CIME	Computer Integrated Manufacturing and Engineering	**
CTR	Conformance Test Report	rapport d'essai de conformité

D

DIN	Deutsches Institut für Normung	organisme national de normalisation de la République Allemande
DIS	Draft International Standard	projet de norme internationale
DP	Draft Proposal	**

E

EC	Editing Committee	**
EDI	Electronic Data Interchange	Echange de Données Informatisé
EDIF	Electronic Data Interchange Format	**
EDT	**	Echange de Données Techniques
EIA	Electronic Industries Association	**
EN	**	Norme Européenne
ENV	**	Prénorme européenne
EPDEN	European Product Data Exchange Network	**
ESPRIT	European Strategic Program for Research and Development in Information Technologies	**
ETS	Executable Test Suite	suite de test exécutables

F

FEM	Finite Element Modeling	Modélisation par Eléments Finis

G

GPDM	Generic Product Data Model	modèle générique de données de produit
GT	**	Groupe de Travail

H

HD	Harmonization Document	document d'harmonisation

I

ICAM	Integrated Computer Aided Manufacturing	**
IDEF	ICAM DEFinition language 0	**

IDEFx	ICAM DEFinition language 1 eXtended	**
IEC	voir CEI	**
IGES	Initial Graphics Exchange Specification	**
**	Integrated application resources	ressources intégrées de l'application
**	Integrated generic resources	ressources intégrées génériques
IR	Integrated Resource	ressource intégrée
IS	International Standard	norme internationale
ISO	International Standard Organization	Organisation Internationale de Normalisation
ISR	Implementation Specific Requirements	**
IUT	Implementation Under Test	mise en oeuvre (logicielle) à examiner

J

JTC	Joint Technical Committee	**
JWG	Joint Working Group	**

L

LEX	Lexical Analyser	**
LOT	Lexical Object Type	**
LNE		Laboratoire National d'Essai

M

MANDATE	MANufacturing DATa Exchange	**
**	application activity model	modèle d'activité de domaine d'application
	**	**

N

NF	**	Norme Française

NFTI	**	Norme Française, Technologie de l'Information
NIAM	Nijssen's Information Analysis Method	**
NIST	National Institute of Standards and Technology	**
NP	New work item Proposal	nouvelle proposition d'étude
NWI	New Work Item	**

P

PDES	Product Data Exchange using STEP	**
PDM	Product Data Management	Gestion de Données Produit
PICS	Protocole Implementation Conformance Statement	déclaration de conformité d'une mise en oeuvre de protocole
PL1	Programming Language One	**
	Part	partie

R

RIM	resource information model	modèle d'information de ressource
**	integrated resource	ressource intégrée
**	conformance test report	rapport d'essai de conformité
RNE	**	Réseau National d'Essai

S

SADT	Standardized Abstract Test Suite	suite de tests abstraits normalisée
SC	Sub-Committee	Sous-Comité
SDAI	Standard Data Access Interface	interface standard d'accès aux données STEP
SET	**	Standard d'Echange et de Transfert

SETS	Selected Executable Test Suite	suite de tests exécutables sélectionnés
SQL	Structured Query Language	**
STEP	STandard for the Exchange of Product Data Model	**
SWG	Strategic Working Group	**

U

UoF	Unit of Functionality	**
UTE	**	Union Technique de l'Electricité

W

WD	Working Draft	document de travail

Y

YACC	Yet Another Compiler Compiler	**

Annexe D

Une calculatrice en JavaCC

```
/*
 * This is the basic expression grammar for four function
 * Expressions. The grammar support the plus (+), minus (-)
 * multiply (*), and divide (/) operations.
 *
 * See Calc1i.jj for a grammar that parses and implements
 * the functions.
 *
 * Example grammar written 11/1/96 by Chuck McManis (cmcmanis@netcom.com)
 */

options {
LOOKAHEAD=1;
}

PARSER_BEGIN(Calc1)

public class Calc1 {

    public static void main(String args[]) throws ParseException {
        Calc1 parser = new Calc1(System.in);
        while (true) {
            System.out.print("Enter Expression: ");
            System.out.flush();
            try {
                switch (parser.one_line()) {
                    case -1:
                        System.exit(0);
                    default:
                        break;
                }
            } catch (ParseException x) {
                System.out.println("Exiting.");
                throw x;
            }
        }
    }
```

```
}
PARSER_END(Calc1)

SKIP :
{
        " "
|       "\r"
|       "\t"
}

TOKEN :
{
        < EOL: "\n" >
}

TOKEN : /* OPERATORS */
{
        < PLUS: "+" >
|       < MINUS: "-" >
|       < MULTIPLY: "*" >
|       < DIVIDE: "/" >
}

TOKEN :
{
    < CONSTANT: ( <DIGIT> )+ >
|   < #DIGIT: ["0" - "9"] >
}

int one_line() :
{}
{
    sum() <EOL>
        { return 1; }
  | <EOL>
        { return 0; }
  | <EOF>
        { return -1; }
}

void sum() :
{ }
{
```

```
    term() (( <PLUS> | <MINUS> ) term())*
}

void term() :
{ }
{
    unary() (( <MULTIPLY> | <DIVIDE> ) unary())*
}

void unary() :
{ }
{
    <MINUS> element()
  |     element()
}

void element() :
{}
{
    <CONSTANT>
|   "(" sum() ")"
}
```

Annexe E

Une calculatrice avec Lex et Yacc

Scanneur lexical Lex :

```
%{
#include "y.tab.h" extern int yylval;/* Cas standard. Absent si redef. dans yacc */
/* car dans y.tab.h*/
%}
%%
[0-9]+      {yylval = atoi(yytext);
             return INTEGER;}
[-+\.]      {return *yytext;} [ \t\n]    ; .
yyerror("Caractere non valide");
%%
int yywrap(void) {
return 1;
}
```

Analyseur syntaxique Yacc :

```
%token INTEGER
%%
program:
        expr '.'            { printf("=%d\n", $1); }
        |
        ;
expr:
        INTEGER
        | expr '+' expr          { $$ = $1 + $3; }
        | expr '-' expr          { $$ = $1 - $3; }
        ;
%%
int main(void) {
        yyparse();
        return 0;
}
```

Annexe F

Analyse lexicale et syntaxique : Expres.jj

```
/*****************************************************************************************************************************************************
** DESCRIPTION: EXPRESS Grammar and Scanner to be used with JavaCC
** FILENAME:    ExpressParser.jj
** DATE:        MAR 1998 - Jason Goodman <jag164@psu.edu>  (Initial grammar)
** UPDATE:      OCT 1998 - Singva Ma <Singva.Ma@leg.ensieg.inpg.fr>  (Production 294 correction
**                                                    Token DCTET corrected)
**
*****************************************************************************************************************************************************/

options {
   LOOKAHEAD = 2;
   STATIC    = false;
}

PARSER_BEGIN(ExpIdentifier1)
options {
        LOOKAHEAD = 2;
        STATIC = false;
}

PARSER_BEGIN(ExpressParser)

public class ExpressParser{


        //
        // MAIN class used for command line parsing
        //
        public static void main(String args[]) {

                ExpressParser parser;

                if (args.length == 0) {
                        System.out.println("ExpressParser:  Reading from standard input . . .");
                        parser = new ExpressParser(System.in);
                }
                else if (args.length == 1) {
                        System.out.println("ExpressParser:  Reading from file " + args[0] + " . . .");
                        try {
                                parser = new ExpressParser(new java.io.FileInputStream(args[0]));
                        }
                        catch (java.io.FileNotFoundException e) {
                                System.out.println("Express Parser:  File " + args[0] + " not found.");
                                return;
                        }
                }
                else {
                        System.out.println("ExpressParser:  Usage is one of:");
                        System.out.println("         java ExpressParser < inputfile");
                        System.out.println("OR");
                        System.out.println("         java ExpressParser inputfile");
                        return;
                }
                try {

                        parser.syntax();
                                System.out.println("ExpressParser: source parsed successfully.");
                }
                catch (ParseException e) {
                        System.out.println(e.getMessage());
                        System.out.println("ExpressParser:  Encountered errors during parse.");
                }
        } // end main

} // end class ExpressParser

PARSER_END(ExpressParser)


/*********************************************
 * Tokens
 *********************************************/
```

```
SKIP : /* WHITE SPACE */
{
  " "
| "\t"
| "\n"
| "\r"
| "\f"
}


SPECIAL_TOKEN : /* COMMENTS */
{
  <TAIL_REMARK: "--" (~["\n","\r"])* ("\n"|"\r"|"\r\n")>
| <EMBEDDED_REMARK: "(*" (~["*"])* "*" ("*" | (~["*",")"] (~["*"])* "*"))* ")">
}


TOKEN : {
<ABS: "ABS">
| <ABSTRACT: "ABSTRACT">
| <ACOS: "ACOS">
| <AGGREGATE: "AGGREGATE">
| <ALIAS: "ALIAS">
| <AND: "AND">
| <ANDOR: "ANDOR">
| <ARRAY: "ARRAY">
| <AS: "AS">
| <ASIN: "ASIN">
| <ATAN: "ATAN">
| <BAG: "BAG">
| <BEGIN: "BEGIN">
| <BINARY: "BINARY">
| <BLENGTH: "BLENGTH">
| <BOOLEAN: "BOOLEAN">
| <BY: "BY">
| <CASE: "CASE">
| <CONSTANT: "CONSTANT">
| <CONTEXT: "CONTEXT">
| <COS: "COS">
| <DERIVE: "DERIVE">
| <DIV: "DIV">
| <ELSE: "ELSE">
| <END: "END">
| <ENTITY: "ENTITY">
| <ENUMERATION: "ENUMERATION">
| <ESCAPE: "ESCAPE">
| <EXISTS: "EXISTS">
| <EXP: "EXP">
| <FALSE: "FALSE">
| <FIXED: "FIXED">
| <FOR: "FOR">
| <FORMAT: "FORMAT">
| <FROM : "FROM">
| <FUNCTION: "FUNCTION">
| <GENERIC: "GENERIC">
| <HIBOUND: "HIBOUND">
| <HIINDEX: "HIINDEX">
| <IF: "IF">
| <IN: "IN">
| <INSERT: "INSERT">
| <INTEGER: "INTEGER">
| <INVERSE: "INVERSE">
| <LENGTH: "LENGTH">
| <LIKE: "LIKE">
| <LIST: "LIST">
| <LOBOUND: "LOBOUND">
| <LOINDEX: "LOINDEX">
| <LOCAL: "LOCAL">
| <LOG: "LOG">
| <LOG10: "LOG10">
| <LOG2: "LOG2">
| <LOGICAL: "LOGICAL">
| <MOD: "MOD">
| <MODEL: "MODEL">
| <NOT: "NOT">
| <NUMBER: "NUMBER">
| <NVL: "NVL">
| <ODD: "ODD">
| <OF: "OF">
| <ONEOF: "ONEOF">
| <OPTIONAL: "OPTIONAL">
| <OR: "OR">
| <OTHERWISE: "OTHERWISE">
| <PI: "PI">
| <PROCEDURE: "PROCEDURE">
```

```
| <QUERY: "QUERY">
| <REAL: "REAL">
| <REFERENCE: "REFERENCE">
| <REMOVE: "REMOVE">
| <REPEAT: "REPEAT">
| <RETURN: "RETURN">
| <ROLESOF: "ROLESOF">
| <RULE: "RULE">
| <SCHEMA: "SCHEMA">
| <SELECT: "SELECT">
| <SELF: "SELF">
| <SET: "SET">
| <SIN: "SIN">
| <SIZEOF: "SIZEOF">
| <SKIP_E: "SKIP">
| <SQRT: "SQRT">
| <STRING: "STRING">
| <SUBTYPE: "SUBTYPE">
| <SUPERTYPE: "SUPERTYPE">
| <TAN: "TAN">
| <THEN: "THEN">
| <TO: "TO">
| <TRUE: "TRUE">
| <TYPE: "TYPE">
| <TYPEOF: "TYPEOF">
| <UNIQUE: "UNIQUE">
| <UNKNOWN: "UNKNOWN">
| <UNTIL: "UNTIL">
| <USE: "USE">
| <USEDIN: "USEDIN">
| <VALUE: "VALUE">
| <VAR: "VAR">
| <WHERE: "WHERE">
| <WHILE: "WHILE">
| <XOR: "XOR">
| <CONST_E: "CONST_E">
| <END_ALIAS: "END_ALIAS">
| <END_CASE: "END_CASE">
| <END_CONSTANT: "END_CONSTANT">
| <END_CONTEXT: "END_CONTEXT">
| <END_ENTITY: "END_ENTITY">
| <END_FUNCTION: "END_FUNCTION">
| <END_IF: "END_IF">
| <END_LOCAL: "END_LOCAL">
| <END_MODEL: "END_MODEL">
| <END_PROCEDURE: "END_PROCEDURE">
| <END_REPEAT: "END_REPEAT">
| <END_RULE: "END_RULE">
| <END_SCHEMA: "END_SCHEMA">
| <END_TYPE: "END_TYPE">
| <VALUE_IN: "VALUE_IN">
| <VALUE_UNIQUE: "VALUE_UNIQUE">
}

TOKEN : {
   <INTEGER_LITERAL: (["+","-"])? ["0"-"9"] (["0"-"9"])*>
|  <#HEX_LITERAL: "0" ["x","X"] (<HEX_DIGIT>)+>
|  <#HEX_DIGIT: ["0"-"9","a"-"f","A"-"F"]>
|  <#OCTAL_LITERAL: "0" (["0"-"7"])*>
|  <REAL_LITERAL: (["+","-"])? (["0"-"9"])+ "." (["0"-"9"])* (<EXPONENT>)? >
|  <#EXPONENT: ["e","E"] (["+","-"])? (["0"-"9"])+>
|  <SIMPLE_STRING_LITERAL: "\'" (<DIGIT> | <LETTER> | <ESCAPED_CHAR> | <PRINT_CHAR> | " " )*  "\'" >
|  <#ESCAPED_CHAR: "\\" (["n","t","b","r","f","\\","\'","\""]) >
|  <#PRINT_CHAR: "~" | "!" | "@" | "#" | "$" | "%" | "^" |"&" | "*" | "(" | <RPAREN> | "_" | "-" | "+" | "=" | "|" | "{" | "}" | "[" | "]" | ":" | ";" | "," | "." |
|  <ENCODED_STRING_LITERAL: "\"" (<ENCODED_CHARACTER>)+ "\"">
|  <BINARY_LITERAL: "%" (["0","1"])* >
|  <OCTET: <HEX_DIGIT> <HEX_DIGIT> >
|  <#ENCODED_CHARACTER: <OCTET> <OCTET> <OCTET> <OCTET> >
|  <SIMPLE_ID: <LETTER> (<LETTER> | <DIGIT> | "_")*>
|  <#LETTER: ["A"-"Z", "a"-"z"]>
|  <#DIGIT: ["0"-"9"]>
}

TOKEN :
{
  < LPAREN: "(" >
| < RPAREN: ")" >
| < LBRACE: "{" >
| < RBRACE: "}" >
| < LBRACKET: "[" >
| < RBRACKET: "]" >
| < SEMICOLON: ";" >
| < COLON: ":" >
| < COMMA: "," >
| < DOT: "." >
```

```
}

TOKEN :
{
  < ASSIGN: ":=" >
| < GT: ">" >
| < LT: "<" >
| < LTE: "<=" >
| < GTE: ">=" >
| < QMARK: "?" >
| < POWER: "**" >
| < BSLASH: "\\" >
| < NE: "<>" >
| < EQ: "=" >
| < ASSIGN_NE: ":<>:">
| < ASSIGN_EQ: ":=:">
| < PLUS: "+" >
| < MINUS: "-" >
| < STAR: "*">
| < SLASH: "/">
| < SC_OR: "||" >
| < SRS: "<*" >
| < BIT_OR: "|" >
}

/*************************************************
*************************************************
** Grammar
*************************************************
*************************************************/


/********** PRODUCTION 145 **********/

void attribute_ref() : {}
{
    attribute_id()
}

/********** PRODUCTION 146 **********/

void constant_ref() : {}
{
    constant_id()
}

/********** PRODUCTION 147 **********/

void entity_ref() : {}
{
    entity_id()
}


/********** PRODUCTION 148 **********/

void enumeration_ref() : {}
{
    enumeration_id()
}

/********** PRODUCTION 149 **********/

void function_ref() : {}
{
    function_id()
}


/********** PRODUCTION 150 **********/

void parameter_ref() : {}
{
    parameter_id()

}

/********** PRODUCTION 151 **********/

void procedure_ref() : {}
{
    procedure_id()
}

/********** PRODUCTION 152 **********/
```

```
void schema_ref() : {}
{
    schema_id()
}

/********** PRODUCTION 153 **********/

void type_label_ref() : {}
{
    type_label_id()
}


/********** PRODUCTION 154 **********/

void type_ref() : {}
{
    type_id()
}

/********** PRODUCTION 155 **********/

void variable_ref() : {}
{
    variable_id()
}

/********** PRODUCTION 156 **********/

void abstract_supertype_declaration() : {}
{
<ABSTRACT> <SUPERTYPE> (subtype_constraint())?
}

/********** PRODUCTION 157 **********/

void actual_parameter_list() : {}
{
<LPAREN> parameter() (<COMMA> parameter())* <RPAREN>
}

/********** PRODUCTION 158 **********/
void add_like_op() : {}
{
<PLUS> | <MINUS> | <OR> | <XOR>
}

/********** PRODUCTION 159 **********/
void aggregate_initializer() : {}
{
 <LBRACKET> (element() (<COMMA> element())*)?  <RBRACKET>
}

/********** PRODUCTION 160 **********/
void aggregate_source() : {}
{
    simple_expression()
}

/********** PRODUCTION 161 **********/
void aggregate_type() : {}
{
    <AGGREGATE> (<COLON> type_label())? <OF> parameter_type()
}

/********** PRODUCTION 162 **********/
void aggregation_types() : {}
{
    array_type()
   | bag_type()
   | list_type()
   | set_type()
}

/********** PRODUCTION 163 **********/
void algorithm_head() : {}
{
    (declaration())* (constant_decl())? (local_decl())
}

/********** PRODUCTION 164 **********/
void alias_stmt() : {}
{
    <ALIAS> variable_id() <FOR> general_ref() (qualifier())* <SEMICOLON> (stmt())+ <END_ALIAS> <SEMICOLON>
}
```

```
/********** PRODUCTION 165 **********/
void array_type() : {}
{
    <ARRAY> bound_spec() <OF> (<OPTIONAL>)? (<UNIQUE>)? base_type()
}

/********** PRODUCTION 166 **********/
void assignment_stmt() : {}
{
    general_ref() (qualifier())* <ASSIGN> expression() <SEMICOLON>
}

/********** PRODUCTION 167 **********/
void attribute_decl() : {}
{
    attribute_id() | qualified_attribute())
}

/********** PRODUCTION 168 **********/
void attribute_id() : {}
{
    simple_id()
}

/********** PRODUCTION 169 **********/
void attribute_qualifier() : {}
{
    <DOT> attribute_ref()
}

/********** PRODUCTION 170 **********/
void bag_type() : {}
{
    <BAG> (bound_spec())? <OF> base_type()
}

/********** PRODUCTION 171 **********/
void base_type() : {}
{
    aggregation_types()
   | simple_types()
   | named_types()
}


/********** PRODUCTION 172 **********/
void binary_type() : {}
{
    <BINARY> (width_spec())?
}

/********** PRODUCTION 173 **********/
void boolean_type() : {}
{
    <BOOLEAN>
}

/********** PRODUCTION 174 **********/
void bound_1() : {}
{
    numeric_expression()
}

/********** PRODUCTION 175 **********/
void bound_2() : {}
{
    numeric_expression()
}

/********** PRODUCTION 176 **********/
void bound_spec() : {}
{
    <LBRACKET> bound_1() <COLON> bound_2() <RBRACKET>
}

/********** PRODUCTION 177 **********/
void built_in_constant() : {}
{
    <CONST_E> | <PI> | <SELF> | <QMARK>
}

/********** PRODUCTION 178 **********/
void built_in_function() :
{
}
{
```

```
    <ABS> | <ACOS> | <ASIN> | <ATAN> | <BLENGTH> | <COS>  |
    <EXISTS> | <EXP> | <FORMAT> | <HIBOUND> | <HIINDEX> |
    <LENGTH> | <LOBOUND> | <LOINDEX> | <LOG> | <LOG2>   |
    <LOG10> | <NVL> | <ODD> | <ROLESOF> | <SIN> | <SIZEOF> |
    <SQRT> | <TAN> | <TYPEOF> | <USEDIN> | <VALUE> |
    <VALUE_IN> | <VALUE_UNIQUE>
}

/********** PRODUCTION 179 **********/
void built_in_procedure() : {}
{
    <INSERT> | <REMOVE>
}

/********** PRODUCTION 180 **********/
void case_action() : {}
{
    case_label() (<COMMA> case_label())* <COLON> stmt()
}

/********** PRODUCTION 181 **********/
void case_label() : {}
{
    expression()
}

/********** PRODUCTION 182 **********/
void case_stmt() : {}
{
    <CASE> selector() <OF> (case_action())* (<OTHERWISE> <COLON> stmt())? <END_CASE> <SEMICOLON>
}

/********** PRODUCTION 183 **********/
void compound_stmt() : {}
{
    <BEGIN> (stmt())+ <END> <SEMICOLON>
}

/********** PRODUCTION 184 **********/
void constant_body() : {}
{
    constant_id() <COLON> base_type() <ASSIGN> expression() <SEMICOLON>
}

/********** PRODUCTION 185 **********/
void constant_decl() : {}
{
    <CONSTANT> (constant_body())+ <END_CONSTANT> <SEMICOLON>
}

/********** PRODUCTION 186 **********/
void constant_factor() : {}
{
    built_in_constant() | constant_ref()
}

/********** PRODUCTION 187 **********/
void constant_id() : {}
{
    simple_id()
}

/********** PRODUCTION 188 **********/
void constructed_types() : {}
{
    enumeration_type() | select_type()
}

/********** PRODUCTION 189 **********/
void declaration() : {}
{
    entity_decl() | function_decl() | procedure_decl() | type_decl()
}

/********** PRODUCTION 190 **********/
void derived_attr() : {}
{
    attribute_decl() <COLON> base_type() <ASSIGN> skipDeriveAssignment() <SEMICOLON>//expression() <SEMICOLON>
}

/********** PRODUCTION 191 **********/
void derive_clause() : {}
{
    <DERIVE> (derived_attr())+
}
```

```
/********** PRODUCTION 192 **********/
void domain_rule() : {}
{
    (label() <COLON>)? expression()
}

/********** PRODUCTION 193 **********/
void element() : {}
{
    expression() (<COLON> repetition())?
}

/********** PRODUCTION 194 **********/
void entity_body() : {}
{
    (explicit_attr())* (derive_clause())? (inverse_clause())? (unique_clause())? (where_clause())?
}

/********** PRODUCTION 195 **********/
void entity_constructor() : {}
{
    entity_ref() <LPAREN> (expression() (<COMMA> expression())*)? <RPAREN>
}

/********** PRODUCTION 196 **********/
void entity_decl() : { }
{
    entity_head() entity_body() <END_ENTITY> <SEMICOLON>
}

/********** PRODUCTION 197 **********/
void entity_head() : {}
{
<ENTITY> entity_id() (subsuper())? <SEMICOLON>
}

/********** PRODUCTION 198 **********/
void entity_id() : {}
{
    simple_id()
}

/********** PRODUCTION 199 **********/
void enumeration_id() : {}
{
    simple_id()
}

/********** PRODUCTION 200 **********/
void enumeration_reference() : {}
{
    (type_ref() <DOT>)? enumeration_ref()
}

/********** PRODUCTION 201 **********/
void enumeration_type() : {}
{
    <ENUMERATION> <OF> <LPAREN> enumeration_id() (<COMMA> enumeration_id())* <RPAREN>
}

/********** PRODUCTION 202 **********/
void escape_stmt() : {}
{
    <ESCAPE> <SEMICOLON>
}

/********** PRODUCTION 203 **********/
void explicit_attr() : {}
{

attribute_decl() (<COMMA> attribute_decl())* <COLON> (<OPTIONAL>)? <SEMICOLON>
}

/********** PRODUCTION 204 **********/
void expression() : {}
{
simple_expression() ( rel_op_extended() simple_expression())?
}

/********** PRODUCTION 205 **********/
void factor() : {}
{
simple_factor() (<POWER> simple_factor())?
}

/********** PRODUCTION 206 **********/
```

```
void formal_parameter() : {}
{
parameter_id() (<COMMA> parameter_id())* <COLON> parameter_type()
}

/********** PRODUCTION 207 **********/
void function_call() : {}
{
(built_in_function() | function_ref()) (actual_parameter_list())?
}

/********** PRODUCTION 208 **********/
void function_decl() : {}
{
function_head() (algorithm_head())? (stmt())+ <END_FUNCTION> <SEMICOLON>
}

/********** PRODUCTION 209 *********/
void function_head() :
{

}
{
<FUNCTION> skipFunction() <END_FUNCTION> <SEMICOLON>
function_id()
(<LPAREN> formal_parameter() (<SEMICOLON> formal_parameter())* <RPAREN>)? <COLON> parameter_type()  <SEMICOLON>
}

/********** PRODUCTION 210 **********/
void function_id() : {}
{
    simple_id()
}

/********** PRODUCTION 211 **********/
void generalized_types() : {}
{
    aggregate_type()
   | general_aggregation_types()
   | generic_type()
}

/********** PRODUCTION 212 **********/
void general_aggregation_types() : {}
{
    general_array_type()
   | general_bag_type()
   | general_list_type()
   | general_set_type()
}

/********** PRODUCTION 213 **********/
void general_array_type() : {}
{
<ARRAY>  (bound_spec())? <OF> (<OPTIONAL>)? (<UNIQUE>)? parameter_type()

}

/********** PRODUCTION 214 **********/
void general_bag_type() : {}
{
<BAG> (bound_spec())? <OF> parameter_type()
}

/********** PRODUCTION 215 **********/
void general_list_type() : {}
{
<LIST> (bound_spec())? <OF> (<UNIQUE>)? parameter_type()
}

/********** PRODUCTION 216 **********/
void general_ref() : {}
{
  parameter_ref()
| variable_ref()
}

/********** PRODUCTION 217 **********/
void general_set_type() : {}
{
<SET> (bound_spec())? <OF> parameter_type()
}

/********** PRODUCTION 218 **********/
void generic_type() : {}
{
```

```
<GENERIC> (<COLON> type_label())?
}

/********** PRODUCTION 219 **********/
void group_qualifier() : {}
{
<BSLASH> entity_ref()
}

/********** PRODUCTION 220 **********/
void if_stmt() : {}
{
<IF> expression() <THEN> (stmt())+ (<ELSE> (stmt())+)? <END_IF> <SEMICOLON>
}

/********** PRODUCTION 221 **********/
void increment() : {}
{
numeric_expression()
}

/********** PRODUCTION 222 **********/
void increment_control() : {}
{
variable_id() <ASSIGN> bound_1() <TO> bound_2() (<BY> increment())?
}

/********** PRODUCTION 223 **********/
void index() : {}
{
numeric_expression()
}

/********** PRODUCTION 224 **********/
void index_1() : {}
{
index()
}

/********** PRODUCTION 225 **********/
void index_2() : {}
{
index()
}

/********** PRODUCTION 226 **********/
void index_qualifier() : {}
{
<LBRACKET> index_1() (<COLON> index_2())? <RBRACKET>
}

/********** PRODUCTION 227 **********/
void integer_type() : {}
{
<INTEGER>
}

/********** PRODUCTION 228 **********/
void interface_specification() : {}
{
  reference_clause()
| use_clause()
}

/********** PRODUCTION 229 **********/
void interval() : {}
{
<LBRACE> interval_low() interval_op() interval_item() interval_op() interval_high() <RBRACE>
}

/********** PRODUCTION 230 **********/
void interval_high() : {}
{
simple_expression()
}

/********** PRODUCTION 231 **********/
void interval_item() : {}
{
simple_expression()
}

/********** PRODUCTION 232 **********/
void interval_low() : {}
{
simple_expression()
```

```
}

/********** PRODUCTION 233 **********/
void interval_op() : {}
{
<LT> | <LTE>
}

/********** PRODUCTION 234 **********/
void inverse_attr() : {}
{
attribute_decl() <COLON> ((<SET>|<BAG>) (bound_spec())? <OF>)? entity_ref() <FOR> attribute_ref() <SEMICOLON>
}

/********** PRODUCTION 235 **********/
void inverse_clause() : {}
{
<INVERSE> (inverse_attr())+
}

/********** PRODUCTION 236 **********/
void label() : {}
{
    simple_id()
}

/********** PRODUCTION 237 **********/
void list_type() : {}
{
    <LIST>  (bound_spec())? <OF> (<UNIQUE>)? base_type()
}

/********** PRODUCTION 238 **********/
void literal() : {}
{
    <BINARY_LITERAL> | <INTEGER_LITERAL> | logical_literal() | <REAL_LITERAL> | string_literal()
}

/********** PRODUCTION 239 **********/
void local_decl() : {}
{
<LOCAL> local_variable() (local_variable())* <END_LOCAL> <SEMICOLON>
}

/********** PRODUCTION 240 **********/
void local_variable() : {}
{
variable_id() (<COMMA> variable_id())* <COLON> parameter_type() (<ASSIGN> expression())? <SEMICOLON>
}

/********** PRODUCTION 241 **********/
void logical_expression() : {}
{
expression()
}

/********** PRODUCTION 242 **********/
void logical_literal() : {}
{
<FALSE> | <TRUE> | <UNKNOWN>
}

/********** PRODUCTION 243 **********/
void logical_type() : {}
{
<LOGICAL>
}

/********** PRODUCTION 244 **********/
void multiplication_like_op() : {}
{
<STAR> | <SLASH> | <DIV> | <MOD> | <AND> | <SC_OR>
}

/********** PRODUCTION 245 **********/
void named_types() : {}
{
entity_ref() | type_ref()
}

/********** PRODUCTION 246 **********/
void named_type_or_rename() : {}
{
named_types() (<AS> (entity_id() | type_id()))?
}
```

```
/********** PRODUCTION 247 **********/
void null_stmt() : {}
{
<SEMICOLON>
}

/********** PRODUCTION 248 **********/
void number_type() : {}
{
<NUMBER>
}

/********** PRODUCTION 249 **********/
void numeric_expression() : {}
{
simple_expression()
}

/********** PRODUCTION 250 **********/
void one_of() : {}
{
<ONEOF> <LPAREN> supertype_expression() (<COMMA> supertype_expression())* <RPAREN>
}

/********** PRODUCTION 251 **********/
void parameter() : {}
{
expression()
}

/********** PRODUCTION 252 **********/
void parameter_id() :
{
   void s;
}
{
s=simple_id()
{
   return s;
}
}

/********** PRODUCTION 253 **********/
void parameter_type() : {}
{
generalized_types() | named_types() | simple_types()
}

/********** PRODUCTION 254 **********/
void population() : {}
{
entity_ref()
}

/********** PRODUCTION 255 **********/
void precision_spec() : {}
{
numeric_expression()
}

/********** PRODUCTION 256 **********/
void primary() :
{
}
{
literal()
| (qualifiable_factor() (qualifier())*)
}

/********** PRODUCTION 257 **********/
void procedure_call_stmt() : {}
{
(built_in_procedure() | procedure_ref()) (actual_parameter_list())? <SEMICOLON>
}

/********** PRODUCTION 258 **********/
void procedure_decl() : {}
{
procedure_head() //(algorithm_head())? (stmt())* <END_PROCEDURE> <SEMICOLON>
}

/********** PRODUCTION 259 **********/
void procedure_head() : {}
{
<PROCEDURE> skipProcedure() <END_PROCEDURE> <SEMICOLON>//procedure_id() (<LPAREN> (<VAR>)? formal_parameter() (<SEMICOLON> (<VAR>)? formal_parameter())* <RPAREN>
```

```
}

/********** PRODUCTION 260 **********/
void procedure_id() : {}
{
    simple_id()
}

/********** PRODUCTION 261 **********/

void qualifiable_factor() : {}
{
 constant_factor()
| function_call()
| general_ref()
| population()
| attribute_ref()
}

/********** PRODUCTION 262 **********/
void qualified_attribute() : {}
{
<SELF> group_qualifier() attribute_qualifier()
}

/********** PRODUCTION 263 **********/
void qualifier() : {}
{
  attribute_qualifier()
| group_qualifier()
| index_qualifier()
}

/********** PRODUCTION 264 **********/
void query_expression() : {}
{
<QUERY> <LPAREN> variable_id() <SRS> aggregate_source() <BIT_OR> logical_expression() <RPAREN>
}

/********** PRODUCTION 265 **********/
void real_type() : {}
{
<REAL> (<LPAREN> precision_spec() <RPAREN>)?
}

/********** PRODUCTION 266 **********/
void referenced_attribute() : {}
{
  attribute_ref()
| qualified_attribute()
}

/********** PRODUCTION 267 **********/
void reference_clause() : {}
{
    <REFERENCE> <FROM> schema_ref() (<LPAREN> resource_or_rename() (<COMMA> resource_or_rename())* <RPAREN> )? <SEMICOLON>
}

/********** PRODUCTION 268 **********/
void rel_op() : {}
{
<LT> | <GT> | <LTE> | <GTE> | <NE> | <EQ> | <ASSIGN_NE> | <ASSIGN_EQ>
}

/********** PRODUCTION 269 **********/
void rel_op_extended() : {}
{
rel_op() | <IN> | <LIKE>
}

/********** PRODUCTION 270 **********/
void rename_id() : {}
{
constant_id()
| s=entity_id()
| s=function_id()
| s=procedure_id()
| s=type_id()
}

/********** PRODUCTION 271 **********/
void repeat_control() : {}
{
(increment_control())? (while_control())? (until_control())?
}
```

```
/********** PRODUCTION 272 **********/
void repeat_stmt() : {}
{
<REPEAT> repeat_control() <SEMICOLON> (stmt())+ <END_REPEAT> <SEMICOLON>
}

/********** PRODUCTION 273 **********/
void repetition() : {}
{
numeric_expression()
}

/********** PRODUCTION 274 **********/
void resource_or_rename() : {}
{
resource_ref() (<AS> rename_id())?
}

/********** PRODUCTION 275 **********/
void resource_ref() : {}
{
  constant_ref()
| entity_ref()
| function_ref()
| procedure_ref()
| type_ref()
}

/********** PRODUCTION 276 **********/
void return_stmt() : {}
{
<RETURN> (<LPAREN> expression() <RPAREN>)? <SEMICOLON>
}

/********** PRODUCTION 277 **********/
void rule_decl() : {}
{
rule_head() //(algorithm_head())? (stmt())* where_clause() <END_RULE> <SEMICOLON>
}

/********** PRODUCTION 278 **********/
void rule_head() : {}
{
<RULE>rule_id()
<END_RULE> <SEMICOLON>
//<FOR> <LPAREN> entity_ref() (<COMMA> entity_ref())* <RPAREN> <SEMICOLON>
}

/********** PRODUCTION 279 **********/
void rule_id() : {}
{
      simple_id()
}

/********** PRODUCTION 280 **********/
void schema_body() : {}
{
(interface_specification())* (constant_decl())? ( declaration() | rule_decl())*
}

/********** PRODUCTION 281 **********/
void schema_decl() :
{
   void s;
}
{
<SCHEMA> schema_id()
<SEMICOLON> schema_body() <END_SCHEMA> <SEMICOLON>
}

/********** PRODUCTION 282 **********/
void schema_id() : {}
{
    simple_id()
}

/********** PRODUCTION 283 **********/
void selector() : {}
{
expression()
}

/********** PRODUCTION 284 **********/
void select_type() : {}
{
<SELECT> <LPAREN> named_types() (<COMMA> named_types() )* <RPAREN>
```

```
}

/********** PRODUCTION 285 **********/
void set_type() : {}
{
<SET> (bound_spec())? <OF> base_type()
}

/********** PRODUCTION 286 **********/
void sign() : {}
{
<PLUS> | <MINUS>
}

/********** PRODUCTION 287 **********/
void simple_expression() : {}
{

term() (add_like_op() term())*
}

/********** PRODUCTION 288 **********/
void simple_factor() : {}
{
   LOOKAHEAD(3)
  aggregate_initializer()
| entity_constructor()
| enumeration_reference()
| interval()
| query_expression()
| ((unary_op())? ( (<LPAREN> expression() <RPAREN>) | primary()))
}

/********** PRODUCTION 289 **********/
void simple_types() : {}
{
  binary_type()
| boolean_type()
| integer_type()
| logical_type()
| number_type()
| real_type()
| string_type()
}

/********** PRODUCTION 290 **********/
void skip_stmt() : {}
{
<SKIP_E> <SEMICOLON>
}

/********** PRODUCTION 291 **********/
void stmt() : {}
{
  alias_stmt()
| assignment_stmt()
| case_stmt()
| compound_stmt()
| escape_stmt()
| if_stmt()
| null_stmt()
| procedure_call_stmt()
| repeat_stmt()
| return_stmt()
| skip_stmt()
}

/********** PRODUCTION 292 **********/
void string_literal() : {}
{
<SIMPLE_STRING_LITERAL> | <ENCODED_STRING_LITERAL>
}

/********** PRODUCTION 293 **********/
void string_type() : {}
{
<STRING> (width_spec())?
}

/********** PRODUCTION 294 **********/
void subsuper() : {}
{
(supertype_constraint() | subtype_declaration())+
}

/********** PRODUCTION 295 **********/
```

```
void subtype_constraint() : {}
{
<OF> <LPAREN> supertype_expression() <RPAREN>
}

/********** PRODUCTION 296 **********/
void subtype_declaration() : {}
{
<SUBTYPE> <OF> <LPAREN> entity_ref() (<COMMA> entity_ref() )* <RPAREN>
}

/********** PRODUCTION 297 **********/
void supertype_constraint() : {}
{
  abstract_supertype_declaration()
| supertype_rule()
}

/********** PRODUCTION 298 **********/
void supertype_expression() : {}
{
supertype_factor() (<ANDOR> supertype_factor())*
}

/********** PRODUCTION 299 **********/
void supertype_factor() : {}
{
supertype_term() (<AND> supertype_term())*
}

/********** PRODUCTION 300 **********/
void supertype_rule() : {}
{
<SUPERTYPE> subtype_constraint()
}

/********** PRODUCTION 301 **********/
void supertype_term() : {}
{
  entity_ref()
| one_of()
| <LPAREN> supertype_expression() <RPAREN>
}

/********** PRODUCTION 302 **********/
void syntax() : {}
{
(schema_decl())+
}

/********** PRODUCTION 303 **********/
void term() : {}
{
factor() (multiplication_like_op() factor())*
}

/********** PRODUCTION 304 **********/
void type_decl() : {}
{
type_id()
<EQ> underlying_type() <SEMICOLON> (where_clause())? <END_TYPE> <SEMICOLON>
}

/********** PRODUCTION 305 **********/
void type_id() : {}
{
    simple_id()
}

/********** PRODUCTION 306 **********/
void type_label() : {}
{
    type_label_id()
   | type_label_ref()
}

/********** PRODUCTION 307 **********/
void type_label_id() : {}
{
    simple_id()
}

/********** PRODUCTION 308 **********/
void unary_op() : {}
{
<PLUS> | <MINUS> | <NOT>
```

```
}

/********** PRODUCTION 309 **********/
void underlying_type() : {}
{
    constructed_types()
   | aggregation_types()
   | simple_types()
   | type_ref()
}

/********** PRODUCTION 310 **********/
void unique_clause() : {}
{
    <UNIQUE> unique_rule() <SEMICOLON> (unique_rule() <SEMICOLON>)*
}

/********** PRODUCTION 311 **********/
void unique_rule() : {}
{
(label() <COLON>)? referenced_attribute() (<COMMA> referenced_attribute())*
}

/********** PRODUCTION 312 **********/
void until_control() : {}
{
<UNTIL> expression()
}

/********** PRODUCTION 313 **********/
void use_clause() : {}
{
<USE> <FROM> schema_ref() (<LPAREN> named_type_or_rename() (<COMMA> named_type_or_rename())* <RPAREN> )? <SEMICOLON>
}

/********** PRODUCTION 314 **********/
void variable_id() :
{
   void s;
}
{
s=simple_id()
{
  return s;
}
}

/********** PRODUCTION 315 *********/
void where_clause() : {}
{
<WHERE> domain_rule() <SEMICOLON> (domain_rule() <SEMICOLON> )*
}


/********** PRODUCTION 316 **********/
void while_control() : {}
{
<WHILE> logical_expression()
}

/********** PRODUCTION 317 **********/
void width() : {}
{
numeric_expression()
}

/********** PRODUCTION 318 **********/
void width_spec() : {}
{
<LPAREN> width() <RPAREN> (<FIXED>)?
}


/********** PRODUCTION 140 **********/
void simple_id() : {}
{
    <SIMPLE_ID>
}
```

Annexe G

Mode Express pour l'outil GNU XEmacs

```
;; Express.el
;; ----------
;;
;; author: Singva Ma, doctorant equipe modélisation
;;         <singva.ma@leg.ensieg.inpg.fr>
;;
;; created: 12 March 2000.
;;
;; express.el - major mode for editing EXPRESS source with GNU Emacs
;; For convient use add something like the
;; following to your .emacs start-up file:
;;
;; (global-font-lock-mode t)
;; (load "~/xemacs/express/express")
;; (setq auto-mode-alist (cons '("\\(\\.exp$\\|\\.express$\\)" . express-mode) auto-mode-alist))
;; (setq express-mode-hook '(lambda () (setq fill-column 74)))

(require 'shell)
(require 'compile)

;; Constants used in all Express-mode buffers.
(defconst express-indent-level 2
  "*The indentation in Express-mode.")

(defconst express-comment-column 40
  "*The goal comment column in Express-mode buffers.")

(defconst exp-block-beg-kw "\\(ALIAS[ ]\\|CASE[ ]|\\CONSTANT[ ]\\|CONTEXT[ ]\\|[ ]ENTITY[ ]\\|FUNCTION[ ]\\|IF[ ]\\|LOCAL[ \t\n]\\|[ ]MODEL[ ]\\|PROCEDURE[ ]\\|R
  "Regular expression for keywords which begin blocks in Express-mode.")
(defconst exp-block-end-kw "\\(END_ALIAS[ ;]\\|END_CASE[ ;]|\\END_CONSTANT[ ;]\\|END_CONTEXT[ ;]\\|END_ENTITY[ ;]\\|END_FUNCTION[ ;]\\|END_IF[ ;]\\|END_LOCAL[ ;]
  "Regular expression for keywords which end blocks.")

;; variables for compilation part
(defvar exp-last-indent-type "unknown"
  "String to tell line type.")

(defvar express-comment-region "--"
  "*String inserted by \\[express-comment-region]\
 at start of each line in region.")

;;;###autoload*
;; Compilation
(defvar express-command "/uh/f3m/step/expToJava"
  "Command to run Express compiler to Java. express must be in the path.
The option and the name of the file, will be added to this string.")

;;;###autoload
(defvar express-last-buffer-compiled nil
  "Buffer which was last compiled.")
;;;###autoload
(defvar express-zap-file nil
  "Temporary file name used for text being sent as input to Express.
Should be a simple file name with no extension or directory specification."
  )
;;;###autoload
(defvar express-directory "."
  "*Directory in which temporary files are left.
You can make this /tmp if your functions has no relative directories in it.")

;;;###autoload
(defvar express-offer-save t
  "*If non-nil, ask about saving modified buffers before \\[express-file] is run.")
;;;###autoload
(defvar express-last-temp-file nil
  "Latest temporary file generated by \\[express-region] and \\[express-compile-buffer].
Deleted when the \\[express-region] or \\[express-compile-buffer] is next run, or when the
express-shell goes away.")

;;;###autoload
(defvar express-trailer nil
  "String appended after the end of a region sent to express by \\[express-region].")

(defvar express-shell-file-name nil
  "*If non-nil, the shell file name to run in the subshell used to run Express.")
```

```
;; Syntax Table
(defvar express-mode-syntax-table nil
  "Syntax table used in Express-mode buffers.")


(if express-mode-syntax-table
    ()
  (setq express-mode-syntax-table (make-syntax-table))
  (modify-syntax-entry ?\\ "." express-mode-syntax-table)
  (modify-syntax-entry ?/ "." express-mode-syntax-table)
  (modify-syntax-entry ?* "." express-mode-syntax-table)
  (modify-syntax-entry ?+ "." express-mode-syntax-table)
  (modify-syntax-entry ?- "." express-mode-syntax-table)
  (modify-syntax-entry ?= "." express-mode-syntax-table)
  (modify-syntax-entry ?< "." express-mode-syntax-table)
  (modify-syntax-entry ?> "." express-mode-syntax-table)
  (modify-syntax-entry ?\[ "(]" express-mode-syntax-table)
  (modify-syntax-entry ?\] ")[" express-mode-syntax-table)
  (modify-syntax-entry ?& "." express-mode-syntax-table)
  (modify-syntax-entry ?| "." express-mode-syntax-table)
  (modify-syntax-entry ?\' "'" express-mode-syntax-table)
  (modify-syntax-entry ?\n ">" express-mode-syntax-table)
  (set-syntax-table express-mode-syntax-table))


;; Abbrev Table
(defvar express-mode-abbrev-table nil
  "Abbrev table used in Express-mode buffers.")

(define-abbrev-table 'express-mode-abbrev-table ())


;; Mode Map
;(defvar express-mode-map nil "Keymap for Express mode.")
(defvar express-mode-map ()   "Keymap used in express-mode.")

(if express-mode-map
    nil
  (setq express-mode-map (make-sparse-keymap))
  (define-key express-mode-map "\r" 'express-return)
  (define-key express-mode-map "\t" 'express-indent-line)
  (define-key express-mode-map "\M-;" 'express-comment)
  (define-key express-mode-map "\C-ct" 'express-line-type)
  (define-key express-mode-map "\C-ci" 'express-indent-type)
  (define-key express-mode-map "\C-c\C-p" 'beginning-of-express-function)
  (define-key express-mode-map "\C-c\C-n" 'end-of-express-function)
  (define-key express-mode-map "\M-\r" 'newline))

(defvar menu-bar-express-menu (make-sparse-keymap "Express"))
(define-key express-mode-map [menu-bar express] (cons "Express" menu-bar-express-menu))
;(define-key menu-bar-express-menu [C2]
;  '("Check Buffer Syntax" . express-compile-buffer))
;(define-key menu-bar-express-menu [C3]
;  '("Check Function Syntax" . express-compile-function))
(define-key menu-bar-express-menu [C4]
  '("Indent Function" . express-indent-function))
(define-key menu-bar-express-menu [C5]
  '("Indent Buffer" . express-indent-buffer))
(define-key menu-bar-express-menu [C6]
  '("Ident Region" . express-indent-region))
(define-key menu-bar-express-menu [end-of-express-function]
  '("End of function" . end-of-express-function))
(define-key menu-bar-express-menu [beginning-of-express-function]
  '("Beginning of function" . beginning-of-express-function))



(define-key express-mode-map [menu-bar express express-compile]
  '(cons "Express Compile" express-compile))
(define-key express-mode-map [menu-bar express express-compile-buffer]
  '(cons "Express Compile Buffer" express-compile-buffer))
(define-key express-mode-map [menu-bar express express-compile-function]
  '(cons "Express Compile Function" express-compile-function))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Express font-lock support
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; emacs-20:  keyword builtin warning variable-name type string reference function-name comment
;; emacs-19:  comment string keyword function-name variable-name type reference
(defvar express-font-lock-keywords
  '(
    ("['\"][^'\"\n]*['\"]" . font-lock-string-face)
```

```
    ("[ ,;\t\n\(\)]\\(ARRAY\\|BAG\\|SET\\|LIST\\|BINARY\\|BOOLEAN\\|INTEGER\\|NUMBER\\|REAL\\|LOGICAL\\|STRING\\)[ ,;\t\n\(\)]" . font-lock-type-face)
    ("[ ,;\t\n\[\(\(]\\(ABS\\|ABSTRACT\\|ACOS\\|AGGREGATE\\|ALIAS\\|AND\\|ANDOR\\|ARRAY\\|AS\\|ASIN\\|ATAN\\|BAG\\|BEGIN\\|BINARY\\|BLENGTH\\|BOOLEAN\\|BY\\|CASE\
    ("[(]\\(SCHEMA\\|TYPE\\)[ ,;\t\n\(\[\)]" . font-lock-keyword-face)
    ("[ \t\n]\\(OF\\)[ \t\n]" . font-lock-keyword-face)
    ("[ ,;\t]\\(resume\\|return\\|pause\\|quit\\|abort\\|^//end\\)[ ,;\t\n]" . font-lock-type-face)
    ("^ *\\(function\\)[ \t\n]" . font-lock-function-name-face)
    ("[^ ,;\t]\\case\\[ \t\n]" . label)
    ("[ ,;\t\n\[\(]\\(ABS\\|ABSTRACT\\|ACOS\\|AGGREGATE\\|ALIAS\\|AND\\|ANDOR\\|ARRAY\\|AS\\|ASIN\\|ATAN\\|BAG\\|BEGIN\\|BINARY\\|BLENGTH\\|BOOLEAN\\|BY\\|CASE\\|

   )
  "Default expressions to highlight in Express mode."
  )



;; Express Mode
(defun express-mode ()
  "Major mode for editing Express source files.  Version 1.0, may 1997.
Will run express-mode-hook if it is non-nil.  Auto-fill-mode seems to work.
Filling does not work (yet).
Special Key Bindings:
\\{express-mode-map}
Variables:
  express-indent-level                  Level to indent blocks.
  express-comment-column                Goal column for on-line comments.
  fill-column                          Column used in auto-fill (default=70).
Commands:
  express-mode                          Enter Express major mode.
  express-return                        Handle return with indenting.
  express-indent-line                   Indent line for structure.
  express-indent-buffer                 Indent buffer for structure.
  express-indent-function         Indent function for structure.
  express-comment                       Add comment to current line.
  express-comment-indent                Compute indent for comment.
  express-comment-region                To comment a region
  express-line-type                     Tell current line type (for debugging).
  express-indent-type                   Tell last indent type (for debugging).
  express-compile-buffer         Compile all express functions in buffer
  express-compile-function              Compile express function

To add automatic support put something like the following in your .emacs file:
  \(autoload 'express-mode \"express-mode\" \"Enter Express-mode.\" t\)
  \(setq auto-mode-alist \(cons '\(\"\\\\.exp$\" . express-mode\) \
auto-mode-alist\)\)
  \(setq express-mode-hook '\(lambda \(\) \(setq fill-column 74\)\)\)"
  (interactive)
  (kill-all-local-variables)
  (use-local-map express-mode-map)
  (setq major-mode 'express-mode)
  (setq mode-name "Express")
  (setq local-abbrev-table express-mode-abbrev-table)
  (set-syntax-table express-mode-syntax-table)
  (make-local-variable 'paragraph-start)
  (setq paragraph-start (concat "^$\\|" page-delimiter))
  (make-local-variable 'paragraph-separate)
  (setq paragraph-separate paragraph-start)
  (make-local-variable 'paragraph-ignore-fill-prefix)
  (setq paragraph-ignore-fill-prefix t)
  (make-local-variable 'indent-line-function)
  (setq indent-line-function 'express-indent-line)
  (make-local-variable 'comment-start-skip)
  (setq comment-start-skip "--")
  (make-local-variable 'comment-column)
  (setq comment-column 'express-comment-column)
  (make-local-variable 'comment-indent-function)
  (setq comment-indent-function 'express-comment-indent)
  (make-local-variable 'fill-column)
  (setq fill-column default-fill-column)
  (make-local-variable 'indent-tabs-mode)
  (setq indent-tabs-mode t)
  (run-hooks 'express-mode-hook)
  ;; The follwoing two entries are set by RMz for font-lock support:
  (make-local-variable 'font-lock-defaults)
  ;;(setq font-lock-defaults '(express-font-lock-keywords nil t))
  (express-insert-menu-in-XEmacs-menubar)
  )

(defun express-return ()
  "Handle carriage return in Express-mode."
  (interactive)
  (if (exp-block-end-line)
      (express-indent-line))
  (newline)
  (express-indent-line))

(defun express-comment ()
```

```
  "Add a comment to the following line, or format if one already exists."
  (interactive)
  (cond
   ((exp-empty-line)
    (express-indent-line)
    (insert "// "))
   ((exp-comment-line))
   (t
    (end-of-line)
    (re-search-backward "[^ \t^]" 0 t)
    (forward-char)
    (delete-horizontal-space)
    (if (< (current-column) express-comment-column)
        (indent-to express-comment-column)
      (insert " "))
    (insert "// "))))

(defun express-comment-indent ()
  "Indent a comment line in Express-mode."
  (exp-calc-indent))

(defun express-indent-line ()
  "Indent a line in Express-mode."
  (interactive)
  (save-excursion
    (beginning-of-line)
    (delete-horizontal-space)
    (indent-to  (exp-calc-indent)))
  (skip-chars-forward "[ \t]"))

(defun express-line-type ()
  "Display type of current line.  Used in debugging."
  (interactive)
  (cond
   ((exp-empty-line)
    (message "express-line-type: empty-line"))
   ((exp-comment-line)
    (message "express-line-type: comment-line"))
   ((exp-continuation-line)
    (message "express-line-type: continuation-line"))
   ((exp-block-beg-end-line)
    (message "express-line-type: block-beg-end-line"))
   ((exp-block-beg-line)
    (message "express-line-type: block-beg-line"))
   ((exp-block-end-line)
    (message "express-line-type: block-end-line"))
   (t
    (message "express-line-type: other"))))

(defun express-indent-type ()
  "Display type of current or previous nonempty line.  Used in debugging."
  (interactive)
  (message (concat "express-ident-type: " exp-last-indent-type)))

(defun express-fill-region (from to &optional justify-flag)
  "Fill the region of comments.
Prefix arg (non-nil third arg, if called from program)
means justify as well."
  (interactive)
  (message "express-fill-region not implemented yet."))


(defun exp-calc-indent ()
  "Return the appropriate indentation for this line as an int."
  (interactive)
  (let ((indent 0))
    (save-excursion
      (forward-line -1)                 ; compute indent based on previous
      (if (exp-empty-line) ;   non-empty line
          (re-search-backward "[^ \t\n]" 0 t))
      (cond
       ((exp-empty-line)
        (setq exp-last-indent-type "empty"))
       ((exp-comment-line)
        (setq exp-last-indent-type "comment"))
       ((exp-unbal-matexp-line)
        (setq exp-last-indent-type "unbalanced-matrix-expression")
        (setq indent (exp-calc-matexp-indent))
(if (exp-continuation-line)
    (progn (setq exp-last-indent-type "continuation")
   (setq indent  (+ indent (* 2 express-indent-level))))))
       ((exp-continuation-line)
        (setq exp-last-indent-type "continuation")
        (setq indent (* 2 express-indent-level))
(message "exp-continuation-line"))
       ((exp-block-beg-end-line)
```

```
(message "exp-block-beg-end-line")
        (setq exp-last-indent-type "block begin-end"))
       ((exp-block-beg-line)
(message "exp-block-beg-line")
        (setq exp-last-indent-type "block begin")
(setq indent express-indent-level))
       (t
        (setq exp-last-indent-type "other")))
      (setq indent (+ indent (current-indentation)))
      (if (= 0 (forward-line -1))
          (if (exp-continuation-line)
              (setq indent (- indent (* 2 express-indent-level))))))
    (if (and (exp-block-end-line) (not (exp-block-beg-end-line)))
(setq indent (- indent express-indent-level)))
    (if (< indent 0) (setq indent 0))
    indent))


(defun exp-empty-line ()
  "Returns t if current line is empty."
  (save-excursion
    (beginning-of-line)
    (looking-at "^[ \t]*$")))

(defun exp-comment-line ()
  "Returns t if current line is an Express comment line."
  (save-excursion
    (beginning-of-line)
    (skip-chars-forward " \t")
    (looking-at "//")))

(defun exp-continuation-line ()
  "Returns t if current line ends in ... and optional comment."
  (save-excursion
    (beginning-of-line)
    (re-search-forward "\\.\\.+[ \t]*\\(//.*\\)?$" (exp-eoln-point) t)))

(defun exp-eoln-point ()
  "Returns point for end-of-line in Express-mode."
  (save-excursion
    (end-of-line)
    (point)))

(defun exp-bol-point()
  "Returns point for beginnind-of-line in Express-mode."
  (save-excursion
    (beginning-of-line)
    (point)))


(defun exp-block-beg-line ()
  "Returns t if line contains beginning of Express block."
  (save-excursion
    (beginning-of-line)
    (and (re-search-forward
  (concat "\\([ ,;\t]\\)?" exp-block-beg-kw) (exp-eoln-point) t )
 (not (express-is-in-string))
 (not (express-is-in-comment))
 )
    ))


(defun exp-block-end-line ()
  "Returns t if line contains end of Express block."
  (save-excursion
    (beginning-of-line)
    (and (re-search-forward
  (concat "\\([ \t]*\\)" exp-block-end-kw) (exp-eoln-point) t )
 (not (express-is-in-string))
 (not (express-is-in-comment)))
    ))


(defun exp-block-beg-end-line ()
  "Returns t if line contains matching block begin-end in Express-mode."
  (save-excursion
    (beginning-of-line)
    (and (re-search-forward
  (concat
   "\\([ ,;\t]\\)?" exp-block-beg-kw
   ".*" "\\([ ,;\t]\\)?" exp-block-end-kw)(exp-eoln-point) t )
 (not (express-is-in-string))
 (not (express-is-in-comment))
 )
    ))
(defun exp-unbal-matexp-line ()
```

```
  (if (= (exp-calc-matexp-indent) 0)
      ()
    t))

(defun exp-calc-matexp-indent ()
  (let ((indent 0))
    (save-excursion
      (beginning-of-line)
      (while (< (point) (exp-eoln-point))
(cond
 ((looking-at "\\[")
  (setq indent (+ indent express-indent-level)))
 ((looking-at "\\]")
  (setq indent (- indent express-indent-level))))
(forward-char)))
    (* 2 indent)))

(defun exp-comment-on-line ()
  "Returns t if current line contains a comment."
  (save-excursion
    (beginning-of-line)
    (looking-at "[^\n]*//")))

(defun exp-in-comment ()
  "Returns t if point is in a comment."
  (save-excursion
    (and (/= (point) (point-max)) (forward-char))
    (search-backward "//" (save-excursion (beginning-of-line) (point)) t)))


(defun express-comment-region (beg-region end-region arg)
  "Comments every line in the region.
Puts express-comment-region at the beginning of every line in the region.
BEG-REGION and END-REGION are args which specify the region boundaries.
With non-nil ARG, uncomments the region."
  (interactive "*r\nP")
  (let ((end-region-mark (make-marker)) (save-point (point-marker)))
    (set-marker end-region-mark end-region)
    (goto-char beg-region)
    (beginning-of-line)
    (if (not arg) ;comment the region
(progn (insert express-comment-region)
       (while (and  (= (forward-line 1) 0)
    (< (point) end-region-mark))
 (insert express-comment-region)))
      (let ((com (regexp-quote express-comment-region))) ;uncomment the region
(if (looking-at com)
    (delete-region (point) (match-end 0)))
(while (and  (= (forward-line 1) 0)
     (< (point) end-region-mark))
  (if (looking-at com)
      (delete-region (point) (match-end 0))))))
    (goto-char save-point)
    (set-marker end-region-mark nil)
    (set-marker save-point nil)))

(defun beginning-of-express-function ()
  "Moves point to the beginning of the current Express function."
  (interactive)
  (let ((case-fold-search t))
    (beginning-of-line -1)
    (re-search-backward "^[ \t]*function" nil 'move)))

(defun end-of-express-function ()
  "Moves point to the end of the current Express subprogram."
  (interactive)
  (let ((case-fold-search t))
    (beginning-of-line 2)
    (re-search-forward "^[ \t]*function" nil 'move)
    (if (eobp)
(end-of-buffer)
      (goto-char (match-beginning 0))
      (forward-line -1))))


(defun mark-express-function ()
  "Put mark at end of Express function, point at beginning.
The marks are pushed."
  (interactive)
  (push-mark (point) nil t)
  (end-of-express-function)
  (push-mark (point) nil t)
  (beginning-of-express-function))

(defun express-indent-function ()
  "Properly indents the Express function  which contains point."
```

```
  (interactive)
  (save-excursion
    (mark-express-function)
    (message "Indenting function...")
    (indent-region (point) (mark) nil))
  (message "Indenting function...done."))

(defun express-indent-region ()
  "Properly indents a Express region."
  (interactive)
  (save-excursion
    (message "Indenting region...")
    (indent-region (mark) (point) nil))
  (message "Indenting region...done."))

(defun express-indent-buffer ()
  "Properly indents a Express buffer."
  (interactive)
  (save-excursion
    (message "Indenting buffer...")
    (indent-region (point-min) (point-max) nil))
  (message "Indenting buffer...done."))

(defun express-is-in-comment()
  "Return non-nil if POS (a buffer position) is inside a Express comment,
nil else."
  (interactive)
  (save-excursion
    (re-search-backward  "--" (exp-bol-point) t)))

(defun express-is-in-string()
  "Return non-nil if POS (a buffer position) is inside a Express string,
nil else."
  (interactive)
  (save-excursion
    (setq where (point))
    (goto-char where)
    (cond
     ((bolp) nil) ; bol is never inside a string
     ((save-excursion ; comment lines too
(beginning-of-line)(looking-at "--")) nil)
     (t (let (;; ok, serious now. Init some local vars:
      (parse-state '(0 nil nil nil nil nil 0))
      (not-done t)
      parse-limit
      end-of-line
      )
  ;; move to start of current statement
  (beginning-of-line)
  ;; now parse up to WHERE
  (while not-done
    (if (or;; skip to next line if:
 ;; - comment line?
 ;;(looking-at comment-line-start-skip)
 ;; - at end of line?
 (eolp)
 ;; - not in a string and after comment-start?
 ;;(and (not (nth 3 parse-state))
 ;;     comment-start
 ;;     (equal comment-start
 ;;     (char-to-string (preceding-char))))
 )
;; get around a bug in forward-line in versions <= 18.57
(if (or (> (forward-line 1) 0) (eobp))
    (setq not-done nil))
      ;; else:

      ;; find out parse-limit from here
      (setq end-of-line (save-excursion (end-of-line)(point)))
      (setq parse-limit (min where end-of-line))
      ;; parse max up to comment-start, if non-nil and in current line
      ;;(if comment-start
      ;;  (save-excursion
      ;;    (if (re-search-forward quoted-comment-start end-of-line t)
      ;; (setq parse-limit (min (point) parse-limit)))))
      ;; now parse if still in limits
      (if (< (point) where)
  (setq parse-state (parse-partial-sexp
     (point) parse-limit nil nil parse-state))
(setq not-done nil))
      ))
  ;; result is
  (if (nth 3 parse-state)
      t
    ()
    )
```

```
  )))))

;; La suite concerne la compilation des functions
;;-----------------------------------------

(defun express-region (beg end)
  "Run Express on the current region, via a temporary file.
The file's name comes from the variable 'express-zap-file' and the
variable 'express-directory' says where to put it.

If the buffer has a header, the header is given to Express before the
region itself.  The buffer's header is all lines between the strings
defined by 'express-start-of-header' and 'express-end-of-header' inclusive.
The header must start in the first 100 lines of the buffer.

The value of 'express-trailer' is given to Express as input after the region.

The value of 'express-command' specifies the command to use to run Express."
  (interactive "r")
  (if (express-shell-running)
      (express-kill-job)
    (express-start-shell))
  (display-buffer (process-buffer (get-process "express-shell")))
  (or
   (setq express-zap-file (express-generate-zap-file-name)))

  (let* ((temp-buffer (get-buffer-create " Express-Output-Buffer"))
; Temp file will be written and Express will be run in zap-directory.
; If the TEXINPUTS file has relative directories or if the region has
; \input of files, this must be the same directory as the file for
; Express to access the correct inputs.  That's why it's safest if
; express-directory is ".".
         (zap-directory
          (file-name-as-directory (expand-file-name express-directory)))
         (express-out-file (concat zap-directory express-zap-file)))

    (express-delete-last-temp-files)
    ;; Write the new temp file.


    ;; compilation
    ;;------------
    ;; Record the file name to be deleted afterwards.
    (setq express-last-temp-file express-out-file)
    (compile-internal (concat express-command " "  buffer-file-name)
      "No more errors" "express-compil"
      nil nil nil)
    (setq express-last-buffer-compiled (current-buffer))))

(defun express-compile-buffer ()
  "Run Express on current buffer.  See \\[express-region] for more information.
Does not save the buffer, so it's useful for trying experimental versions.
See \\[express-file] for an alternative."
  (interactive)
  (express-region (point-min) (point-max)))

(defun express-compile-function ()
  "Run Express on current buffer.  See \\[express-region] for more information.
Does not save the buffer, so it's useful for trying experimental versions.
See \\[express-file] for an alternative."
  (interactive)
  (mark-express-function)
  (express-region (point) (mark)))


(defun express-start-shell ()
  (save-excursion
    (set-buffer
     (make-comint
      "express-shell"
      (or express-shell-file-name (getenv "ESHELL") (getenv "SHELL") "/bin/sh")
      nil))
    (let ((proc (get-process "express-shell")))
      (set-process-sentinel proc 'express-shell-sentinel)
      (process-kill-without-query proc)
      (setq express-shell-map (copy-keymap shell-mode-map))
      ;;(express-define-common-keys express-shell-map)
      (use-local-map express-shell-map)
      (run-hooks 'express-shell-hook)
      (while (zerop (buffer-size))
(sleep-for 1)))))
(defun express-shell-running ()
  (and (get-process "express-shell")
       (eq (process-status (get-process "express-shell")) 'run)))

(defun express-kill-job ()
```

```
  "Kill the currently running TeX job."
  (interactive)
  (quit-process (get-process "express-shell") t))

(defun express-delete-last-temp-files ()
  "Delete any junk files from last temp file."
  (if express-last-temp-file
      (let* ((dir (file-name-directory express-last-temp-file))
             (list (file-name-all-completions
                    (file-name-nondirectory express-last-temp-file) dir)))
        (while list
          (delete-file (concat dir (car list)))
          (setq list (cdr list))))))

(add-hook 'kill-emacs-hook 'express-delete-last-temp-files)
(defun express-generate-zap-file-name ()
  "Generate a unique name suitable for use as a file name."
  ;; Include the shell process number and host name
  ;; in case there are multiple shells (for same or different user).
  (format "#tz%-d%s"
          (process-id (get-buffer-process "*express-shell*"))
  (express-strip-dots (system-name))))

(defun express-strip-dots (s)
  (setq s (copy-sequence s))
  (while (string-match "\\." s)
    (aset s (match-beginning 0) ?-))
  s)


(defun express-set-buffer-directory (buffer directory)
  "Set BUFFER's default directory to be DIRECTORY."
  (setq directory (file-name-as-directory (expand-file-name directory)))
  (if (not (file-directory-p directory))
      (error "%s is not a directory" directory)
    (save-excursion
      (set-buffer buffer)
      (setq default-directory directory))))


;;Fin de la partie dediee a la compilation des functions
;;-----------------------------------------------------
(provide 'express-mode)

;;commande de compilation
;;-----------------------

(defvar express-menu
  (list "Express"
["Compile"           express-compile-function t]
["Indent buffer"     express-indent-buffer t]
)
  "Menu for Express.")

(defun express-insert-menu-in-XEmacs-menubar ()
  "Insert Express menu in the XEmacs menu bar."
  (if (and
       (not (featurep 'infodock))
       (not (memq 'infodock c-emacs-features))
       (boundp 'current-menubar)
       current-menubar)
      (if (fboundp 'add-submenu)
  (add-submenu nil express-menu)
(add-menu nil "Express" (cdr express-menu)))))

;; --- last line of express-mode.el ---
```

Annexe H

Module Express-G pour l'outil GNU Dia

Sheet

```
<?xml version="1.0" encoding="iso-8859-1"?> <!-- -*- xml -*- -->

<sheet>
  <name>ExpressG</name>
  <description xml:lang="no"></description>
  <description xml:lang="fr">Objets pour dessiner des graphes Express-G</description>
  <description xml:lang="de"></description>
  <description xml:lang="en">Objects to draw Express-G diagrams</description>
  <contents>
    <object name="ExpressG - Entity">
      <description xml:lang="no">Entity</description>
      <description xml:lang="fr">Entity</description>
      <description xml:lang="de">Entity</description>
      <description>Entity</description>
    </object>
    <object name="ExpressG - SimpleType">
      <description xml:lang="no">Simple Type</description>
      <description xml:lang="fr">Simple Type</description>
      <description xml:lang="de">Simple Type</description>
      <description>Simple Type</description>
    </object>
    <object name="ExpressG - SelectType">
      <description xml:lang="no">Select Type</description>
      <description xml:lang="fr">Select Type</description>
      <description xml:lang="de">Select Type</description>
      <description>Select Type</description>
    </object>
    <object name="ExpressG - EnumerationType">
      <description xml:lang="no">Enumeration Type</description>
      <description xml:lang="fr">Enumeration Type</description>
      <description xml:lang="de">Enumeration Type</description>
      <description>Enumeration Type</description>
    </object>
    <object name="ExpressG - DefinedType">
      <description xml:lang="no">Defined Type</description>
      <description xml:lang="fr">Defined Type</description>
      <description xml:lang="de">Defined Type</description>
      <description>Defined Type</description>
    </object>
    <object name="ExpressG - Reference">
      <description xml:lang="no">Reference</description>
      <description xml:lang="fr">Reference</description>
      <description xml:lang="de">Reference</description>
      <description>Reference</description>
    </object>
    <object name="ExpressG - Use">
      <description xml:lang="no">Use</description>
      <description xml:lang="fr">Use</description>
      <description xml:lang="de">Use</description>
      <description>Use</description>
    </object>
    <object name="ExpressG - ToPage">
      <description xml:lang="no">To Page "page#, ref# name"</description>
      <description xml:lang="fr">To Page "page#, ref# name"</description>
      <description xml:lang="de">To Page "page#, ref# name"</description>
      <description>To Page "page#, ref# name"</description>
    </object>
    <object name="ExpressG - FromPages">
      <description xml:lang="no">From Pages "page#, ref# (#,#,...)"</description>
      <description xml:lang="fr">From Pages "page#, ref# (#,#,...)"</description>
      <description xml:lang="de">From Pages "page#, ref# (#,#,...)"</description>
      <description>From Pages "page#, ref# (#,#,...)"</description>
    </object>
 </contents>
</sheet>
```

Shapes

ToPage.shape

```
<?xml version="1.0"?>

<shape xmlns="http://www.daa.com.au/~james/dia-shape-ns"
       xmlns:svg="http://www.w3.org/TR/2000/03/WD-SVG-20000303/DTD/svg-20000303-stylable.dtd">
  <name>ExpressG - ToPage</name>
  <description>To Page "page#, ref# name"</description>
  <icon>ToPage.xpm</icon>
  <connections>
    <point x="0"      y="0"/>
    <point x="2"      y="0"/>
    <point x="4"      y="0"/>
    <point x="6"      y="0"/>
    <point x="8"      y="0"/>
    <point x="10"     y="0"/>
    <point x="12"     y="0"/>
    <point x="12"     y="1.25"/>
    <point x="12"     y="2.5"/>
    <point x="12"     y="3.75"/>
    <point x="12"     y="5"/>
    <point x="10"     y="5"/>
    <point x="8"      y="5"/>
    <point x="6"      y="5"/>
    <point x="4"      y="5"/>
    <point x="2"      y="5"/>
    <point x="0"      y="5"/>
    <point x="0"      y="3.75"/>
    <point x="0"      y="2.5"/>
    <point x="0"      y="1.25"/>
  </connections>
  <textbox x1="0" y1="0" x2="12" y2="5"/>
  <svg:svg width="12" height="5">
    <svg:ellipse cx="6" cy="2.5" rx="6" ry="2.5"/>
  </svg:svg>
</shape>
```

DefinedType.shape

```
<?xml version="1.0"?>

<shape xmlns="http://www.daa.com.au/~james/dia-shape-ns"
       xmlns:svg="http://www.w3.org/TR/2000/03/WD-SVG-20000303/DTD/svg-20000303-stylable.dtd">
  <name>ExpressG - DefinedType</name>
  <description>Defined Type</description>
  <icon>DefinedType.xpm</icon>
  <connections>
    <point x="0"      y="0"/>
    <point x="2"      y="0"/>
    <point x="4"      y="0"/>
    <point x="6"      y="0"/>
    <point x="8"      y="0"/>
    <point x="10"     y="0"/>
    <point x="12"     y="0"/>
    <point x="12"     y="1.25"/>
    <point x="12"     y="2.5"/>
    <point x="12"     y="3.75"/>
    <point x="12"     y="5"/>
    <point x="10"     y="5"/>
    <point x="8"      y="5"/>
    <point x="6"      y="5"/>
    <point x="4"      y="5"/>
    <point x="2"      y="5"/>
    <point x="0"      y="5"/>
    <point x="0"      y="3.75"/>
    <point x="0"      y="2.5"/>
    <point x="0"      y="1.25"/>
  </connections>
  <textbox x1="0" y1="0" x2="12" y2="5"/>
  <svg:svg width="12" height="5">
    <svg:line style="stroke-pattern:dashed; stroke-dashlength: 0.3" x1="0" y1="0" x2="12" y2="0"/>
    <svg:line style="stroke-pattern:dashed; stroke-dashlength: 0.3" x1="12" y1="0" x2="12" y2="5"/>
    <svg:line style="stroke-pattern:dashed; stroke-dashlength: 0.3" x1="12" y1="5" x2="0" y2="5"/>
    <svg:line style="stroke-pattern:dashed; stroke-dashlength: 0.3" x1="0" y1="5" x2="0" y2="0"/>
  </svg:svg>
</shape>
```

Entity.shape

```
<?xml version="1.0"?>

<shape xmlns="http://www.daa.com.au/~james/dia-shape-ns"
       xmlns:svg="http://www.w3.org/TR/2000/03/WD-SVG-20000303/DTD/svg-20000303-stylable.dtd">
```

```
  <name>ExpressG - Entity</name>
  <description>Entity</description>
  <icon>Entity.xpm</icon>
  <connections>
    <point x="0"      y="0"/>
    <point x="2"      y="0"/>
    <point x="4"      y="0"/>
    <point x="6"      y="0"/>
    <point x="8"      y="0"/>
    <point x="10"     y="0"/>
    <point x="12"     y="0"/>
    <point x="12"     y="1.25"/>
    <point x="12"     y="2.5"/>
    <point x="12"     y="3.75"/>
    <point x="12"     y="5"/>
    <point x="10"     y="5"/>
    <point x="8"      y="5"/>
    <point x="6"      y="5"/>
    <point x="4"      y="5"/>
    <point x="2"      y="5"/>
    <point x="0"      y="5"/>
    <point x="0"      y="3.75"/>
    <point x="0"      y="2.5"/>
    <point x="0"      y="1.25"/>
  </connections>
  <textbox x1="0" y1="0" x2="12" y2="5"/>
  <svg:svg width="12" height="5">
    <svg:line x1="0" y1="0" x2="12" y2="0"/>
    <svg:line x1="12" y1="0" x2="12" y2="5"/>
    <svg:line x1="12" y1="5" x2="0" y2="5"/>
    <svg:line x1="0" y1="5" x2="0" y2="0"/>
  </svg:svg>
</shape>
```

EnumerationType.shape

```
<?xml version="1.0"?>

<shape xmlns="http://www.daa.com.au/~james/dia-shape-ns"
       xmlns:svg="http://www.w3.org/TR/2000/03/WD-SVG-20000303/DTD/svg-20000303-stylable.dtd">
  <name>ExpressG - EnumerationType</name>
  <description>Enumeration Type</description>
  <icon>EnumerationType.xpm</icon>
  <connections>
    <point x="0"      y="0"/>
    <point x="2"      y="0"/>
    <point x="4"      y="0"/>
    <point x="6"      y="0"/>
    <point x="8"      y="0"/>
    <point x="10"     y="0"/>
    <point x="12"     y="0"/>
    <point x="12"     y="1.25"/>
    <point x="12"     y="2.5"/>
    <point x="12"     y="3.75"/>
    <point x="12"     y="5"/>
    <point x="10"     y="5"/>
    <point x="8"      y="5"/>
    <point x="6"      y="5"/>
    <point x="4"      y="5"/>
    <point x="2"      y="5"/>
    <point x="0"      y="5"/>
    <point x="0"      y="3.75"/>
    <point x="0"      y="2.5"/>
    <point x="0"      y="1.25"/>
  </connections>
  <textbox x1="0" y1="0" x2="10" y2="5"/>
  <svg:svg width="12" height="5">
    <svg:line style="stroke-pattern:dashed; stroke-dashlength: 0.3" x1="0" y1="0" x2="12" y2="0"/>
    <svg:line style="stroke-pattern:dashed; stroke-dashlength: 0.3" x1="12" y1="0" x2="12" y2="5"/>
    <svg:line style="stroke-pattern:dashed; stroke-dashlength: 0.3" x1="12" y1="5" x2="0" y2="5"/>
    <svg:line style="stroke-pattern:dashed; stroke-dashlength: 0.3" x1="0" y1="5" x2="0" y2="0"/>
    <svg:line style="stroke-pattern:dashed; stroke-dashlength: 0.3" x1="10" y1="0" x2="10" y2="5"/>
  </svg:svg>
</shape>
```

FromPages.shape

```
<?xml version="1.0"?>

<shape xmlns="http://www.daa.com.au/~james/dia-shape-ns"
       xmlns:svg="http://www.w3.org/TR/2000/03/WD-SVG-20000303/DTD/svg-20000303-stylable.dtd">
  <name>ExpressG - FromPages</name>
  <description>From Pages "page#, ref# (#,#,...)"</description>
  <icon>FromPages.xpm</icon>
  <connections>
```

```
    <point x="0"      y="0"/>
    <point x="2"      y="0"/>
    <point x="4"      y="0"/>
    <point x="6"      y="0"/>
    <point x="8"      y="0"/>
    <point x="10"     y="0"/>
    <point x="12"     y="0"/>
    <point x="12"     y="1.25"/>
    <point x="12"     y="2.5"/>
    <point x="12"     y="3.75"/>
    <point x="12"     y="5"/>
    <point x="10"     y="5"/>
    <point x="8"      y="5"/>
    <point x="6"      y="5"/>
    <point x="4"      y="5"/>
    <point x="2"      y="5"/>
    <point x="0"      y="5"/>
    <point x="0"      y="3.75"/>
    <point x="0"      y="2.5"/>
    <point x="0"      y="1.25"/>
  </connections>
  <textbox x1="0" y1="0" x2="12" y2="5"/>
  <svg:svg width="12" height="5">
    <svg:ellipse cx="6" cy="2.5" rx="6" ry="2.5"/>
  </svg:svg>
</shape>
```

Reference.shape

```
<?xml version="1.0"?>

<shape xmlns="http://www.daa.com.au/~james/dia-shape-ns"
       xmlns:svg="http://www.w3.org/TR/2000/03/WD-SVG-20000303/DTD/svg-20000303-stylable.dtd">
  <name>ExpressG - Reference</name>
  <description>Reference</description>
  <icon>Reference.xpm</icon>
  <connections>
    <point x="0"      y="0"/>
    <point x="2"      y="0"/>
    <point x="4"      y="0"/>
    <point x="6"      y="0"/>
    <point x="8"      y="0"/>
    <point x="10"     y="0"/>
    <point x="12"     y="0"/>
    <point x="12"     y="1.25"/>
    <point x="12"     y="2.5"/>
    <point x="12"     y="3.75"/>
    <point x="12"     y="5"/>
    <point x="10"     y="5"/>
    <point x="8"      y="5"/>
    <point x="6"      y="5"/>
    <point x="4"      y="5"/>
    <point x="2"      y="5"/>
    <point x="0"      y="5"/>
    <point x="0"      y="3.75"/>
    <point x="0"      y="2.5"/>
    <point x="0"      y="1.25"/>
  </connections>
  <textbox x1="0" y1="0" x2="12" y2="5"/>
  <svg:svg width="12" height="5">
    <svg:line style="stroke-pattern:dashed; stroke-dashlength: 0.3" x1="0" y1="0" x2="12" y2="0"/>
    <svg:line style="stroke-pattern:dashed; stroke-dashlength: 0.3" x1="12" y1="0" x2="12" y2="5"/>
    <svg:line style="stroke-pattern:dashed; stroke-dashlength: 0.3" x1="12" y1="5" x2="0" y2="5"/>
    <svg:line style="stroke-pattern:dashed; stroke-dashlength: 0.3" x1="0" y1="5" x2="0" y2="0"/>
    <svg:ellipse cx="6" cy="2.5" rx="6" ry="2.5"/>

  </svg:svg>
</shape>
```

SelectType.shape

```
<?xml version="1.0"?>

<shape xmlns="http://www.daa.com.au/~james/dia-shape-ns"
       xmlns:svg="http://www.w3.org/TR/2000/03/WD-SVG-20000303/DTD/svg-20000303-stylable.dtd">
  <name>ExpressG - SelectType</name>
  <description>Select Type</description>
  <icon>SelectType.xpm</icon>
  <connections>
    <point x="0"      y="0"/>
    <point x="2"      y="0"/>
    <point x="4"      y="0"/>
    <point x="6"      y="0"/>
    <point x="8"      y="0"/>
    <point x="10"     y="0"/>
```

```
    <point x="12"     y="0"/>
    <point x="12"     y="1.25"/>
    <point x="12"     y="2.5"/>
    <point x="12"     y="3.75"/>
    <point x="12"     y="5"/>
    <point x="10"     y="5"/>
    <point x="8"      y="5"/>
    <point x="6"      y="5"/>
    <point x="4"      y="5"/>
    <point x="2"      y="5"/>
    <point x="0"      y="5"/>
    <point x="0"      y="3.75"/>
    <point x="0"      y="2.5"/>
    <point x="0"      y="1.25"/>
  </connections>
  <textbox x1="2" y1="0" x2="12" y2="5"/>
  <svg:svg width="12" height="5">
    <svg:line style="stroke-pattern:dashed; stroke-dashlength: 0.3" x1="0" y1="0" x2="12" y2="0"/>
    <svg:line style="stroke-pattern:dashed; stroke-dashlength: 0.3" x1="12" y1="0" x2="12" y2="5"/>
    <svg:line style="stroke-pattern:dashed; stroke-dashlength: 0.3" x1="12" y1="5" x2="0" y2="5"/>
    <svg:line style="stroke-pattern:dashed; stroke-dashlength: 0.3" x1="0" y1="5" x2="0" y2="0"/>
    <svg:line style="stroke-pattern:dashed; stroke-dashlength: 0.3" x1="2" y1="0" x2="2" y2="5"/>
  </svg:svg>
</shape>
```

SimpleType.shape

```
<?xml version="1.0"?>

<shape xmlns="http://www.daa.com.au/~james/dia-shape-ns"
       xmlns:svg="http://www.w3.org/TR/2000/03/WD-SVG-20000303/DTD/svg-20000303-stylable.dtd">
  <name>ExpressG - SimpleType</name>
  <description>Simple Type</description>
  <icon>SimpleType.xpm</icon>
  <connections>
    <point x="0"      y="0"/>
    <point x="2"      y="0"/>
    <point x="4"      y="0"/>
    <point x="6"      y="0"/>
    <point x="8"      y="0"/>
    <point x="10"     y="0"/>
    <point x="12"     y="0"/>
    <point x="12"     y="1.25"/>
    <point x="12"     y="2.5"/>
    <point x="12"     y="3.75"/>
    <point x="12"     y="5"/>
    <point x="10"     y="5"/>
    <point x="8"      y="5"/>
    <point x="6"      y="5"/>
    <point x="4"      y="5"/>
    <point x="2"      y="5"/>
    <point x="0"      y="5"/>
    <point x="0"      y="3.75"/>
    <point x="0"      y="2.5"/>
    <point x="0"      y="1.25"/>
  </connections>
  <textbox x1="0" y1="0" x2="10" y2="5"/>
  <svg:svg width="12" height="5">
    <svg:line x1="0" y1="0" x2="12" y2="0"/>
    <svg:line x1="12" y1="0" x2="12" y2="5"/>
    <svg:line x1="12" y1="5" x2="0" y2="5"/>
    <svg:line x1="0" y1="5" x2="0" y2="0"/>
    <svg:line x1="10" y1="0" x2="10" y2="5"/>
  </svg:svg>
</shape>
```

Use.shape

```
<?xml version="1.0"?>

<shape xmlns="http://www.daa.com.au/~james/dia-shape-ns"
       xmlns:svg="http://www.w3.org/TR/2000/03/WD-SVG-20000303/DTD/svg-20000303-stylable.dtd">
  <name>ExpressG - Use</name>
  <description>Use</description>
  <icon>Use.xpm</icon>
  <connections>
    <point x="0"      y="0"/>
    <point x="2"      y="0"/>
    <point x="4"      y="0"/>
    <point x="6"      y="0"/>
    <point x="8"      y="0"/>
    <point x="10"     y="0"/>
    <point x="12"     y="0"/>
    <point x="12"     y="1.25"/>
    <point x="12"     y="2.5"/>
```

```
    <point x="12"     y="3.75"/>
    <point x="12"     y="5"/>
    <point x="10"     y="5"/>
    <point x="8"      y="5"/>
    <point x="6"      y="5"/>
    <point x="4"      y="5"/>
    <point x="2"      y="5"/>
    <point x="0"      y="5"/>
    <point x="0"      y="3.75"/>
    <point x="0"      y="2.5"/>
    <point x="0"      y="1.25"/>
  </connections>
  <textbox x1="0" y1="0" x2="12" y2="5"/>
  <svg:svg width="12" height="5">
    <svg:line  x1="0" y1="0" x2="12" y2="0"/>
    <svg:line  x1="12" y1="0" x2="12" y2="5"/>
    <svg:line  x1="12" y1="5" x2="0" y2="5"/>
    <svg:line  x1="0" y1="5" x2="0" y2="0"/>
    <svg:ellipse cx="6" cy="2.5" rx="6" ry="2.5"/>

  </svg:svg>
</shape>
```

Annexe I

ARM schéma stepmsa_schema

```
(*============================================
  =                                          =
  =                Materials                 =
  =                                          =
  ============================================
*)

ENTITY math_scalar
SUPERTYPE OF (ONEOF(real_scalar, complex_scalar));
END_ENTITY;

ENTITY real_scalar
SUBTYPE OF (math_scalar);
   value : REAL;
END_ENTITY;

ENTITY complex_scalar
SUBTYPE OF (math_scalar);
   rvalue : REAL;
   ivalue : REAL;
 END_ENTITY;


ENTITY fea_material_property_model
SUPERTYPE OF (ONEOF (constant,
                     line,
                     exponential,
                     simple_analytical_saturation,
                     adjustable_analytical_saturation,
                     spline_saturation,
                     parabola_plus_line,
                     constant_plus_gaussian,
                     exponential_plus_gaussian,
                     constant_mul_exponential,
                     simple_analytical_saturation_mul_exponential,
                     adjustable_analytical_saturation_mul_exponential,
                     node_values,
                     user_defined));
   name : STRING;
```

```
END_ENTITY;

ENTITY constant_model
SUBTYPE OF (fea_material_property_model);
    value : math_scalar;
END_ENTITY;

ENTITY line_model
SUBTYPE OF (fea_material_property_model);
    zero_value : math_scalar;
    a  : math_scalar;
END_ENTITY;

ENTITY exponential_model
SUBTYPE OF (fea_material_property_model);
   zero_value : math_scalar;
   tau : math_scalar;
   constant : math_scalar;
END_ENTITY;

ENTITY simple_analytical_saturation
SUBTYPE OF (fea_material_property_model);
   sat_value : math_scalar;
   init_value : math_scalar;
   void_constant : math_scalar;
END_ENTITY;

ENTITY adjustable_analytical_saturation
SUBTYPE OF (fea_material_property_model);
   sat_value : math_scalar;
   init_value : math_scalar;
   void_constant : math_scalar;
   a : math_scalar;
END_ENTITY;

ENTITY spline_saturation
SUBTYPE OF (fea_material_property_model);
   v1_values : LIST [0:?] OF math_scalar;
   v2_values : LIST [0:?] OF math_scalar;
   sat_value : math_scalar;
END_ENTITY;

ENTITY parabola_plus_line
SUBTYPE OF (fea_material_property_model);
   init_value : math_scalar;
```

```
   end_value : math_scalar;
   intersection_value : math_scalar;
   void_constant : math_scalar;
END_ENTITY;

ENTITY constant_plus_gaussian
SUBTYPE OF (fea_material_property_model);
   transition_energy : math_scalar;
   curie_temperature : math_scalar;
   sigma_value : math_scalar;
   ro_cp : math_scalar;
 END_ENTITY;

ENTITY exponential_plus_gaussian
SUBTYPE OF (fea_material_property_model);
   transition_energy : math_scalar;
   curie_temperature : math_scalar;
   sigma_value : math_scalar;
   ro_cp0 : math_scalar;
   ro_cpi : math_scalar;
   tau : math_scalar;
END_ENTITY;

ENTITY constant_mul_exponential
SUBTYPE OF (fea_material_property_model);
   curie_temperature : math_scalar;
   mur0 : math_scalar;
   c : math_scalar;
END_ENTITY;

ENTITY simple_analytical_saturation_mul_exponential
SUBTYPE OF (fea_material_property_model);
   sat_value : math_scalar;
   mur0 : math_scalar;
   curie_temperature : math_scalar;
   c : math_scalar;
END_ENTITY;

ENTITY adjustable_analytical_saturation_mul_exponential
SUBTYPE OF (fea_material_property_model);
   sat_value : math_scalar;
   mur0 : math_scalar;
   a : math_scalar;
   curie_temperature : math_scalar;
   c : math_scalar;
END_ENTITY;
```

```
ENTITY node_values
SUBTYPE OF (fea_material_property_model);
   v1 : LIST[0:?] OF  math_scalar;
   v2 :  LIST[0:?] OF  math_scalar;
END_ENTITY;

ENTITY user_defined
SUBTYPE OF (fea_material_property_model);
   id : STRING;
END_ENTITY;

(*=============================================
  =                                           =
  =               Properties                  =
  =                                           =
  =============================================
*)

ENTITY fea_material_magnetic_property_representation_item
SUBTYPE OF (fea_material_property_representation_item)
SUPERTYPE OF (ONEOF (permeability,
                     real_coercitif_magnetic_field,
                     remanent_magnetic_field,
                     reluctivity
                     ));
   model: fea_material_property_model;
END_ENTITY;

ENTITY real_coercitif_magnetic_field
SUBTYPE OF (fea_material_magnetic_property)
END_ENTITY;

ENTITY permeability
SUBTYPE OF (fea_material_magnetic_property) ;
END_ENTITY;

ENTITY remanent_magnetic_field
SUBTYPE OF (fea_material_magnetic_property) ;
END_ENTITY;

ENTITY reluctivity
SUBTYPE OF (fea_material_magnetic_property) ;
END_ENTITY;

(*=============================================
```

```
  =                                                =
  =                Operators                       =
  =                                                =
  ================================================
*)

TYPE derive_operator = ENUMERATION OF
(
 rotational,
 gradient,
 divergence,
 scal_laplace,
 vect_laplace
)
END_TYPE;


(*================================================
  =                                                =
  =                Variables                       =
  =                                                =
  ================================================
*)

ENTITY variable
SUPERTYPE OF (ONEOF(vector_variable,
                    scalar_variable));
END_ENTITY;

ENTITY vector_variable
SUBTYPE OF (variable)
SUPERTYPE OF (ONEOF(magnetic_vector,
                    induction_vector,
                    magnetic_vector_potential,
                    volume_current_density,
                    surface_current_density,
                    electric_vector_potential,
                    derived_vector_variable
                    ));
   dimension : INTEGER;
   values : SET[0:?] OF math_scalar;
END_ENTITY;

ENTITY magnetic_vector
SUBTYPE OF (vector_variable);
END_ENTITY;
```

```
ENTITY induction_vector
SUBTYPE OF (vector_variable);
END_ENTITY;

ENTITY magnetic_vector_potential
SUBTYPE OF (vector_variable);
END_ENTITY;

ENTITY volume_current_density
SUBTYPE OF (vector_variable);
END_ENTITY;

ENTITY surface_current_density
SUBTYPE OF (vector_variable);
END_ENTITY;

ENTITY electric_vector_potential
SUBTYPE OF (vector_variable);
END_ENTITY;

ENTITY derived_vector_variable
SUBTYPE OF (vector_variable);
   source_variable : variable;
   operation_func  : operator;
END_ENTITY;

ENTITY scalar_variable
SUBTYPE OF (variable)
SUPERTYPE OF (ONEOF(magnetic_scalar_potential,
                    reduced_magnetic_scalar_potential,
                    derived_scalar_variable
                    ));
   value : math_scalar;
END_ENTITY;

ENTITY derived_scalar_variable
SUBTYPE OF (scalar_variable);
   source_variable : variable;
   operation_func  : operator;
END_ENTITY;


(*================================================
  =                                              =
  =                Regions                       =
```

```
  =                                                =
  ================================================
*)

ENTITY region;
    material : element_material;
    active_formulation : formulation;
END_ENTITY;

ENTITY region_1d
SUBTYPE OF (region);
END_ENTITY;

ENTITY region_2d
SUBTYPE OF (region);
END_ENTITY;

ENTITY region_3d
SUBTYPE OF (region);
END_ENTITY;

ENTITY region_geometric_relationship;
    region_ref : region;
    item       : SET[0:?] OF geometric_representation_item;
END_ENTITY;

ENTITY region_fea_relationship;
    region_ref : region;
    elements : SET[0:?] OF element_representation;
END_ENTITY;

(*================================================
  =                                              =
  =                Constraints                   =
  =                                              =
  ================================================
*)

ENTITY constraint
SUPERTYPE OF (ONEOF(symmetry,
                    antisymmetry,
                    dirichlet_constraint,
                    neumann_constraint,
                    cyclic_constraint,
                    anticyclic_constraint,
                    translation_constraint));
```

```
   geometric_item : geometric_representation_item;
END_ENTITY;

ENTITY symmetry
SUPERTYPE OF (ONEOF(symmetry_1d, symmetry_2d, symmetry_3d));
END_ENTITY;

ENTITY symmetry_1d
SUBTYPE OF (symmetry);
   center : point;
END_ENTITY;

ENTITY symmetry_2d
SUBTYPE OF (symmetry);
   ref_axe : curve;
END_ENTITY;

ENTITY symmetry_3d
SUBTYPE OF (symmetry);
   ref_surface : surface;
END_ENTITY;

ENTITY antisymmetry
SUPERTYPE OF (ONEOF(antisymmetry_1d, antisymmetry_2d, antisymmetry_3d));
END_ENTITY;

ENTITY antisymmetry_1d
SUBTYPE OF (antisymmetry);
   center : point;
END_ENTITY;

ENTITY antisymmetry_2d
SUBTYPE OF (antisymmetry);
   ref_axe : curve;
END_ENTITY;

ENTITY antisymmetry_3d
SUBTYPE OF (antisymmetry);
   ref_surface : surface;
END_ENTITY;

ENTITY dirichlet_constraint
SUBTYPE OF (constraint);
    value : REAL;
END_ENTITY;
```

```
ENTITY neumann_constraint
SUBTYPE OF (constraint);
END_ENTITY;

ENTITY floating_constraint
SUBTYPE OF (constraint);
END_ENTITY;

ENTITY cyclic_constraint
SUBTYPE OF (constraint);
   ref_frontier : geometric_representation_item;
END_ENTITY;

ENTITY anticyclic_constraint
SUBTYPE OF (constraint);
   ref_frontier : geometric_representation_item;
END_ENTITY;

ENTITY translation_constraint
SUBTYPE OF (constraint);
   value : SET[0:?] OF REAL;
END_ENTITY;

(*================================================
  =                                              =
  =                Formulations                  =
  =                                              =
  ================================================
*)

ENTITY formulation
SUPERTYPE OF (ONEOF(magnetostatic_formulation))
   name : label;
   description : text;
   state_variable : variable;
   validity_domain : maths_space;
   allowed_constraints : SET [0:?] OF constraint;
   accessible_variables : SET [0:?] OF variable;
   needed_material_properties : SET [0:?] OF material_property;
END_ENTITY;

ENTITY magnetostatic_formulation
SUBTYPE OF (formulation);
END_ENTITY;

ENTITY magnetostatic_3d_vector_potential
```

```
SUBTYPE OF (magnetostatic_formulation);
WHERE
  WR1: 'SELF.state_variable'
       IN TYPEOF (MAGNETIC_VECTOR_POTENTIAL));
END_ENTITY;

ENTITY magnetostatic_3d_total_scalar_potential
SUBTYPE OF (magnetostatic_formulation);
END_ENTITY;

ENTITY magnetostatic_3d_reduced_scalar_potential
SUBTYPE OF (magnetostatic_formulation);
END_ENTITY;

ENTITY magnetostatic_3d_coupled_reduced_total_scalar_potential
SUBTYPE OF (magnetostatic_formulation);
END_ENTITY;

ENTITY magnetostatic_3d_axisymmetric_vector_potential
SUBTYPE OF (magnetostatic_formulation);
END_ENTITY;

ENTITY magnetostatic_3d_edge_element_vector_potential_formulation
SUBTYPE OF (magnetostatic_formulation);
END_ENTITY;

ENTITY magnetostatic_3d_edge_element_vector_potential_with_t0
SUBTYPE OF (magnetostatic_formulation);
END_ENTITY;

ENTITY magnetostatic_3d_t0_reduced_scalar_potential_formulation
SUBTYPE OF (magnetostatic_formulation);
END_ENTITY;

ENTITY magnetostatic_3d_coupled_potentials_on_cut
SUBTYPE OF (magnetostatic_formulation);
END_ENTITY;

(*=============================================
  =                                           =
  =                Sources                    =
  =                                           =
  =============================================
*)

ENTITY field_source
```

```
SUPERTYPE OF (ONEOF(magnetic_field_source));
END_ENTITY;

ENTITY magnetic_field_source
SUBTYPE OF (field_source)
SUPERTYPE OF (coil, magnet);
   id : STRING;
END_ENTITY;

ENTITY  current_inductor;
   current_value : REAL;
END_ENTITY;

ENTITY coil
SUBTYPE OF (magnetic_field_source);
   associated_inductor : current_inductor;
   turns_number : INTEGER;
END_ENTITY;

ENTITY  turn_cross_section_coil
SUBTYPE OF (coil)
ABSTRACT SUPERTYPE OF (ONEOF( regular_turn_cross_section,
                        complex_turn_cross_section));
   center  : SET[1:3] OF REAL;
END_ENTITY;

ENTITY  regular_turn_cross_section_coil
SUPERTYPE OF (ONEOF( 1d_circular_turn_cross_section,
                     1d_rectangular_turn_cross_section,
                     2d_circular_turn_cross_section,
                     2d_rectangular_turn_cross_section,
                     3d_circular_turn_cross_section,
                     3d_rectangular_turn_cross_section))
SUBTYPE OF ( turn_cross_section_coil);

END_ENTITY;

ENTITY  1d_circular_turn_cross_section_coil
SUBTYPE OF ( regular_turn_cross_section_coil);
   radius : REAL;
END_ENTITY;

ENTITY  1d_rectangular_turn_cross_section_coil
SUBTYPE OF ( regular_turn_cross_section_coil);
    length : REAL;
    width : REAL;
```

```
    radius : REAL;
END_ENTITY;

ENTITY  2d_circular_turn_cross_section_coil
SUBTYPE OF ( regular_turn_cross_section_coil);
    radius : REAL;
    height : REAL;
END_ENTITY;

ENTITY  2d_rectangular_turn_cross_section_coil
SUBTYPE OF ( regular_turn_cross_section_coil);
    length : REAL;
    width : REAL;
    radius : REAL;
    height : REAL;
END_ENTITY;

ENTITY  3d_circular_turn_cross_section_coil
SUBTYPE OF ( regular_turn_cross_section_coil);
    radius : REAL;
    height : REAL;
    thickness : REAL;
END_ENTITY;

ENTITY  3d_rectangular_turn_cross_section_coil
SUBTYPE OF ( regular_turn_cross_section_coil);
    length : REAL;
    width : REAL;
    radius : REAL;
    height : REAL;
    thickness : REAL;
END_ENTITY;

ENTITY  complex_turn_cross_section_coil
SUPERTYPE OF (ONEOF( saddle_turn_cross_section_coil))
SUBTYPE OF ( turn_cross_section_coil);
END_ENTITY;

ENTITY  saddle_turn_cross_section_coil
SUBTYPE OF ( complex_turn_cross_section_coil);
  phi : REAL;
  psi : REAL;
  length : REAL;
  height : REAL;
  radius : REAL;
  number_of_pole_pairs : INTEGER;
```

```
END_ENTITY;
```