N° d'ordre:

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

U.F.R SCIENCES

THÈSE

En vue de l'obtention du grade de

DOCTEUR DE L'UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Discipline: Informatique

présentée par

Emmanuel MALET

Équipe d'accueil : CMCAO
Composante universitaire : L.E.R.I DE REIMS

Titre de la thèse :

Interopérabilité des systèmes d'IAO : les services de prototypage rapide

Soutenue le 24 Novembre 2003 devant la commission d'examen

Président	Yvon	GARDAN	Professeur à l'université de Reims
Rapporteurs	Michel	LUCAS	Professeur à l'école centrale de Nantes
	Parisa	GHODOUS	MCF à l'université Claude Bernard Lyon I
Examinateurs	Roland	MARANZANA	Professeur à l'ETS de Montréal
	Umberto	BARALDI	Directeur adjoint du CRIF de Liège
Directeur de thèse	Yvon	GARDAN	Professeur à l'université de Reims

Remerciements

Je tiens à remercier,

Monsieur YvonGARDAN, Professeur à l'université de Reims, d'avoir accepté de présider ce jury.

Monsieur Michel LUCAS, Professeur à l'école centrale de Nantes et Madame Parisa GHODOUS, MCF à l'université Claude Bernard Lyon I, pour l'attention qu'ils ont pu porter à mes travaux en acceptant d'en être les rapporteurs.

Monsieur Roland MARANZANA, Professeur à l'ETS de Montréal et Monsieur Umberto BARALDI, Directeur adjoint du CRIF de Liège, d'avoir bien voulu juger ce travail.

Monsieur YvonGARDAN, Professeur à l'université de Reims, de m'avoir accueilli dans son équipe de recherche et permis d'effectuer ce travail dans les meilleures conditions.

Monsieur Roland Maranzana de m'avoir accueilli dans son Laboratoire de l'École de Technologie Supérieure de Montréal et su partager son expertise du domaine de la fabrication.

Je remercie également,

Les collègues doctorants, maîtres de conférence ainsi que le personnel administratif que j'ai pu côtoyer tout au long de cette thèse à l'IFTS de Charleville comme à l'ÉTS de Montréal.

Ma famille et mes proches pour le soutien et l'écoute qu'ils ont su m'apporter dans les moments difficiles.

Emmanuel Malet

Table des matières

Introduction

A l'heure de la mondialisation, nous assistons à la globalisation du marché et de la concurrence. Dans ce contexte, nous assistons à une réorganisation de l'activité des entreprises sous forme de sous-traitance et de délocalisation. En accord avec le modèle de Taylor, le fractionnement et la spécialisation de l'activité globale en activités élémentaires permet l'amélioration de la qualité et de la productivité en rationalisant le processus de développement. La spécialisation et la délocalisation permettent, en effet, d'optimiser les tâches élémentaires et de profiter d'avantages locaux tels que des ressources humaines, matérielles ou encore de certaines législations... Mais si la réussite globale passe par la maîtrise de tâches élémentaires, ceci n'est valable qu'au prix d'une bonne coordination de celles-ci. Ainsi, une activité globale performante passe par la coordination d'activités locales spécialisées et donc par l'interaction et la communication des différents acteurs associés.

Dans le domaine du développement de produit, de nombreux logiciels sont utilisés dans différents services d'ingénierie, de la conception à la fabrication, en passant par la simulation, on parle de systèmes d'IAO (Ingénierie Assistée par Ordinateur). Malheureusement, ces systèmes souffrent d'un manque d'interopérabilité. En effet, il n'est pas encore possible d'intégrer ou de réutiliser, au sein d'un système particulier, des modèles numériques issus de systèmes hétérogènes et éventuellement distants. L'interopérabilité des systèmes d'IAO, longtemps cantonnée à des échanges de données, se heurte, aujourd'hui encore, aux différentes représentations utilisées. Celles-ci ont forcées le développement de traducteurs de formats natifs. Mais le nombre, trop important, des traducteurs nécessaires et la difficulté de leur maintenance à long terme ont forcé la création de formats neutres. Définis dans les années quatre vingts dans divers secteurs industriels et basés sur des représentations pouvant différer d'un système à l'autre, les formats neutres comme IGES, SET, VDA, etc. assurent, tant bien que mal, les échanges de données entre systèmes hétérogènes. Au milieu des années quatre vingt, le projet de normalisation des échanges de données STEP doit permettre à terme les échanges de données entre systèmes hétérogènes.

Afin que le lecteur appréhende précisément la norme STEP, nous présentons, au chapitre I, les principes et le cadre méthodologique sur lesquels il repose. Nous verrons que STEP est une tentative de normalisation des modèles de données du produit utilisés tout au long de son cycle de vie. Au chapitre II, nous présentons l'approche orientée objet de l'interopérabilité. Nous faisons en sorte que le lecteur appréhende les principes orientés objet à travers la présentation de deux architectures majeures du marché, l'architecture de composants logiciels COM et l'architecture d'objets distribués CORBA. Nous verrons qu'en ne réduisant pas l'interopérabilité à des échanges de données, l'approche orienté objet se différencie de l'approche STEP par l'abstraction des représentations et la normalisation des services. Nous évoquerons ensuite les récents travaux de l'OMG, quant à la définition et à la normalisation, sous forme d'interfaces

CORBA, des services des systèmes d'IAO.

Les chapitres I et II présentent précisément au lecteur les deux approches de l'interopérabilité des systèmes d'IAO afin que celui-ci puisse en appréhender les différences fondamentales. Après avoir présenté, dans le chapitre III, le domaine du prototypage rapide et les systèmes informatiques associés, nous proposons au chapitre IV, de considérer l'interopérabilité entre systèmes de CAO et systèmes de prototypage au moyen d'interfaces CORBA. Nous proposons de spécifier les interfaces d'une architecture de composants logiciels orientés prototypage rapide. Celle-ci doit permettre d'assurer une meilleure connectivité entre procédés de prototypage rapide et systèmes de CAO. Au chapitre V, nous présentons une application, disponible sur Internet, implémentant l'architecture spécifiée. Nous présentons tout d'abord les méthodes de tranchage et de remplissage au cœur de l'architecture, puis des services de plus haut niveau tel que la simulation de la rugosité, le positionnement, le calcul de la géométrie tranchée, le calcul des trajectoires d'outil et finalement un service distant d'estimation du coût de fabrication.

Chapitre 1

La norme STEP

1.1 Introduction

Les modèles de données d'IAO (Ingénierie Assistée par Ordinateur) sont constitués d'entités, structurées et reliées entre elles selon une organisation propre à chaque système. L'échange et le partage des données entre systèmes, de fonctionnalités identiques ou spécifiques à chaque corps de métier, pose des problèmes de compatibilité des représentations et donc d'interprétation [1], [2], [3].

De nombreuses normes et interfaces ont été définies (IGES[1], SET[2], VDA[3]). D'un faible niveau d'abstraction, celles-ci se sont heurtées aux évolutions des technologies de l'information et à la complexité des environnements industriels.

Né au milieu des années quatre vingts, le projet de standardisation des échanges des données du produit, STEP (STandard for the Exchange of Product model data) implique des utilisateurs de diverses industries (automobile, aéronautique, pétrochimie, etc.). En 1994, l'architecture est normalisée par l'ISO (International Organization for Standardization) sous la référence ISO-10303 (www.iso.ch).

1.2 Objectif

L'objectif de STEP est de proposer une technologie d'échange et de partage des données du produit indépendamment de toute architecture d'implémentation, qu'elle soit matérielle ou logicielle. Pour cela, STEP adopte une approche d'un haut niveau d'abstraction orientée vers les utilisateurs. Elle définit ce que doit être l'architecture permettant de définir un modèle numérique du produit couvrant tout son cycle de vie, de la conception jusqu'au service après vente en passant par l'analyse, la fabrication, le contrôle qualité. Dans ce but STEP doit supporter des informations générales de natures diverses comme la géométrie, la topologie, le tolérancement, les attributs, les relations, l'assemblage, les configurations etc. mais aussi des informations spécifiques à chaque industrie. Pour ce faire elle définit un langage de modélisation des données, EXPRESS ainsi que des méthodes de mise en oeuvre et de partage.

1. IGES, première norme répondant aux critères de format neutre (1979).
2. SET, issue de l'industrie aérospatiale et normalisée par l'AFNOR (réf. Z68-300) (1983).
3. VDA, issue de l'industrie automobile allemande pour le transfert des surfaces (1983).

1.3 Structure de la norme STEP

Pour accomplir ce but ambitieux, STEP est composé de plusieurs normes ISO qui lui confèrent un bon niveau d'extensibilité. Les composants de base sont complètement définis et publiés, alors que d'autres sont encore en développement. Ces composants sont regroupés dans les catégories suivantes :

— Méthodes de description des données ou DM (Description Methods).
— Ressources communes ou CR (Common Ressources).
— Protocoles d'application ou AP (Application Protocoles).
— Méthodologie de test de conformité CTM (Conformance Testing Methodology).
— Méthodes d'implémentation ou IM (Implementation Method).

Voir annexe A «STEP on a page».

1.4 Méthodes de description

Pour décrire les modèles conceptuels de données du produit, tout au long de son cycle de vie et dans les différents corps de métier, STEP utilise plusieurs méthodes.

— La méthode de modélisation fonctionnelle IDEFØ utilisée pour modéliser les processus de l'entreprise et identifier les besoins informatifs dans un contexte donné.
— Les méthodes de modélisation conceptuelle de données, NIAM, IDEF1X, EXPRESS_G, utilisées pour exprimer la structure conceptuelle de l'information.
— Le langage de modélisation des données EXPRESS utilisé pour exprimer formellement le modèle conceptuel de données.

1.4.1 La méthode NIAM

La problématique de développement d'un système d'information est d'identifier les besoins que doit satisfaire le système et de créer le système répondant à ces besoins. La création d'un système commence au niveau conceptuel en modélisant l'univers du discours UoD (Univers of Discourse) sous forme de schémas conceptuels CS (Conceptual Schema). Les professeurs G.M. Nijssens et E.D. Falkenburg développèrent une méthodologie de conception de schémas conceptuels, la méthode NIAM (Nijssens Information Analysis Method) [4]. C'est sous l'influence du linguiste C.J. Fillmore (1968) qu'E.D. Falkenburg eut l'idée de fonder le concept de schéma conceptuel sur la représentation des phrases élémentaires du langage naturel. Ainsi, NIAM repose sur l'approche relationnelle binaire et consiste à représenter des phrases dans lesquelles deux termes jouent un rôle par leur association binaire [5].

1.4.2 Les méthodes IDEFØ et IDEF1X

Dans les années soixante dix, l'armée américaine avait pour objectif d'améliorer la production industrielle en définissant des méthodes de développement des systèmes d'information. Les laboratoires AFWAL (Air Force Wright Aeronautical Laboratories) développèrent, au sein du programme ICAM (Integrated Computer-Aided Manufacturing), un ensemble de méthodes de modélisation nommé IDEF (Integration DEFinition). Le lecteur pourra trouver à la section

1.9.1 la liste exhaustive des méthodes IDEF. Les méthodes utilisées dans le cadre de STEP sont :

— IDEFØ pour la modélisation des fonctions et flux d'information d'un système.

— IDEF1X pour la modélisation de la structure des données d'un système.

Afin que le lecteur situe précisemment le cadre méthodologique de STEP, ces méthodes sont détaillées aux sections 1.9.2 et 1.9.3.

1.4.3 EXPRESS et EXPRESS_G

EXPRESS est un langage de description des schémas conceptuels. Il définit la syntaxe permettant d'exprimer les concepts d'entité et de relation. EXPRESS_G est la forme graphique du formalisme. L'utilisation d'EXPRESS_G n'est pas obligatoire, des méthodes similaires telles que NIAM ou IDEF1X peuvent être utilisées. Aussi nous ne présenterons pas la syntaxe EXPRESS en soi mais ses concepts à travers la méthode IDEF1X.

1.5 Les ressources communes (CR)

Ce sont un ensemble de modèles de données écris en langage EXPRESS et compilés sous forme de bibliothèques regroupées dans les catégories suivantes :

— Ressources intégrées ou IR (Integrated Ressources)

— Constructions interprétées ou AIC (Application Interpreted Constructs)

— Modules d'application ou AM (Application Modules)

1.5.1 Ressources intégrées (IR)

Les ressources intégrées permettent de construire les modèles de données d'un produit dans un domaine d'application donné. STEP distingue deux types de ressources intégrées : les ressources génériques et les ressources d'application.

Ressources génériques

Les ressources génériques sont indépendantes du domaine d'application. Elles représentent des concepts généraux tels que la géométrie et la topologie.

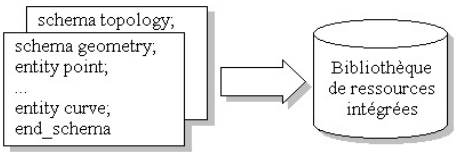


FIGURE 1.1 – *Compilation des ressources intégrées*

Ressources d'application

Les ressources d'application réutilisent et adaptent les ressources génériques pour représenter des concepts métiers. L'adaptation consiste à ajouter des propriétés structurelles (des attributs) ou comportementales (des contraintes).

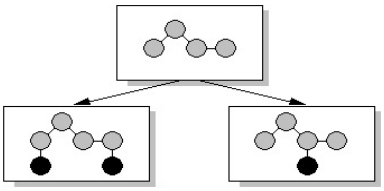


FIGURE 1.2 – *Réutilisation des ressources intégrées*

Exemple : La définition de l'entité géométrique «point» est utilisée pour définir deux nouvelles entités. La première ajoute une propriété structurelle, un attribut «couleur», la deuxième une propriété comportementale, une contrainte sur les coordonnées «$x + y + z = 0$».

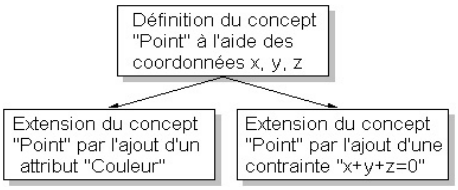


FIGURE 1.3 – *Exemple de spécialisation de l'entité «point»*

Le modèle d'une application donnée utilise donc une partie des ressources intégrées. Après interprétation on obtient le modèle interprété de l'application. Les ressources intégrées sont constituées d'informations assez générales destinées à être réutilisées par des applications issues de domaines industriels différents.

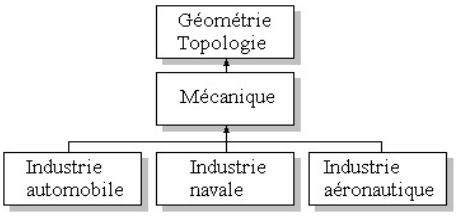


FIGURE 1.4 – *Ressources et modèles interprétés d'application*

1.5.2 Les constructions interprétées d'application (AIC)

Une AIC (Application Interpreted Construction) est une spécification de la structure et de la sémantique des données communes à plusieurs protocoles d'application. Quand plusieurs protocoles d'application ont les mêmes besoins informatifs, la même interprétation des ressources intégrées est utilisée dans chacun de leur modèle interprété AIM (Application Interpreted Model). Cette interprétation est réalisée par inclusion d'un module commun, l'AIC, dans chaque protocole d'application. Une AIC représente les données partagées entre applications de plusieurs industries.

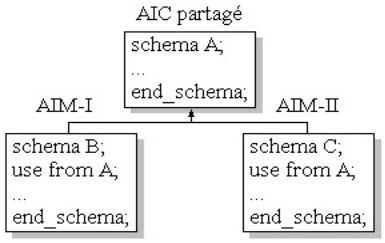


FIGURE 1.5 – *Les AIC dans le développement d'un AIM*

Le but principal d'une AIC est de fournir un mécanisme d'identification et d'encapsulation des besoins communs dans des contextes d'application distincts représentés par leurs modèles de référence respectifs. Cependant, il s'est révélé que les AIC comportaient certaines faiblesses. En effet, une modification d'un protocole d'application portant sur une AIC oblige la modification des protocoles qui la partagent. Le concept d'AIC est en cours de remplacement par le concept de module d'application AM (Application Module) dans lequel une construction peut être référencée directement par un protocole d'application.

1.5.3 Les modules d'application (AM)

Les modules d'application ont été créés pour améliorer, à la fois, l'interopérabilité entre les protocoles d'application et le processus de développement des standards. L'objectif de la modularisation de STEP est d'améliorer son développement, sa normalisation, son implémentation et son déploiement sans en changer les composants fondamentaux.

Un module d'application est une collection réutilisable de déclarations couvrant divers besoins informatifs. Le modèle interprété MIM (Module Interpreted Model) autorise l'usage spécifique des données du produit à travers de multiples contextes d'application.

Un MIM est un modèle d'information qui utilise les ressources communes pour satisfaire à la fois ses besoins informatifs et les contraintes du modèle de référence de l'application à l'intérieur du module d'application.

1.6 Les protocoles d'application (AP)

Un protocole d'application définit le contexte, les besoins informatifs d'une application donnée, ainsi que les méthodes STEP permettant de les exprimer. Le domaine couvert par un protocole est caractérisé par le type de produit, les phases du cycle de vie prises en compte, les types de données et les disciplines les utilisant. Un protocole d'application comporte des règles de conformité servant à valider son implémentation. Chaque domaine d'activité développant une solution d'échange dans l'environnement STEP doit définir et réaliser un protocole d'application.

1.6.1 Développement d'un protocole d'application

Pour développer un protocole d'application, il faut définir trois types de modèles :

— Le modèle d'activité AAM (Application Activity Model) qui modélise les processus à l'aide d' IDEFØ pour en extraire les flux informatifs.
— Le modèle de référence ARM (Application Reference Model) qui modélise les concepts informatifs et leurs relations à l'aide de NIAM, IDEF1X ou EXPRESS_G.
— Le modèle interprété AIM (Application Interpreted Model) qui implémente, en langage EXPRESS, le modèle de référence par spécialisation et ajout de contraintes.

Le processus de développement d'un protocole d'application consiste en la définition du domaine d'application, la modélisation dans le modèle d'activité AAM, des processus du domaine, l'identification des besoins informatifs, leur spécification dans le modèle de référence ARM et leur implémentation dans le modèle interprété AIM. Finalement, chaque protocole doit disposer d'une suite de tests abstraits ATS (Abstract Test Suite) servant à valider leur implémentation.

A noter que les méthodes de spécification utilisées ne sont pas considérées comme définitives et que leur traduction en langage EXPRESS s'effectue, le plus souvent, manuellement.

1.6.2 Développement d'un AIM

Les modèles AIM sont construit par spécialisation des concepts définis dans les bibliothèques de ressources intégrées. On parle de modèles courts et de modèles longs. Les modèles courts contiennent des références aux concepts définis dans les bibliothèques de ressources intégrées. Ils doivent, pour cela, spécifier l'emplacement dans les bibliothèques, des définitions à l'aide de la directive «USE».

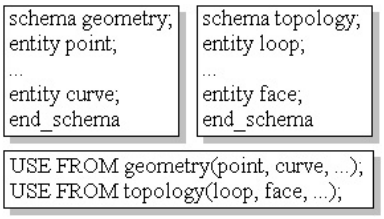


FIGURE 1.6 – *Développement d'un AIM au format «court»*

Comme le spécifie la norme STEP, le format court doit être transformé en format long. Cette transformation consiste à insérer, dans le format court, les définitions EXPRESS des concepts référencés.

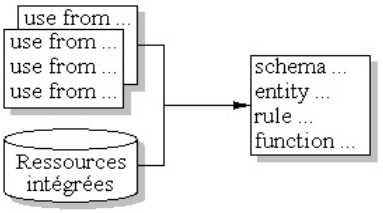


FIGURE 1.7 – *Transformation d'un AIM «court» en AIM «long»*

1.7 Méthodologie de test de conformité (CTM)

Afin de valider son implémentation, un protocole d'application doit posséder une suite de tests abstraits ATS (Abstract Tests Suite). Ces tests consistent à valider le modèle de référence ARM, le modèle interprété AIM et ses méthodes d'implémentation. La conformité du modèle interprété AIM dépend essentiellement d'une bonne utilisation des concepts définis dans les ressources intégrées. Il faut pour cela, que la cohérence sémantique soit préservée, par la réutilisation et la spécialisation des concepts référencés ainsi qu'entre les différentes interprétations que pourrait avoir un même concept dans différents protocoles d'application (cf. AIC). Mais comme nous l'avons déjà évoqué, les AIC ne facilitent pas le processus de développement des protocoles d'application et sont progressivement remplacées par le concept de module d'application. Ainsi, un protocole d'application utilisant des modules d'application ne devrait plus contenir de références à des spécifications de données de terminologie industrielle, de besoins de conformité, mais devrait référencer le module d'application les contenant.

1.8 Les méthodes d'implémentation

L'objectif de la norme STEP est de permettre l'échange et le partage des informations du produit entre systèmes hétérogènes. Pour cela, la norme STEP offre, à l'aide du langage EXPRESS, un mécanisme neutre de description des informations du produit. Elle offre, via les méthodes d'implémentation, un mécanisme neutre de mise en oeuvre, d'accès et de partage de ces informations. Les méthodes d'implémentation regroupent :

— L'interface standard d'accès aux données SDAI (STEP Data Access Interface).
— Le format d'échange de données.

1.8.1 L'interface standard d'accès aux données (SDAI)

Les modèles EXPRESS étant mis en oeuvre de façon spécifique à chaque système (SGBD relationnel ou orienté objet), STEP définit une interface neutre permettant l'instanciation, l'accès et le partage des données indépendamment des architectures matérielles et logicielles.

SDAI est une spécification de l'interface d'une bibliothèque de fonctions pouvant être implémentées à l'aide de différents langages tels que C, C++, Fortran. Les opérations SDAI sont regroupées en différentes catégories. On trouve la catégorie «environnement» pour la création d'une session, la catégorie «session» pour l'ouverture du dépôt de données, la catégorie «dépôt de données» pour la création d'un modèle, la catégorie «modèle» pour la création des instances, les catégories «instances d'entité» et «instances d'application» pour la consultation et la modification des instances d'entités à l'intérieur d'applications.

1.8.2 Le format d'échange de données

Le format d'échange de données STEP constitue une partie de la norme. Il décrit la syntaxe des fichiers de données relatives à un modèle EXPRESS ainsi qu'un ensemble de règles à appliquer pour valider l'association entre les données et le modèle.

Les modèles EXPRESS contenus dans les bibliothèques de ressources intégrées représentent des entités génériques ou métiers suivant le protocole d'application considéré. Les fichiers

d'échange de données STEP contiennent des instances formelles de ces entités. Ainsi, tout système implémentant le même protocole d'application implémente les mêmes entités et est donc capable de les instancier.

Un fichier d'échange STEP se compose d'une section d'en-tête contenant des informations sur l'origine du fichier, sa date de création, sa version et surtout les noms des schémas EXPRESS associés aux données et d'une section de données contenant les instances à échanger.

```
iso-10303-21;
header;
 file_description('fichier d'échange STEP');
 file_name('exemple',('E.Malet','19-07-2002'));
 file_schema('Geometry','Topology');
endsec;
data
 #01=Point_3D(1.25,1.50,1.75);
 #02=Point_3D(2.00,2.25,2.50);
endsec;
end-iso-10303-21;
```

FIGURE 1.8 – *Structure d'un fichier d'échange STEP*

La section d'en-tête contient des chaînes de caractères ou des listes de chaînes de caractères. C'est une partie informative sur les données produites par une application donnée. La section des données constitue l'ensemble des instances à échanger. Une instance est distinguée par son identificateur défini par le caractère # suivi d'un nombre entier. La propriété d'unicité des instances est assurée par la non répétition dans la numérotation. Un fichier d'échange peut contenir jusqu'à un milliard d'instances. La norme n'impose aucune contrainte sur l'ordre d'apparition des instances, une instance peut donc être référencée avant ou après sa définition.

1.9 Cadre méthodologique de STEP

Nous avons évoqué brièvement les méthodes utilisées dans le cadre de STEP. IDEFØ pour le recueil des besoins informatifs, EXPRESS_G, NIAM ou IDEF1X, pour la structuration de l'information. Afin que le lecteur situe précisément le cadre méthodologique de STEP nous présentons les méthodes IDEFØ et IDEF1X.

1.9.1 Les méthodes IDEF

Durant les années soixante dix, le programme ICAM mené par les laboratoires AFWAL a pour but d'accroître la productivité de fabrication à travers une application systématique des technologies informatiques. Le programme ICAM identifia les besoins, d'une meilleure analyse et de l'usage de techniques de communication pour les personnes impliquées dans l'amélioration de la productivité. Dans ce cadre fut développée une série de normes connues sous le nom de techniques IDEF.

— IDEFØ appelée aussi FMM (Function Modeling Method) pour la modélisation des fonctions et flux d'information d'un système [6].

— IDEF1 appelée aussi IMM (Information Modeling Method) pour la modélisation de la structure de l'information d'un système [7].
— IDEF1X appelée aussi DMM (Data Modeling Method) pour la modélisation de la structure des données d'un système [8].
— IDEF2 appelée aussi SMD (Simulation Modeling Method) pour la modélisation de la dynamique des systèmes.

En 1983, le programme IISS (Integrated Information Support System) étend la technique IDEF1 à la modélisation de la sémantique des données pour former la méthode IDEF1X.

Développées afin de soutenir les efforts de modélisation pour un large éventail d'entreprises et de domaines d'application, les techniques IDEFØ et IDEF1X sont aujourd'hui largement utilisées dans les secteurs industriels, gouvernementaux et commerciaux.

En 1991, l'institut national des standards et des technologies NIST (National Institute of Standards and Technology) reçut le soutien, du bureau de gestion de l'information industrielle du département de la défense américaine, le DoD/CIM (U.S. Department of Defence Office, Corporate Information Management), de développer, pour ses techniques de modélisation, une ou plusieurs normes fédérales de traitement de l'information, FIPS (Federal Information Processing Standards). Aujourd'hui, seules IDEFØ, IDEF1 et IDEF1X ont été normalisées. Depuis, plusieurs propositions de méthodes IDEF ont été faites et certaines restent en cours de développement.

— IDEF3 appelée aussi PDCM (Process Description Capture Method) pour la modélisation des processus d'un système [9].
— IDEF4 appelée aussi OODM (Oriented Object Design Method) pour la modélisation orientée objet [10].
— IDEF5 appelée aussi ODCM (Ontology Description Capture Method) pour la description des concepts ontologiques du domaine [11].
— IDEF6 appelée aussi DRCM (Design Rational Capture Method).
— IDEF7 appelée aussi ISAM (Information System Audit Method) pour la modélisation des audits des systèmes d'information.
— IDEF8 appelée aussi UIMM (User Interface Modeling Method) pour la conception des interfaces utilisateurs.
— IDEF9 appelée aussi SISDM (Scenario-driven Information System Design Method) pour la conception, dirigée par les scénarios, des systèmes d'information [12].
— IDEF10 appelée aussi IAMM (Implementation Architecture Modeling Method) pour la modélisation d'architectures d'implémentation.
— IDEF11 appelée aussi IAMM (Information Artifact Modeling Method).
— IDEF12 appelée aussi OMM (Organization Modeling Method) pour la modélisation organisationnelle.
— IDEF13 appelée aussi TSMDM (Tree Schema Mapping Design Method).
— IDEF14 appelée aussi NDM (Network Design Method) pour la conception des réseaux.

En pratique, les méthodes les plus utilisées sont IDEFØ pour l'analyse fonctionnelle, IDEF1X pour la modélisation des données et IDEF3 pour la modélisation des processus.

1.9.2 La méthode IDEFØ

IDEFØ (Integration DEFinition language Ø) est basée sur SADT^{TM} (Structured Analysis and Design Technique^{TM}), développée par Douglas T. Ross et la société SofTech, Inc. Dans sa forme originale IDEFØ définissait en même temps un langage graphique de modélisation (la syntaxe et la sémantique) et donnait une description d'une méthodologie compréhensive pour développer des modèles.

Objectifs

IDEFØ peut être utilisée pour modéliser une large variété de systèmes automatisés ou non automatisés. Pour les nouveaux systèmes, IDEFØ peut être utilisée pour définir les besoins, les fonctions et pour en concevoir une implémentation adéquate. Pour les systèmes existants, IDEFØ peut être utilisée pour l'analyse des fonctions et des mécanismes par lesquels ils les réalisent.

Les modèles IDEFØ aident à organiser l'analyse d'un système et à promouvoir la communication entre analystes et clients. En tant qu'outil de communication, IDEFØ améliore la participation des experts du domaine et l'établissement d'un consensus décisionnel à travers sa représentation graphique. En tant qu'outil d'analyse, IDEFØ assiste le concepteur à identifier les fonctions à réaliser et les besoins pour les réaliser.

L'application d'IDEFØ à un système résulte en un modèle de diagrammes hiérarchisés se référençant les uns les autres. Les deux composants de base sont les fonctions (représentées par des boîtes) et les données qui relient les fonctions (représentées par des flèches).

Concepts des modèles IDEFØ

Un modèle est une représentation de l'ensemble des composants d'un système. Le modèle est développé pour la compréhension, l'analyse, l'amélioration ou le remplacement du système. Les systèmes sont composés de parties interfacées ou interdépendantes, fonctionnant ensemble pour réaliser une fonction utile. Les parties du système peuvent être des combinaisons de personnes, d'informations, de logiciels, de processus, d'équipements, de produits, de matières premières etc. Le modèle décrit ce que le système fait, ce qui le contrôle, les choses qu'il manipule, ce dont il se sert pour réaliser ses fonctions et ce qu'il doit produire.

IDEFØ est une technique de modélisation basée sur une combinaison de graphiques et de textes, présentés de façon organisée et systématique, au profit de la compréhension et du soutien de l'analyse. IDEFØ fournit une logique pour les changements potentiels, la spécification des besoins, le soutien des niveaux de conception des systèmes et l'intégration des activités. Un modèle IDEFØ est composé d'une hiérarchie de diagrammes qui exposent graduellement leurs niveaux de détails croissants en décrivant les fonctions et leurs interfaces dans le contexte du système. Il y a trois types de diagrammes : les graphiques, les textes et les glossaires. Les graphiques définissent les fonctions et leurs relations à travers la syntaxe et la sémantique de boîtes et de flèches. Les textes et les glossaires fournissent des informations additionnelles aux graphiques.

La syntaxe d'IDEFØ est composée de boîtes, de flèches, de règles et de diagrammes. Les boîtes représentent les fonctions définies comme des activités, des processus ou des transformations. Les flèches représentent des données ou des objets apparentés aux fonctions. Les règles définissent la manière dont les composants sont utilisés et les diagrammes fournissent un format pour représenter les modèles verbalement et graphiquement. Le format fournit aussi les bases

de la gestion des configurations du modèle. La sémantique est la signification des composants syntaxiques du langage et aide à une interprétation correcte.

Les boîtes

Une boîte fournit une description de ce qui se passe dans une fonction. Chaque boîte possède un nom et un numéro. Le nom doit être un verbe ou une phrase verbale qui décrit la fonction représentée. Le numéro sert à identifier la boîte et le texte associé. Chaque coté de la boîte a un sens bien précis dans la relation entre boîtes et flèches. Le coté de la boîte avec lequel une flèche s'interface reflète le rôle de la flèche.

Les flèches

Les flèches ne représentent pas les flux ou les séquences des modèles traditionnels de processus, elles transmettent des données ou des objets apparentés aux fonctions à réaliser. Les fonctions sont contraintes par la disponibilité des données et des objets qu'elles reçoivent. Une flèche est composée d'un ou plusieurs segments pouvant être droits ou incurvés et pouvant se diviser ou se rejoindre. Les segments possèdent des labels sous forme de noms explicitant les données ou objets qu'ils représentent.

Les boîtes et les flèches

Les flèches qui entrent à gauche représentent les entrées et celles qui ressortent à droite, les sorties. Les entrées sont transformées ou consommées par la fonction pour produire les sorties.

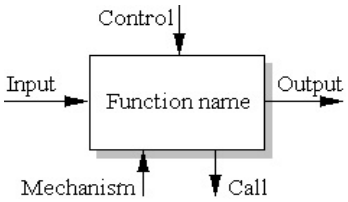


FIGURE 1.9 – *Positions et rôles des flèches*

Les flèches qui entrent par le haut représentent des contrôles qui spécifient les conditions nécessaires pour que la fonction produise un bon résultat. Les flèches qui entrent par le bas représentent des besoins nécessaires à l'exécution de la fonction. Ces besoins peuvent être hérités de la boîte de plus haut niveau. Les flèches sortant par le bas permettent de partager des détails entre les modèles ou des parties du même modèle. La boîte appelée fournit des détails à la boîte appelante. Ce type de flèche doit posséder comme label une référence à la boîte détaillant le sujet.

Labels et noms

Les boîtes représentent les fonctions devant être accomplies. Un nom de fonction doit être un verbe ou une phrase verbale. Les flèches identifient des données ou des objets nécessaires ou produit par la fonction. Chaque flèche doit posséder un label sous forme de nom ou de phrase nominale.

Règles entre boîtes et flèches

Chaque coté d'une boîte doit avoir une signification standard. Les flèches d'entrée doivent s'interfacer à gauche, les flèches de contrôle en haut, les flèches de sortie à droite. Les flèches de

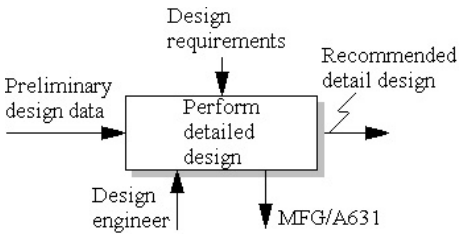


FIGURE 1.10 – *Sémantique des noms et labels*

mécanismes et les flèches d'appel doivent respectivement entrer et sortir par le bas. Les labels doivent être expressifs et une ligne brisée peut être utilisée pour relier un label à sa flèche si il y a ambiguïté.

Diagrammes IDEFØ

Les modèles IDEFØ sont composés de trois types d'information : les diagrammes, le texte, et les glossaires. Ces diagrammes se réfèrent les uns les autres. Le diagramme est le composant majeur d'un modèle IDEFØ, il contient des boîtes et des flèches interconnectées. Les boîtes représentent les fonctions principales d'un sujet. Ces fonctions sont éclatées ou décomposées en des diagrammes plus détaillés jusqu'à ce que le sujet soit décrit à un niveau suffisant. Le diagramme de plus haut niveau fournit la description la plus générale et abstraite, il est composé d'une série de diagrammes enfants fournissant plus de détails.

Diagrammes contextuels A-0

Chaque modèle doit avoir un diagramme contextuel, appelé diagramme A-0 dans lequel le sujet du modèle est représenté par une boîte seule et ses flèches. Les flèches de ce diagramme s'interfacent avec des fonctions extérieures au sujet. Comme cette boîte représente le sujet d'ensemble, son nom est général. De même pour les flèches qui représentent toutes les interfaces avec l'extérieur du sujet.

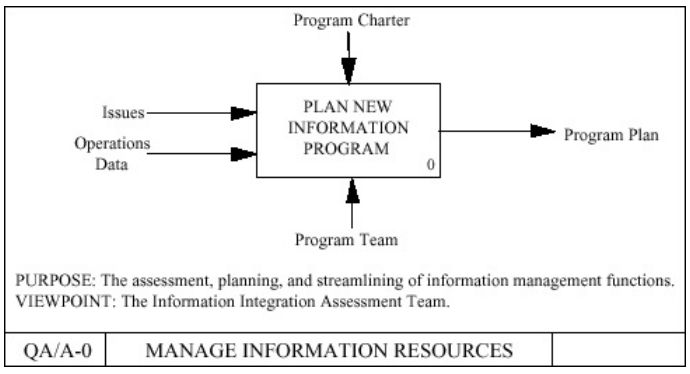


FIGURE 1.11 – *Diagramme contextuel A-0*

Le diagramme contextuel A-0 contient des emplacements spécifiant le point de vue et le but du modèle. Le point de vue détermine ce qui peut être vue dans le contexte. Suivant à qui s'adresse le modèle, différents points de vue peuvent être adoptés pour mettre l'accent

sur différents aspects du sujet. Le but exprime la raison pour laquelle le modèle a été créé et détermine sa structure. Les caractéristiques les plus importantes figurent en premier dans la hiérarchie. La fonction de plus haut niveau est décomposée en sous fonctions, elles même décomposées jusqu'à ce que tous les détails pertinents soient exposés. Chaque sous fonction est modélisée individuellement par une boîte. Les boîtes parents sont détaillées, au niveau inférieur, par des diagrammes enfants. Tous les diagrammes enfants doivent être dans le champ du diagramme contextuel.

Diagrammes enfants

L'unique fonction du diagramme contextuel doit être décomposée en sous fonctions dans des diagrammes enfants. Ces sous fonctions doivent être décomposées, à tour de rôle, en sous fonctions dans des diagrammes enfants de niveaux inférieurs. Pour un diagramme donné, aucune ou toutes les fonctions doivent être décomposées. Chaque diagramme enfant contient des boîtes et des flèches qui détaillent leur diagramme parent. Le diagramme enfant issu de la décomposition couvre le même champ que son parent qu'il détaille. Le diagramme enfant doit être vu comme étant le contenu de son parent. Voir annexe B «Diagrammes IDEFØ».

Diagrammes parents

Un diagramme parent contient une ou plusieurs boîtes enfants. Tout diagramme ordinaire (non contextuel) est aussi un diagramme enfant qui détaille sa boîte parent. Un diagramme peut donc être à la fois parent et enfant. De la même façon, une boîte peut être une boîte parent (détaillée par des diagrammes enfants) et une boîte enfant (figurant sur un diagramme parent). La relation hiérarchique primordiale se trouve entre une boîte parent et le diagramme enfant qui la détaille.

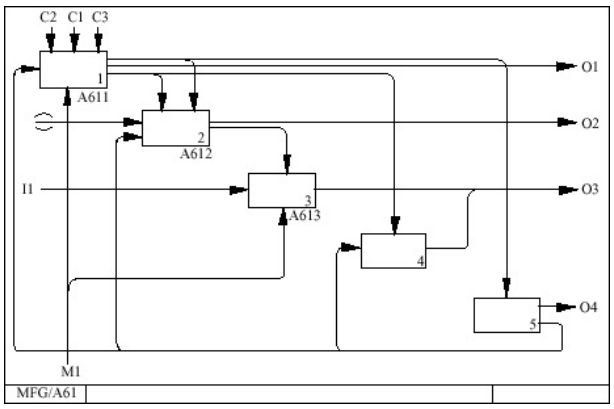


FIGURE 1.12 – *Detail Reference Expression (DRE)*

Le fait qu'une boîte enfant soit détaillée et qu'elle soit, par conséquent, aussi une boîte parent est indiqué par la présence d'un code DRE (Detail Reference Expression) situé dans le coin inférieur droit de la boîte détaillée et qui pointe sur ses diagrammes enfants.

Synthèse

Nous n'avons pas détaillé tous les points de la norme mais avons donné une vue d'ensemble de la démarche IDEFØ. Cette méthode nous semble intéressante pour le développement du modèle

produit. En effet, cette méthode d'analyse fonctionnelle permet de modéliser les différentes étapes du cycle de vie du produit et de mettre en évidence leurs besoins informatifs. A notre avis, cette méthode est utile durant la phase d'expression des besoins du processus de développement du modèle produit pour un domaine d'application donné. De plus, sa représentation graphique en fait un bon outil de communication et par conséquent un bon support pour la critique et l'amélioration continue des modèles.

1.9.3 La méthode IDEF1X

La première méthode IDEF de modélisation de l'information (IDEF1) fut publiée dans le cadre du programme ICAM en 1981. Basée sur les recherches actuelles et les besoins de l'industrie elle trouve ses fondements théoriques dans les travaux sur la théorie relationnelle de E. F. Codd et sur le modèle entité relation de P. P. S. Chen [13],[14],[15]. La technique IDEF1 est basée sur les travaux de R. R. Brown et T. L. Ramey de la société Hugues Aircraft et D. S. Coleman de la société D. Appleton avec une revue critique et l'influence de C. W. Bachman, de P. P. S. Chen, M. A. Melkanoff et G. M. Nijssen [8].

En 1983, dans le cadre du projet ICAM, l'U.S Air force lance le projet IISS (Integrated Information Support System) dont l'objectif est de fournir une technologie permettant d'intégrer logiquement et physiquement un réseau d'ordinateurs à base de matériels et de logiciels hétérogènes. Ce projet identifia le besoin de techniques de modélisation de l'information. Les applications au sein de l'industrie ont mené, en 1982, au développement des techniques de conception des bases de données logiques LDDT (Logical Design Data base Technique) de R. G. Brown du groupe de conception des bases de données DDG (Data base Design Group). La technique est basée sur le modèle relationnel de E. F. Codd, le modèle entité relation de P. P. S. Chen et les concepts de généralisations de J. M. Smith et D. C. P. Smith. Ces techniques ont fourni différents niveaux de modèles ainsi qu'un ensemble de graphiques représentant la vue conceptuelle de l'information dans l'entreprise. LDDT possède un grand nombre de caractéristiques de IDEF1, une sémantique améliorée, des constructions graphiques, et traite des besoins identifiés par le programme IISS. Sous la direction technique de M. E. S. Loomis de la société D. Appleton, un sous ensemble de LDDT fut combiné avec la méthodologie IDEF1 pour former une version étendue IDEF1X qui fut publiée par l'ICAM en 1985.

Objectif

L'objectif principal d'IDEF1X est de soutenir l'intégration. L'approche suivie dans le projet IISS focalise sur la capture, la gestion et l'usage d'une unique définition sémantique des données, le schéma conceptuel. La notion de schéma conceptuel fournit une seule définition intégrée, indépendante des méthodes de stockage et d'accès, des données d'une entreprise, quelle qu'en soit l'application. L'objectif premier du schéma conceptuel est de fournir une définition consistante de la signification et de l'interrelation des données pouvant être utilisée pour intégrer, partager et gérer l'intégrité des données. Un schéma conceptuel doit posséder trois caractéristiques importantes :

- Il doit être cohérent avec l'infrastructure du marché et être valide pour tous les domaines d'application.
- Il doit être extensible, pour permettre à de nouvelles données d'être définies sans altérer les précédentes.

— Il doit être transformable pour s'adapter aux vues de l'utilisateur et à une variété de structures de stockage et d'accès.

Concepts des modèles IDEF1X

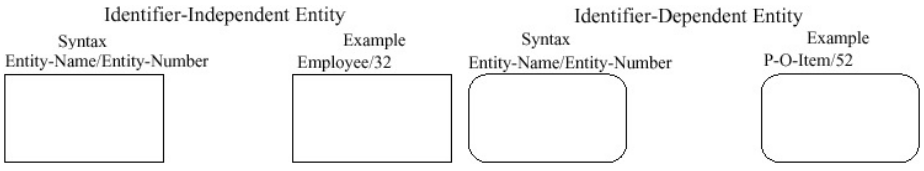
IDEF1X est utilisée pour produire des modèles graphiques d'information représentant la structure et la sémantique des informations d'un environnement ou d'un système. L'usage de cette norme permet de modéliser la sémantique des données pour en favoriser la gestion en tant que ressource, l'intégration des systèmes d'information et la construction des bases de données. Les objectifs premiers de cette norme sont de fournir :

— Un moyen d'analyse et de compréhension de l'organisation des données.
— Un moyen commun de représenter et de communiquer la complexité des données.
— Une méthode pour présenter une vue d'ensemble des données nécessaires au fonctionnement d'une entreprise.
— Un moyen de définir une vue des données, indépendante de l'application et pouvant être validée et utilisée pour la conception d'une base de données.
— Une méthode pour dériver et intégrer de nouvelles définitions de données aux ressources existantes.

Les entités

Les entités représentent un ensemble de choses réelles ou abstraites (personnes, objets, événements, lieux, idées, etc.) qui ont des attributs ou caractéristiques communes. Un membre individuel de l'ensemble est appelé instance de l'entité. Un objet réel est représenté par plusieurs entités suivant la vue considérée. Par exemple, Johnny Hallyday peut à la fois être, une instance de l'entité «motard» et «chanteur». Une entité peut aussi représenter une combinaison d'objets réel. Par exemple, Johnny Hallyday et Laetitia peuvent être une instance de l'entité «couple marié».

Une entité est dite indépendante si chacune de ses instances peut être identifiée d'après ses relations avec d'autres entités. Une entité est dite dépendante si une instance de l'entité peut être identifiée de façon unique d'après ses relations avec d'autres entités.



(a) Indépendantes (b) Dépendantes

FIGURE 1.13 – *Entités*

Les entités sont représentées par des boîtes. Les entités dépendantes sont représentées par des boîtes aux coins arrondis. A chaque entité est attribué un label situé au dessus de la boîte. Ce label doit contenir le nom et le numéro de l'entité.

Les domaines

Un domaine représente un nom et un ensemble défini de valeurs que peuvent prendre les attributs. IDEF1X définit les domaines séparément des entités et des vues pour permettre leur réutilisation et leur standardisation à travers l'entreprise.

Un domaine est considéré comme une classe pour laquelle il existe un ensemble fini ou non d'instances. Par exemple, «code d'état» peut être considéré comme un domaine où l'ensemble des valeurs autorisées satisfaisant la définition du code d'état (identifiant unique d'un état).

Chaque instance du domaine doit avoir une valeur unique dans toutes les représentations considérées du domaine. Le domaine «code d'état» peut être représenté de plusieurs manières : par les noms [Alabama, Alaska, Arizona], par les abréviations [AL, AK, AR] ou par les numéros d'états [1, 2, 3]. Chaque représentation d'instance doit être unique à l'intérieur du domaine. La représentation utilisant les premières lettres des états [A, A, A] n'est donc pas valide.

Il existe deux types de domaines, les domaines de base et les domaines typés. Les domaines de base sont constitués de types de base auxquels sont assignées des règles de validité. Les domaines typés sont des sous types des types de base ou d'autres domaines typés qui peuvent, eux aussi, posséder des contraintes de validité. Ainsi, il est possible de définir une hiérarchie de domaines allant des plus généraux aux plus particuliers.

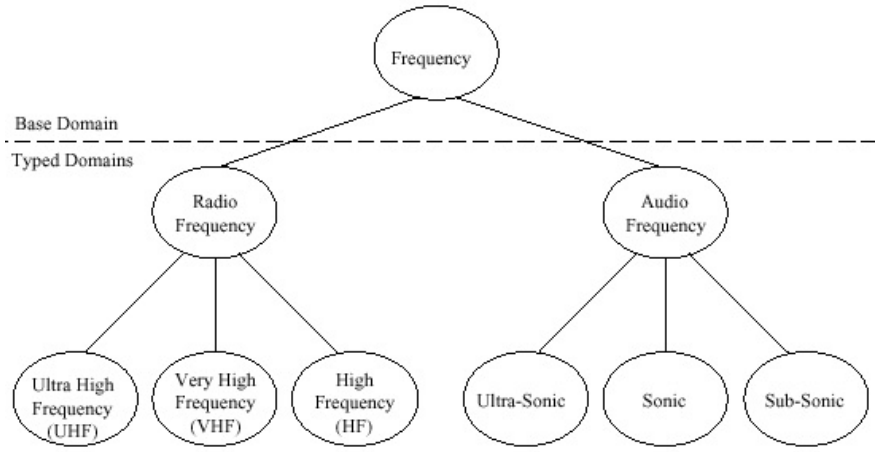


FIGURE 1.14 – *Exemple de hiérarchie de domaines*

Aucune syntaxe ne correspond à la notion de domaine dans IDEF1X. Les informations relatives aux types de données pour les domaines de base, aux relations de sous typage pour les domaines typés, ainsi que les règles de validité et les représentations sont indiquées de manière informelle dans un glossaire.

Les vues

Une vue IDEF1X est une collection d'entités constituée d'un assemblage de domaines (les attributs). Une vue existe dans un certain but et peut couvrir en partie, ou en totalité, le domaine à modéliser. Un modèle IDEF1X est constitué de diagrammes de vues représentant la sémantique sous-jacente. Les entités et les domaines sont définis dans un glossaire commun et projetés dans les vues. Ainsi une entité peut apparaître dans plusieurs vues, plusieurs modèles et avoir des attributs différents dans chacun d'eux. Une entité doit avoir une seule signification par vue. Une vue possède un nom et des informations descriptives optionnelles comme le nom de l'auteur, la date, la révision, etc. Une description textuelle peut accompagner la vue, celle-ci doit décrire succinctement les entités, leurs attributs et relations ainsi que leurs contraintes de validité.

Les attributs

Un domaine associé à une entité dans une vue se réfère à un attribut de l'entité. Dans une vue IDEF1X un attribut représente un type de caractéristiques ou de propriétés associées avec un ensemble abstrait ou concret de choses (personnes, objets, lieux, événements, idées, etc.). Une instance d'attribut est définie par son type et sa valeur. Un attribut peut dans certains cas avoir une valeur indéfinie. Une entité peut avoir un ou une combinaison d'attributs dont les valeurs identifient de façon unique chaque instance de l'entité. Ces attributs forment la clef de l'entité.

Dans une vue basée sur les clefs ou dans une vue complètement attribuée tous les attributs sont la propriété d'une seule entité. Par exemple, l'attribut «mensualisé» peut s'appliquer à certaines instances de la classe «employé» mais pas à toutes. Néanmoins, il est possible d'identifier une catégorie distincte mais apparentée pouvant posséder l'attribut «mensualisé», la catégorie «employé salarié». Ainsi, un employé salarié représente à la fois, une instance des entités «employé» et «employé salarié». Les attributs communs à tous les employés, tels leur nom et leur date de naissance, sont définis au niveau de l'entité «employé» et non «employé salarié». De tels attributs sont dit hérités par l'entité de la catégorie et ne figurent pas dans la liste de ces attributs mais dans la liste des attributs de l'entité générique.

En plus d'appartenir à une entité, un attribut peut être présent dans une entité grâce à sa migration le long d'une relation, spécifique ou de catégorisation. Si, par exemple, tous les employés sont affectés à un département, alors l'attribut «numéro de département» peut être un attribut de l'entité «employé» par migration de l'entité «département» vers l'entité «employé». Par contre, les attributs n'identifiant pas l'entité «département», c'est-à-dire ne faisant pas partie de sa clef primaire, tel l'attribut «nom de département», ne doivent pas migrer.

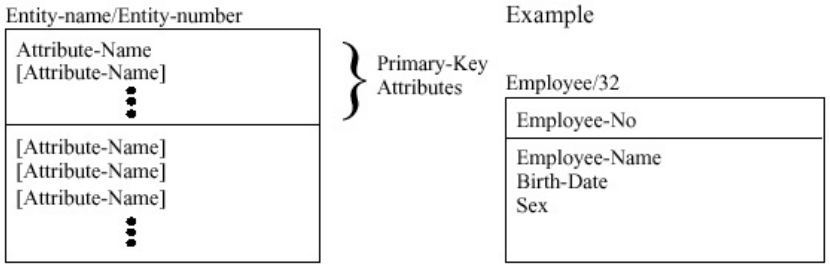


FIGURE 1.15 – *Attributs et clef primaire*

Les attributs sont représentés à l'intérieur de l'entité par leurs noms et ceux qui forment la clef primaire sont situés dans la partie supérieure, séparés des autres par une ligne horizontale.

Les relations spécifiques d'association

Ce sont des relations parent/enfants où toutes les instances de l'entité mère sont associées à zéro, une ou plusieurs instances de l'entité fille et où chaque instance de l'entité fille est associée à une seule instance de l'entité mère.

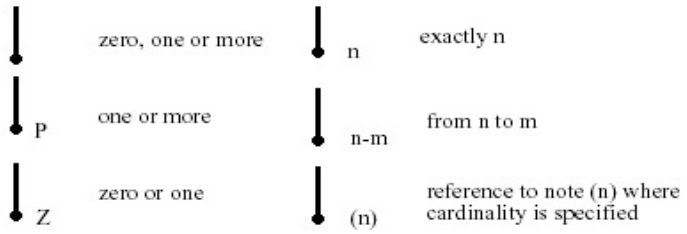
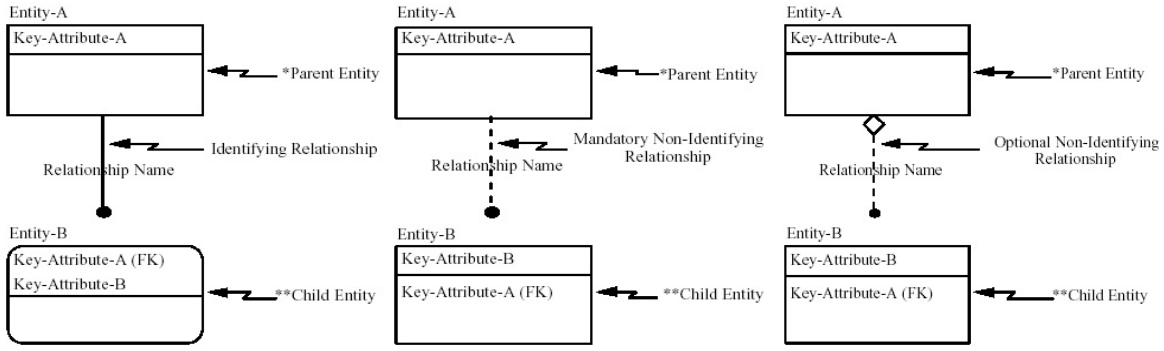


FIGURE 1.16 – *Relations d'association*

Si les instances des entités filles ne peuvent être identifiées sans connaître leur association avec l'entité mère, on parle de relation identifiante. Les entités filles sont existentiellement dépendantes de l'entité mère, elles ne peuvent exister sans elle. L'identification des instances des entités filles passe par l'identification des instances de l'entité mère (cf. clés étrangères).

Si les instances des entités filles peuvent être identifiées, de façon unique, sans connaître leur association avec l'entité mère, on parle de relation non identifiante. Dans ce cas, les entités filles sont indépendantes de l'entité mère.



(a) Identifiante (b) Non identifiante (c) Non identifiante optionnelle

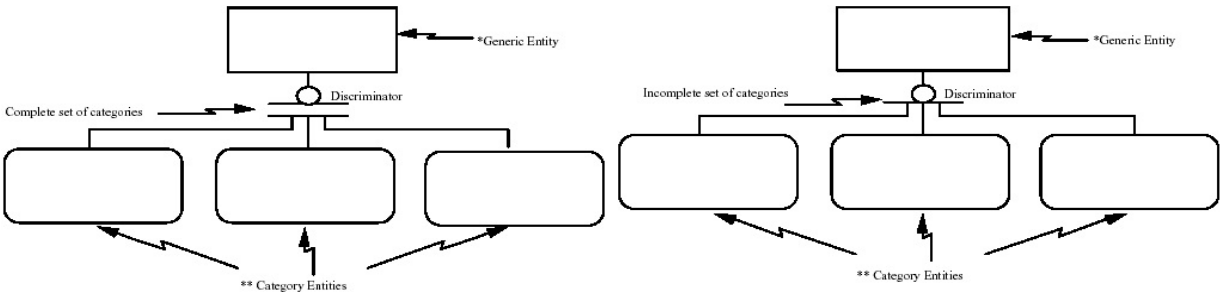
FIGURE 1.17 – *Relations*

De plus, dans le cas d'une relation non identifiante, si chaque instance de l'entité fille est associée à zéro ou une entité mère, on parle de relation non identifiante optionnelle.

Les relations de catégorisation

Les relations de catégorisation sont utilisées pour structurer les entités en catégories et sous catégories. De la même manière qu'il existe des catégories de choses réelles, il est possible de catégoriser les entités. Prenons l'exemple des employés. Certaines informations sont relatives à tous les employés, d'autres sont spécifiques aux employés mensualisés ou encore aux employés journalisés.

Une relation de catégorisation intervient entre une entité dite générique et une entité de catégorie. Dans l'exemple précédent, l'entité «employé» est une entité générique et les entités «employé mensualisé» et «employé journalisé» sont des entités de catégorie. Il y a donc deux relations de catégorisation la relation entre «employé» et «employé mensualisé» et la relation entre «employé» et «employé journalisé». Les catégories d'un même ensemble sont mutuellement exclusives, un employé mensualisé n'est pas journalisé. De plus, un ensemble de catégories peut être soit complet soit incomplet. Par exemple, un employé est soit une femme soit un homme, l'ensemble des catégories «homme salarié» et «femme salariée» est complet. Un attribut de l'entité générique doit servir de discriminant pour un ensemble de catégories. La valeur de cet attribut détermine la catégorie à laquelle appartiennent les instances d'une entité générique.



(a) Complet (b) Incomplet

FIGURE 1.18 – *Ensemble de catégories*

Les relations non spécifiques d'association

Les relations non spécifiques sont utilisées dans les vues de haut niveau pour représenter les relations qu'ont plusieurs entités avec plusieurs autres entités. Les relations parent/enfants et les relations de catégorisation sont considérées comme spécifiques car elles définissent précisément combien d'instances d'une entité correspondent à une entité. Dans une vue basée sur les clefs ou complètement attribuée, toutes les associations doivent être spécifiques. Cependant, dans les premières phases du développement d'un modèle, il est souvent utile d'identifier les relations non spécifiques. Celles-ci seront raffinées dans les phases de développement ultérieures.

Une relation non spécifique est une association entre deux entités pour lesquelles ont associe zéro, une ou plusieurs instances de la deuxième entité à une instance de la première et inversement on associe zéro, une ou plusieurs instances de la première entité à une instance de la deuxième.

Par exemple, plusieurs projets peuvent être attribués à un employé et plusieurs employés peuvent être attribués à un projet. Cette relation non spécifique est remplacée en introduisant une troisième entité «attribution de projet». Celle-ci devient une entité fille commune aux entités «employés» et «projets» en les reliant par des relations spécifiques.

Les clefs primaires et alternatives

Une clef candidate est formée d'un ou plusieurs attributs dont les valeurs identifient chaque instance de l'entité. Par exemple, l'attribut «identificateur d'ordre d'achat» identifie une instance de l'entité «ordre d'achat». Une combinaison des attributs «identificateur de compte» et «identificateur de chèque» identifie une instance de l'entité «chèque».

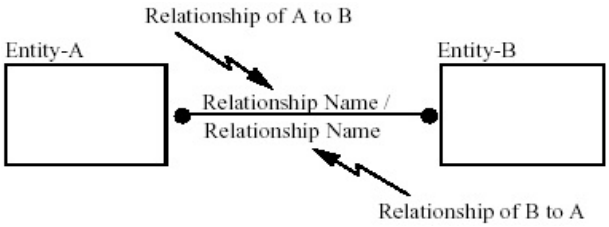


FIGURE 1.19 – *Relation non spécifique*

Dans une vue basée sur les clefs ou complètement attribuée, toutes les entités doivent avoir au moins une clef candidate. Dans certains cas, une entité a plus d'un attribut identifiant ses instances. Par exemple, les attributs «identificateur d'employé» et «numéro de sécurité sociale» identifient tous les deux une instance de l'entité «employé». Si plus d'une clef est candidate alors une clef est désignée comme étant la clef primaire et les autres comme des clefs alternatives. Si il n'y a qu'une clef candidate, c'est forcément la clef primaire.

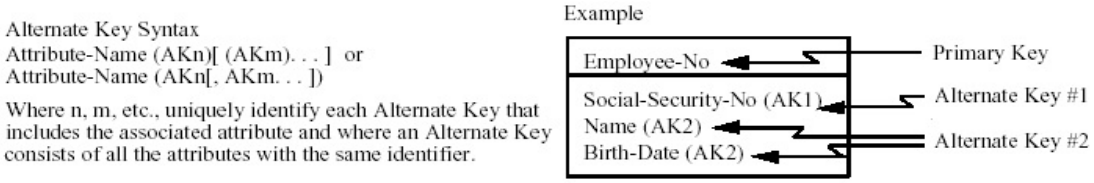


FIGURE 1.20 – *Clef alternée*

Les attributs définissant la clef primaire sont placés dans la boîte de l'entité, au sommet de la liste des attributs, il sont séparés par une ligne horizontale. A chaque clé alternative est attribué un identifiant entre parenthèses (exemple : AK1). Un attribut individuel peut être une partie d'une clef alternative. Une clef primaire peut aussi être une partie d'une clef alternée.

Les clefs étrangères

Ce sont les attributs d'une entité qui désignent les instances des entités qui lui sont reliées. Si une association spécifique ou une relation de catégorisation existe entre deux entités, alors les attributs qui forment la clef primaire de l'entité mère, ou générique, migrent pour devenir des attributs des entités filles ou de catégorie, c'est ce qu'on appelle les clefs étrangères. Par exemple, si «tâche» est une entité fille de l'entité «projet» alors la clef primaire «identificateur de projet» de l'entité «projet» migre vers l'entité «tâche».

Un attribut immigré peut faire partie d'une partie ou de la totalité, de la clef primaire, d'une clef alternative, d'un attribut de l'entité. Si tous les attributs de l'entité mère migrent vers l'entité fille, en tant que parties de la clef primaire, alors la relation concernée est une relation identifiante. Si certains des attributs immigrés ne font pas partie de la clef primaire alors la relation est non identifiante. Par exemple, si des tâches sont numérotées à l'intérieur d'un projet, l'attribut immigré «identificateur de projet» devrait être combiné avec l'attribut «identificateur de tache» pour définir la clef primaire de l'entité «tache». Les entités «projet» et «tâche» sont reliées par une relation identifiante. Si, d'un autre coté, l'attribut «identificateur

de tâche» est toujours unique pour tous les projets alors l'attribut «identificateur de projet» n'est pas un attribut clef de l'entité «tâche». Dans ce cas, les entités «projet» et «tâche» sont reliées par une relation non identifiante. Quand une partie de la clef primaire de l'entité mère migre et devient une partie de la clef primaire de l'entité fille et que la partie restante se transforme en attribut non clef, la clef étrangère est appelée clef déchirée et la relation est non identifiante.

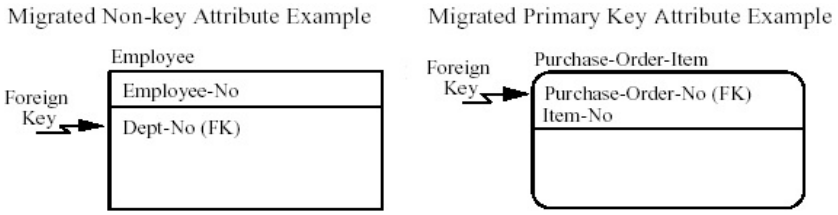


FIGURE 1.21 – *Clef étrangère*

Dans une relation de catégorisation, les entités génériques et de catégorie représentent le même monde réel, par conséquent, les clefs primaires de toutes les entités de catégorie sont des clefs immigrées de l'entité mère le long de la relation de catégorisation. Par exemple, si «employé mensualisé» et «employé journalisé» sont des catégories de l'entité générique «employé», l'attribut «identificateur d'employé» est la clef primaire de l'entité «employé» et doit être la clef primaire des entités «employé mensualisé» et «employé journalisé».

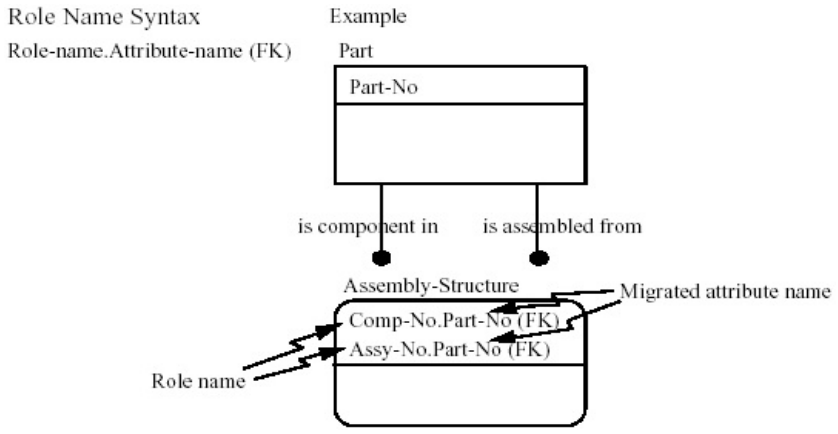


FIGURE 1.22 – *Noms de rôles*

Dans certains cas, une entité fille peut avoir de multiples relations avec la même entité mère. La clef primaire de l'entité mère devrait apparaître dans l'entité fille, pour chaque relation, comme attribut immigré. Pour une instance donnée de l'entité fille, les valeurs des attributs immigrés peuvent être différentes pour chaque relation, c'est-à-dire que plusieurs instances de l'entité mère peuvent être référencées. Etant donné qu'un attribut ne doit figurer qu'une seule fois dans une entité, les différentes occurrences des attributs immigrés sont fusionnées, à moins que des noms de rôles différents leur soient attribués.

Les niveaux de vue

Il existe trois niveaux de schémas conceptuels différents syntaxiquement et sémantiquement.
— La vue entités/relations (ER) où ne figurent pas les clefs.
— La vue basée sur les clefs (KB) où figurent les clefs et certains attributs non clef.
— La vue complètement attribuée (FA) où figurent tous les attributs.

Les différents niveaux de vue du schéma conceptuel fournissent la structure nécessaire à la conception des bases de données pour différents systèmes. La syntaxe graphique d'IDEF1X est souvent utilisée informellement pour décrire la structure physique des bases de données. Ceci peut être utile à la rétro ingénierie des systèmes existants, en fournissant des méthodes pour dériver et intégrer de nouvelles définitions de données à celles qui existent déjà.

Feature \ View Type	ER Level	KB Level	FA Level
Entities	yes	yes	yes
Specific Relationships	yes	yes	yes
Non-specific Relationships	yes	no	no
Relations Categorizationhips	yes	yes	yes
Primary Keys	no	yes	yes
Alternate Keys	no	yes	yes
Foreign Keys	no	yes	yes
Non-key Attributes	yes (note 1)	yes	yes
Notes	yes	yes	yes

FIGURE 1.23 – *Niveaux de vue*

La vue entités/relations doit contenir les entités, leurs relations, les attributs et ne doit contenir aucune clef. De ce fait, il est impossible de distinguer entités dépendantes ou indépendantes et relations identifiantes ou non identifiantes. La vue entités/relations fait figurer les relations non spécifiques et de catégorisation mais l'attribut discriminant est optionnel.

La vue basée sur les clefs doit contenir les entités, les relations, les clefs primaires, les clefs étrangères. Les entités et les relations se distinguent respectivement en tant que dépendantes ou indépendantes et identifiantes ou non identifiantes. Dans les relations non identifiantes, les cardinalités côté parent doivent être précisées comme obligatoires ou optionnelles. Chaque ensemble de catégories doit faire figurer son attribut discriminant. Les associations non spécifiques ne sont pas autorisées. Chaque entité doit posséder une clef primaire, des clefs alternatives additionnelles, pour chaque contrainte d'identification unique, des clefs étrangères pour chaque relation d'association et de catégorisation dans lesquelles elle est l'entité fille ou de catégorie. La vue basée sur les clefs peut contenir des attributs non clef.

La vue complètement attribuée doit satisfaire les mêmes critères que la vue basée sur les clefs en faisant figurer, en plus, tous les attributs en rapport avec le sujet de la vue.

Synthèse

IDEF1X est une méthode graphique de description des schémas conceptuels. Basée sur l'approche relationnelle elle comporte, via le concept de généralisation ou d'héritage, les prémisses de l'approche orientée objet. Cette méthode est donc bien adaptée à la conception de bases de données relationnelles et ses concepts sont également importants du point de vue de la conception orientée objet.

Pourtant, il semble que certains aspects fondamentaux de l'approche orientée objet soient indispensables à l'interopérabilité logiciel. En effet, cette dernière se distingue de l'approche relationnelle par la présence de méthodes qui, en encapsulant la représentation des données, confèrent aux entités un aspect fonctionnel, on parle du comportement de l'objet. Aussi, les attributs ou données membres ne sont plus les seuls critères de catégorisation et les fonctionnalités viennent s'y adjoindre. Finalement, le principe de polymorphisme, qui consiste à dissocier, l'interface de l'implémentation d'une méthode, permet l'encapsulation des traitements. En encapsulant les données et traitements, ces mécanismes permettent l'abstraction des représentations nécessaire à l'interopérabilité logiciel. Il semble donc qu'une solution permettant l'interopérabilité logicielle doive intégrer ces concepts par l'utilisation de méthodes appropriées telles que IDEF4, Merise, OSE, OMT, Booch ou l'unification de celles-ci, UML [16].

1.10 Conclusion

L'objectif de STEP est de répondre au problème de l'échange et du partage du modèle produit. Nous avons évoqué l'aspect méthodologique de STEP qui consiste à spécifier le modèle produit. Dans ce cadre, ont été évoquées les méthodes IDEFØ pour l'identification des flux d'information le long du processus de développement, et IDEF1X pour leur structuration sous forme de schémas conceptuels.

Rappelons que le problème des échanges de données est, avant tout, dû à la diversité et à la complexité des représentations faites par les différents systèmes. Ainsi, STEP propose l'uniformisation de celles-ci, sous forme de bibliothèques de schémas relationnels. Pour cela, il définit le langage de description des schémas EXPRESS. Les schémas génériques sont réutilisés pour former des schémas spécialisés. Ensuite, sont définies des solutions d'échanges, spécifiques à chaque métier, les protocoles d'application, qui définissent un ensemble de bibliothèques de schémas métiers. Puis STEP définit un format d'échange permettant d'instancier les modèles EXPRESS appartenant à un protocole d'application donné. Finalement, l'interface SDAI permet d'interfacer les instances EXPRESS avec les instances natives et inversement.

Le principal reproche qu'il puisse être fait à STEP est qu'en se focalisant sur les données, il réduit l'interopérabilité logicielle à l'échange et au partage de celles-ci. Or, depuis qu'a débuté son développement, d'autres approches de l'interopérabilité logicielle ont été menées. Basées sur les notions d'échanges de services, d'abstraction des représentations et des traitements, les approches orientées objet grâce aux principes, d'héritage, d'encapsulation et de polymorphisme proposent de nouvelles perspectives au problème de l'interopérabilité logicielle. Nous aborderons donc ces approches dans le prochain chapitre. Nous présenterons la problématique de l'interopérabilité logicielle à travers l'architecture de composants COM et l'architecture d'objets distribués CORBA. Puis nous présenterons les récentes approches de l'interopérabilité dans les domaines de l'IAO.

Chapitre 2

L'approche médiateur

2.1 Introduction

En ouvrant de nouvelles perspectives de travail collaboratif, l'avènement d'Internet et des réseaux informatiques pose la problématique de l'interopérabilité logicielle. En effet, comment diverses applications peuvent elles collaborer à une même oeuvre ? Comment construire des applications par assemblage de composants logiciels inter opérables ? Comment une application peut elle tirer profit, de façon transparente, de ressources matérielles distribuées ? Nous présentons la problématique de l'interopérabilité logicielle à travers l'architecture de composants logiciels, COM et l'architecture d'objets distribués, CORBA. Enfin, nous présentons les récentes approches de l'interopérabilité logicielle dans le domaine de l'IAO.

2.2 L'architecture de composants logiciels COM

COM (Component Object Model) est une architecture logicielle permettant l'interopérabilité de composants logiciels binaires [17]. Développée par Microsoft, COM permet aux applications et systèmes d'être construits à base de composants binaires délivrés par différents distributeurs. Indépendamment des langages de programmation, disponible sur différentes plateformes (Windows, Apple Macintosh, UNIX), extensible, COM permet l'évolution robuste des logiciels et systèmes à base de composants.

L'architecture COM représente les fondements de services logiciels de plus haut niveau, tel que OLE (Object Linking and Embedding). Les services OLE couvrent différents aspects des composants logiciels incluant les documents composites, les contrôles personnalisés, les scripts inter applications, le transfert de données et autres interactions logicielles.

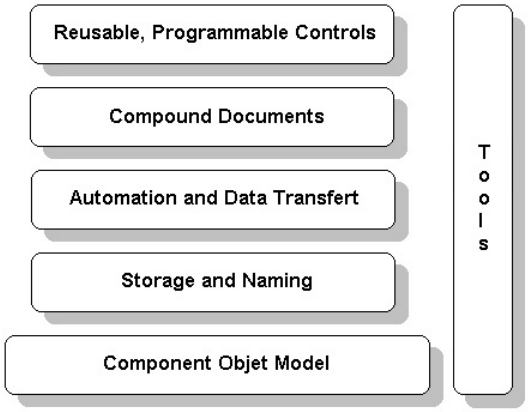


FIGURE 2.1 – *Les services d'application OLE*

Ces services fournissent différentes fonctionnalités à l'utilisateur. Ils partagent, cependant, les mêmes besoins fondamentaux : un mécanisme permettant à des composants logiciels binaires, délivrés par différents distributeurs, d'être connectés et de communiquer d'une manière bien définie. COM fournit ces mécanismes en permettant aux composants de communiquer au delà des limites des processus et réseaux. Il est important de noter que COM est une architecture générale de composants logiciels et que tout projet de développement peut en tirer parti.

Nous présentons l'interopérabilité que fournit COM en définissant premièrement ses principes de conception ainsi que ses concepts architecturaux. Ce faisant, nous examinerons les problèmes spécifiques que COM est sensé résoudre ainsi que les solutions qu'il propose.

— La problématique des logiciels à base de composants : comment des composants exécutables, délivrés par divers fournisseurs, développés à différents endroits et à différentes périodes peuvent-ils inter-opérer au sein d'un système cohérent ? Ce problème passe par la résolution de quatre sous problèmes.
— L'interopérabilité de base : comment des développeurs peuvent ils créer leurs propres composants en étant surs qu'ils inter-opéreront avec des composants écris par d'autres programmeurs ?
— La gestion de version : comment un système peut il être mis à jour sans pour autant que tous ses composants soient mis à jour ?
— L'indépendance face au langage : comment des composants écrits dans des langages de programmation différents peuvent ils communiquer ?
— La transparence de l'interopérabilité : comment peut on donner aux développeurs la flexibilité d'écrire des composants, s'exécutant indifféremment dans un même processus ou dans des processus différents, en utilisant un modèle de programmation simple et unique ?

2.2.1 Caractéristiques fondamentales de COM

COM définit les concepts fondamentaux qui forment ses fondements structurels.
— Un standard binaire pour les appels de fonctions entre composants.
— La mise à disposition de groupes fortement typés de fonctions, les interfaces.

— La capacité des composants à découvrir dynamiquement les interfaces qu'ils implémentent.
— Le comptage des références permettant aux composants de suivre leur propre cycle de vie et de se détruire le cas échéant.
— Un mécanisme d'identification des composants et de leurs interfaces.
— Un chargeur de composants préparant les composants et leurs interactions.

2.2.2 Standard binaire

Pour chaque plateforme (combinaison d'architecture matérielle et système d'exploitation), COM définit une façon standard de disposer en mémoire les tables de fonctions virtuelles (vtables) et une façon d'appeler ces fonctions via la vtable.

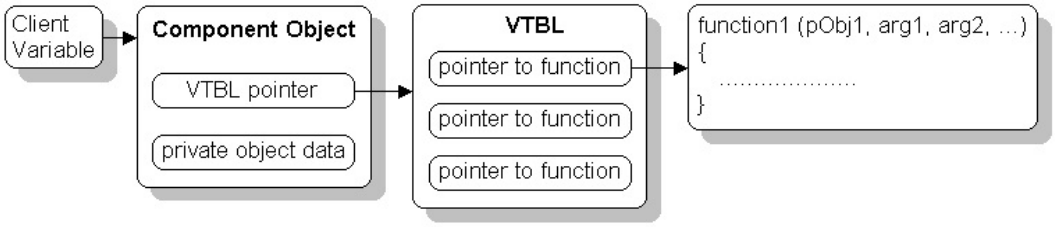


FIGURE 2.2 – *Tables des fonctions virtuelles*

Ainsi, tout langage permettant l'appel d'une fonction via un pointeur peut servir à écrire un composant inter opérable (C, C++, Small Talk, Ada, Basic). De plus, chaque instance d'une même classe de composants partage la même table de fonctions virtuelles, ce qui permet à des systèmes à grand nombre de composants de minimiser l'occupation mémoire.

2.2.3 Objets et composants

Il ne faut pas confondre les objets COM avec les objets des langages orientés objet tel que C++. Un composant COM est un morceau de code compilé rendant des services au reste du système. Un composant COM possède habituellement des données mais, contrairement aux objets C++, elles ne peuvent être accédées directement. A la place, les composants interagissent entre eux par l'intermédiaire d'un pointeur sur leurs interfaces, c'est le principe d'encapsulation des données et des traitements. Ce principe est fondamental dans les architectures de composants logiciels car il autorise l'accès aux données au delà des limites des processus et réseaux.

2.2.4 Interfaces

Dans COM, les composants interagissent via des collections d'opérations liées sémantiquement, les interfaces. Elles représentent des contrats fortement typés entre composants en définissant leurs responsabilités et leurs comportements. Un pointeur sur un composant est, en réalité, un pointeur sur une des interfaces qu'il implémente. Il est donc impossible de modifier directement les données de l'objet, seul l'appel de méthodes est autorisé. Il existe une interface implémentée par tout composant et de laquelle dérivent toutes autres interfaces, l'interface

«IUnknown». Celle-ci regroupe des services de gestion du cycle de vie des composants, ainsi qu'un service permettant de renseigner, dynamiquement à d'autres composants, les interfaces implémentées par un composant donné.

2.2.5 Propriétés des interfaces

— Une interface est un contrat dans lequel un composant expose ses services. C'est un groupe d'opérations liées sémantiquement. Les interfaces d'un objet sont couramment représentées graphiquement comme des prises de courant.
— Une interface n'est pas une classe car elle ne définit pas l'implémentation de ses opérations, les méthodes. Elle ne peut donc être instancier directement. Cependant, plusieurs classes peuvent implémenter, de différentes manières, une même interface, à condition que le contrat soit respecté, c'est le principe du polymorphisme.
— Une interface n'est pas un composant, c'est le moyen à travers lequel un client communique avec le composant indépendamment de la manière dont il a été implémenté (langage de programmation et représentation interne).
— Les composants interagissent via un pointeur sur leurs interfaces. C'est un moyen opaque, masquant tous les aspects de l'implémentation.

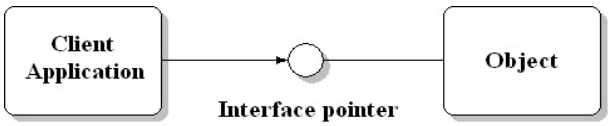


FIGURE 2.3 – *Les composants interagissent via leurs interfaces*

— Contrairement aux pointeurs d'objets, un pointeur d'interface ne permet pas d'accéder aux attributs. C'est le principe d'encapsulation qui fournit la transparence des interactions locales et distantes.

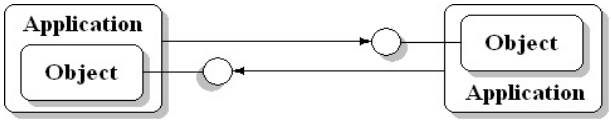


FIGURE 2.4 – *Connexion entre composants d'applications différentes*

— Les composants peuvent implémenter plusieurs interfaces afin de proposer plusieurs ensembles de services.

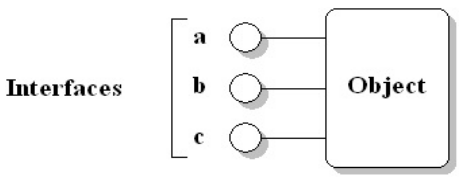


FIGURE 2.5 – *Objet supportant trois interfaces*

— Une interface est fortement typée. Chaque interface possède un identifiant unique GUID (Global Unique IDentifier) attribué par l'environnement de développement et nécessaire au programmeur pour obtenir un pointeur sur une interface. Ceci, afin d'éviter les conflits liés aux noms lisibles ainsi que les conflits de versions.
— Les interfaces COM sont immuables. En effet, les interfaces COM ne possèdent pas de version car chacune d'entre elles est unique, ce qui évite les conflits de versions entre anciens et nouveaux composants. Une nouvelle interface créée pour ajouter des fonctionnalités ou en modifier la sémantique est une nouvelle interface à laquelle est assigné un nouvel identifiant. Même si une opération est ajoutée ou si la sémantique d'une méthode existante est modifiée, nouvelle et ancienne interface n'entrent pas en conflit. De plus, deux interfaces peuvent partager la même implémentation. Par exemple, si une nouvelle interface ajoute un service à une interface existante, afin de supporter à la fois, anciens et nouveaux clients, les deux interfaces doivent coexister. Pour cela, l'ancienne interface sera implémentée comme un sous ensemble de l'implémentation de la nouvelle.

2.2.6 Identifiant unique GUID

COM identifie chaque interface et chaque classe de composant par un entier de 128 bits, unique à travers le monde (l'espace et le temps), appelé GUID (Globally Unique Identifier). Ces identifiants répondent à la définition d'UUID (Universally Unique IDentifier) faite par l'OSF DCE (Open Software Foundation's Distributed Computing Environment). Des noms lisibles sont associés mais leur portée est locale, on parle de CLSID (CLasS IDentifier) pour les classes et de IID (Interface IDentifier) pour les interfaces. Ces identifiants sont embarqués dans chaque composant et sont utilisés par le système pour garantir leur bonne connexion.

2.2.7 L'interface «IUnknown»

COM définit des interfaces spéciales qui implémentent des fonctionnalités essentielles. Tous les composants doivent implémenter l'interface «IUnknown» et toute autre interface comme par exemple OLE et COM dérive de «IUnknown».

```
interface IUnknown {

   virtual HRESULT QueryInterface (IID& iid, void** ppvObj) = 0;
   virtual ULONG   AddRef  () = 0;
   virtual ULONG   Release () = 0;

}
```

FIGURE 2.6 – *L'interface «IUnknown»*

«AddRef» et «Release» sont des méthodes de comptage des références des composants utilisant les interfaces d'un composant donné.

Tant qu'il est utilisé, un composant reste en mémoire, dès qu'il ne l'est plus, il se décharge de lui-même. La méthode «QueryInterface» permet d'obtenir des pointeurs sur les interfaces d'un composant, c'est par conséquent, un moyen pour les composants, de se renseigner sur

les services qu'ils implémentent. Une application désirant utiliser les services d'une interface demande au composant concerné si il la supporte. Suivant le code renvoyé par la méthode «QueryInterface» soit, le pointeur d'interface est renvoyé et peut être utilisé pour invoquer la méthode souhaitée soit une erreur est signalée à l'utilisateur.

```
LPLOOKUP *pLookup;
TCHAR szNumber[64];
HRESULT hRes;

// Call QueryInterface on the component object PhoneBook, asking for a pointer
// to the ILookup interface identified by a unique interface ID.

hRes = pPhoneBook->QueryInterface(IID_ILOOKUP,&pLookup);
if (SUCCEEDED(hRes))
{
  pLookup->LookupByName("Daffy Duck", &szNumber); // use ILookup interface pointer
  pLookup->Release(); // finished using the IPhoneBook interface
}
else
{
  // Failed to acquire ILookup interface pointer.
}
```

FIGURE 2.7 – *La méthode «QueryInterface»*

La figure 2.7 montre la méthode «QueryInterface» utilisée pour déterminer si un composant de la classe «PhoneBook» supporte l'interface «ILookup». Si l'appel à «QueryInterface» réussi, les méthodes «LookupByName» et «LookupByNumber» peuvent être invoquées.

A noter que la méthode «AddRef» n'est pas appelée explicitement, c'est la méthode «QueryInterface» qui l'effectue avant de renvoyer le pointeur d'interface.

2.2.8 La bibliothèque de composants COL

La bibliothèque de composants COL (Component Object Library) renferme les mécanismes de COM. Elle encapsule les tâches de mise en route, de connexion des composants et permet d'appeler les services de l'interface «IUnknown» au travers des processus.

Typiquement, quand une application souhaite créer un composant, elle passe son CLSID à la bibliothèque, celle-ci va rechercher dans un registre le code serveur associé. Si celui-ci est un fichier exécutable, la COL lance son exécution et attend qu'il enregistre sa fabrique de classes («Class Factory») via un appel à «CoRegisterClassFactory». Si le code serveur est une bibliothèque à liaison dynamique DLL («Dynamic Link Library»), la COL la charge et appelle «DllGetClassFactory» afin d'obtenir la fabrique de classes. Ensuite, COM utilise l'interface «IClassFactory» de la classe «Factory» pour créer une instance d'un composant et retourner, à l'application, un pointeur sur l'interface demandée. L'application appelante ne se soucie pas de la localisation du serveur, elle utilise simplement le pointeur d'interface qui lui est retourné pour communiquer avec le composant nouvellement créé. COM n'est pas qu'une spécification, il est implémenté sous Windows NT dans la bibliothèque de composants «ole32.dll».

2.2.9 COM résout la problématique des composants logiciels

La problématique des logiciels à base de composants soulève quatre problèmes :

— L'interopérabilité de base : Elle est fournie par la table des fonctions virtuelles qui définit un standard d'interface binaire pour l'appel de méthodes entre composants. Un appel entre composants est aussi performant qu'un appel de méthode C++.

— La gestion de version : De par sa conception, COM a éliminé les conflits de versions et le besoin d'une gestion centrale de celles-ci. En effet, l'usage d'interfaces immuables permet aux composants d'évoluer en préservant la compatibilité avec les anciens clients. Même s'il évolue, un composant continue à implémenter ses anciennes interfaces. Ce mécanisme permet l'évolution sans qu'il soit nécessaire de recompiler la partie cliente. COM permet également, aux composants en ayant connaissance, de profiter de nouvelles fonctionnalités. L'interface «IUnknown» implémentée par tous les composants permet, via le service «QueryInterface», d'interroger un composant pour connaître les interfaces qu'il implémente. Les interfaces peuvent également être réutilisées via le mécanisme d'héritage.

— L'indépendance face au langage de programmation : Tout langage permettant les appels implicites ou explicites de fonctions, via un pointeur, peut servir à créer des composants COM. Les composants peuvent être implémentés dans plusieurs langages et utilisés par des clients écrits dans différents langages. Car, contrairement aux langages orientés objet qui représentent des standards de code source, COM représente un standard d'objets binaires. C'est la raison de son indépendance face au langage de programmation. C'est un avantage fondamental d'une architecture de composants face à la programmation orientée objet (POO). Car les objets définis en POO interagissent uniquement avec des objets définis dans le même langage, ce qui limite nécessairement leur réutilisation.

— Transparence de l'interopérabilité : La problématique des logiciels à base de composants serait moindre si les interactions entre composants se produisaient dans un même processus. Or, les spécifications de COM intègrent, dès le début, le fait que l'interopérabilité des composants doit outrepasser la barrière des processus et des réseaux, les mécanismes associés étant mis en oeuvre par la bibliothèque de composants. En effet, les composants doivent pouvoir interagir, indifféremment, à l'intérieur d'un même processus, entre processus différents ou encore, entre différentes machines, à travers différents réseaux et ce, de façon transparente au niveau du code source.

Le standard binaire permet à COM d'intercepter un appel à un objet et, selon le cas, le remplacer par un appel interprocessus IPC (Inter Process Call) pour les composants situés dans un autre processus ou par un appel de procédure distante RPC (Remote Procedure Call) pour les composants situés sur une machine distante. A noter que nous englobons, dans l'appellation COM, la version locale et la version distribuée DCOM (Distributed COM) [18].

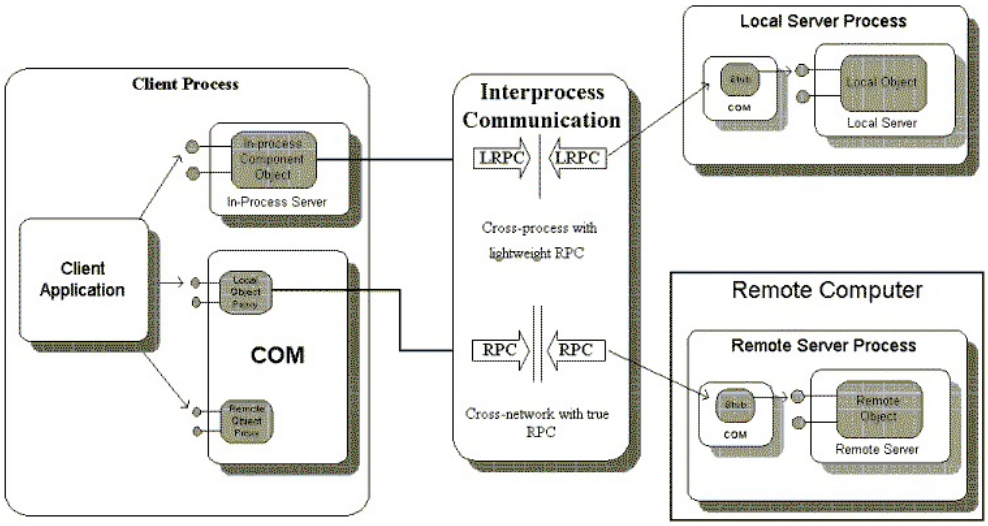


FIGURE 2.8 – *Transparence de l'interopérabilité COM*

Du point de vue du client, les composants sont accédés via des pointeurs d'interface. Pourtant, la portée d'un pointeur est toujours limitée à un processus. En fait, tout appel de fonction atteint, initialement, une partie de code située à l'intérieur du processus client. Si le composant se trouve dans le même processus, l'appel est alors direct, s'il se trouve à l'extérieur, l'appel atteint un objet COM, appelé «souche ou talon, appelé» ou «proxy», qui se chargera d'effectuer l'appel approprié suivant si l'objet se trouve sur la même machine ou à distance.

Du point de vue du serveur, tout appel de fonction est effectué via un pointeur sur une interface. Mais la portée d'un pointeur étant limitée à un processus, tout appel atteint une partie de code située à l'intérieur du processus serveur. Si le composant se trouve dans le même processus, l'appel est alors direct, s'il se trouve à l'extérieur, un objet COM, appelé «souche ou talont, appelant» ou «stub» réceptionne l'appel distant venant du «proxy» pour le transformer en un appel à l'interface du composant serveur.

2.2.10 Définition d'interfaces personnalisées

Pour définir une nouvelle interface, un développeur utilise le langage de définition des interfaces, MIDL (Microsoft Interface Definition Langage). Le compilateur IDL se charge de générer, pour l'application utilisant ces interfaces, les fichiers d'entête, le code source pour la création des objets «proxy» et «stub». MIDL est l'extension, développée par Microsoft, du langage IDL défini par l'OSF DCE, norme industrielle de calcul distribué basée sur RPC [19].

Présentons le fichier IDL utilisé pour définir l'interface personnalisée «ILookup» implémentée par le composant «PhoneBook».

```
[
  object,
  // Use the GUID for the ILookup interface
  uuid(c4910d71-ba7d-11cd-94e8-08001701a8a3),
  pointer_default(unique)
]
// ILookup interface derives from IUknown
interface ILookup : IUnknown
{
  // Bring in the supplied IUnknown IDL
  import "unknown.idl";
  // Define member function LookupByName
  HRESULT LookupByName([in] LPTSTR lpName, [out, string] WCHAR **lplpNumber);
  // Define member function LookupByNumber
  HRESULT LookupByNumber([in] LPTSTR lpNumber, [out, string] WCHAR **lplpName);
}
```

FIGURE 2.9 – *Définition de l'interface «ILookup»*

IDL est simplement un outil de commodité pour le concepteur d'interface et n'est pas central à l'interopérabilité COM. Il épargne aux développeurs la création manuelle des fichiers d'entête, des objets «proxy» et «stub» pour chaque environnement de programmation. IDL sert uniquement à la création d'interfaces personnalisées, la bibliothèque de composants fournit déjà les objets «proxy» et «stub» pour toutes les interfaces COM et OLE.

2.2.11 Synthèse

COM est une architecture générale de composants logiciels binaires permettant de connecter et de faire communiquer, de manière bien définie, des composants logiciels écrits dans différents endroits du monde, par différentes personnes, dans des langages de programmation différents. L'indépendance face au langage confère aux composants COM un bon niveau de réutilisation. Le caractère immuable des interfaces élimine le problème de la gestion des versions et le support des anciennes interfaces assure la compatibilité avec les anciens composants. L'interface «IUknown» permet la gestion du cycle de vie des composants et des ressources associées, mais elle permet également aux composants en ayant connaissance de profiter de leurs dernières évolutions. La transparence de l'interopérabilité permet de développer indifféremment des composants fonctionnant dans un ou plusieurs processus, sur une ou plusieurs machines d'un réseau. COM est un modèle permettant de développer des applications évolutives éventuellement distribuées et donc robustes. COM n'est pas qu'une spécification, il est implémenté sous Windows, Apple Macintosh et certaines versions d'UNIX. De plus amples informations peuvent être trouvées dans [17].

2.3 L'architecture d'objets distribués CORBA

CORBA (Common Object Request Broker Architecture) est une spécification de l'OMG (Object Management Group). Fondée en 1989, l'OMG est une organisation internationale rassemblant plus de huit cents membres de l'industrie logicielle. Comme le spécifie l'OMA (Object Management Architecture), son but est de promouvoir et de développer la théorie et la pratique des technologies orientées objet en spécifiant un cadre de travail commun. L'OMA définit l'infrastructure conceptuelle sur laquelle sont basées toutes les spécifications de l'OMG. Ses

objectifs premiers sont la réutilisabilité, la portabilité et l'interopérabilité, dans des environnements hétérogènes et distribués, des logiciels à base d'objets.

2.3.1 Objectif

L'objectif de CORBA est de définir un cadre commun permettant à des composants logiciels, délivrés par différents fournisseurs et distribués sur différentes machines d'inter-opérer. CORBA assume l'indépendance quant au langage d'implémentation, à la plateforme d'exécution et à l'hétérogénéité des réseaux. Pour ce faire, CORBA définit des mécanismes permettant aux objets, écrits dans différents langages de programmation et fonctionnant sur différentes plateformes, d'émettre des requêtes et de recevoir des réponses à travers des réseaux hétérogènes. Ces mécanismes forment un bus logiciel au coeur de CORBA, l'ORB (Object Request Broker). CORBA spécifie également l'interopérabilité inter ORB permettant l'intégration de différents systèmes d'objets tel que COM/DCOM et fonctionnant dans des environnements différents tel que l'OSF DCE. Pour cela, CORBA définit le protocole de communication IOP (Inter ORB Protocol) décliné en GIOP (General Inter ORB Protocol) et IIOP (Internet Inter ORB Protocol) ainsi que le protocole ESIOP (Environment Specific Inter ORB Protocol) tirant parti de l'environnement OSF DCE.

2.3.2 Le modèle objet

Nous présentons ici le modèle objet qui sous tend CORBA. Ce modèle est concret et dérive du modèle abstrait «Core Object Model» défini dans l'OMA. Un système objet est une collection d'objets qui isole le demandeur de services (le client) du fournisseur (le serveur) par une interface bien définie encapsulant l'implémentation des services. Les clients sont donc isolés du code exécutable et de toute représentation des données. Le modèle objet de CORBA est l'exemple d'un modèle objet classique où les objets clients envoient des messages et où les objets serveurs interprètent les messages pour décider des services à délivrer. Dans le modèle classique, un message identifie un objet et zéro ou plusieurs paramètres mais dans la plupart des modèles un paramètre identifiant l'opération à effectuer est nécessaire. L'interprétation du message consiste en la sélection, par l'objet ou par l'ORB, de la méthode appropriée à l'opération spécifiée.

Les objets

Un système objet comprend des entités appelées objets. Un objet est une entité identifiable, encapsulée capable de fournir des services et de recevoir des requêtes. Les objets effectuant des requêtes et les objets fournisseurs de services sont respectivement appelés objets clients et objets serveurs. Un objet est dit client ou serveur dans un contexte donné mais il est généralement à la fois client et serveur.

Les requêtes

Les clients demandent des services en émettant des requêtes. Le terme requête est couramment employé pour évoquer la suite logique d'événements se produisant entre l'initiation et la terminaison d'une requête. Les informations associées à une requête sont l'objet cible, l'opération demandée, zéro ou plusieurs paramètres et un contexte optionnel. La forme de la requête est décrite dans un langage particulier, IDL (Interface Definition Language). Les interfaces renseignées en IDL permettent au programmeur de construire ses requêtes et d'invoquer

les services qu'il souhaite. Ce type d'invocation est dit statique parce que l'interface de l'objet cible est connue durant la compilation (cf. SII). Mais ceci n'est pas indispensable car certains mécanismes permettent de consulter le ou les interfaces d'un objet durant l'exécution. Celles-ci sont alors encapsulées dans des objets et transmises au client qui peut alors construire sa requête et invoquer dynamiquement le service souhaité (cf. DII). Une requête possède des paramètres d'entrée communiqués à l'objet cible et de sortie retournés au client, ceux-ci sont soit des valeurs de types IDL soit des références d'objet.

Création et destruction

Du point de vue du client il n'y a pas de mécanismes spéciaux pour créer et détruire un objet. La création et la destruction des objets sont le résultat de requêtes. Le résultat d'une création est une référence à l'objet nouvellement créé.

Les types

CORBA supporte les types de base et structurés communs à la plupart des langages de programmation.

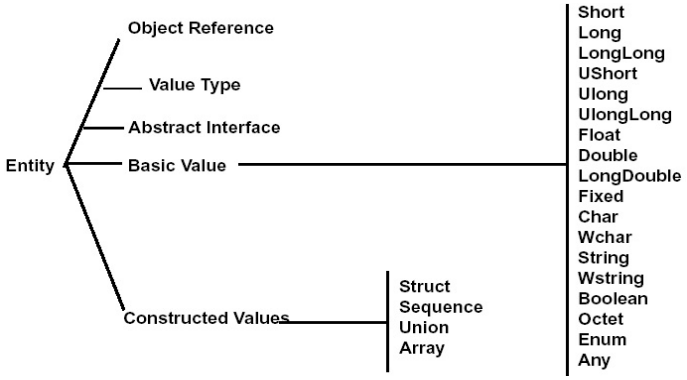


FIGURE 2.10 – *Types supportés*

Les interfaces

Une interface fournit une description syntaxique de la façon d'accéder, via un ensemble d'opérations, à un service délivré par un objet supportant cette interface. Les interfaces sont spécifiées en IDL et le mécanisme d'héritage permet aux objets de supporter plusieurs interfaces.

Les types d'objets

Un type d'objet est une combinaison de types structurés et d'interfaces, c'est une description d'un ensemble d'opérations et d'état accessibles au client. Une instance d'un type d'objet peut être implémentée dans un langage de programmation. Une interface abstraite est un objet qui, durant l'exécution, représente soit une interface classique soit un type d'objet.

Les opérations

Une opération est une entité identifiable qui dénote une prestation de service. Les opérations sont dites potentiellement génériques, ce qui signifie qu'une opération peut être invoquée de la même façon sur des objets, d'implémentations différentes et qu'il en résulte des comportements

différents. La généricité ou polymorphisme est réalisée via l'héritage d'interface et le découplage entre interface et implémentation.

Les implémentations

L'implémentation d'un objet est responsable de la mise en oeuvre des services qu'il délivre. Elle mène les activités calculatoires qui réalisent son comportement. Ces activités incluent le calcul du résultat de la requête et la mise à jour de l'état du système. Ce processus peut faire intervenir d'autres requêtes.

Les méthodes

Un service est accompli par l'exécution de code agissant sur des données. Ces données faisant partie de l'état du système, l'exécution d'un service peut donc changer l'état de ce système. Le code qui accomplit un service s'appelle une méthode. Quand un client procède à une requête, une méthode de l'objet cible est appelée. Les paramètres d'entrée lui sont transmis et les paramètres de sortie sont, eux, retournés au demandeur.

2.3.3 Principe de l'ORB

En assurant, le transport des objets et l'invocation de méthodes distantes l'ORB est au coeur de CORBA. Lors d'une requête, il est chargé de localiser l'implémentation de l'objet et de lui communiquer les données caractérisant cette requête. L'interface que voit le client est complètement indépendante de la localisation de l'objet, du langage de programmation et de tous autres aspects non reflétés par l'interface.

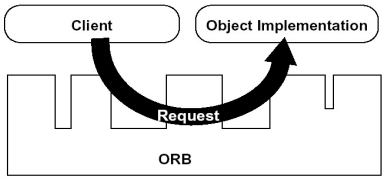


FIGURE 2.11 – *Requête envoyée à travers l'ORB*

La figure 2.11 montre une requête envoyée par un client à l'implémentation d'un objet. Le client est l'entité souhaitant effectuer une opération sur l'objet et l'implémentation de l'objet est le code et les données qui implémentent actuellement l'objet.

2.3.4 Structure de l'ORB

Pour faire une requête, le client peut utiliser l'interface d'invocation statique SII (Static Invocation Interface) ou l'interface d'invocation dynamique DII (Dynamic Invocation Interface).

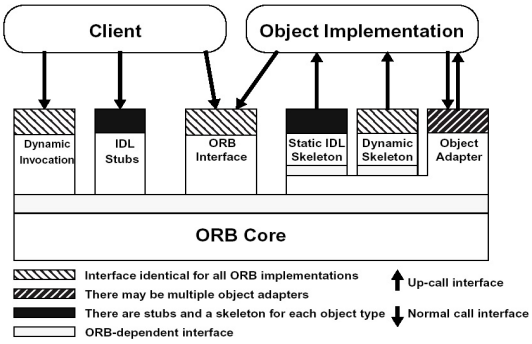


FIGURE 2.12 – *Structure de l'ORB*

La figure 2.12 montre la structure d'un ORB. Ses interfaces sont représentées par des rectangles hachurés et les flèches indiquent si l'ORB est appelé ou si il effectue un appel à travers l'interface.

Via l'interface d'invocation statique la requête passe par un objet spécifique à l'interface de l'objet cible, la souche client ou «stub» IDL. En revanche, l'interface d'invocation dynamique est indépendante de l'interface de l'objet cible. Le client peut également faire appel à certains services de l'ORB. Les souches IDL sont générées lors de la compilation à partir des documents IDL relatifs à l'interface de l'objet cible. L'interface d'invocation dynamique elle, utilise les services du dépôt d'interfaces IR (Interface Repository) pour consulter l'interface de l'objet cible et permettre au client de construire sa requête dynamiquement. Le dépôt d'interfaces représente les interfaces comme des objets et permet un accès à ces composants pendant l'exécution.

Coté client

Pour effectuer une requête le client doit posséder une référence à l'objet cible, connaître son type et l'opération désirée. Le client lance la requête en appelant la routine de la souche spécifique à l'objet ou en construisant la requête dynamiquement.

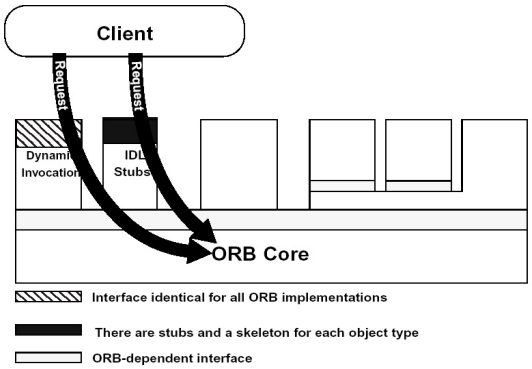


FIGURE 2.13 – *Un client utilisant une souche ou l'interface d'invocation dynamique (DII)*

Les interfaces d'invocation de type souche et dynamique satisfont la même sémantique et le receveur du message ne sait pas comment la requête a été invoquée.

Coté serveur

Du coté du serveur, l'implémentation reçoit la requête soit comme un appel d'une souche d'implémentation ou «skeleton» statique, générée à partir des documents IDL de l'objet cible, soit comme un appel d'une souche dynamique. L'implémentation peut également appeler les services de l'adaptateur d'objets OA (Object Adapter) ainsi que ceux de l'ORB.

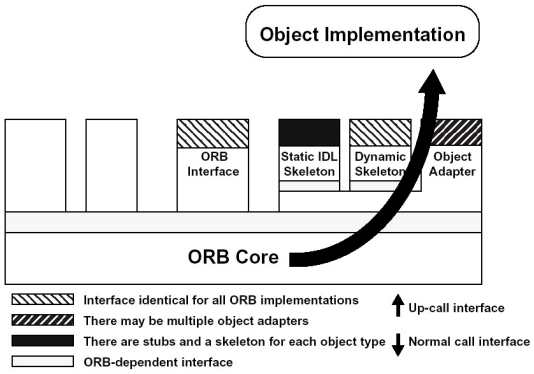


FIGURE 2.14 – *L'implémentation d'un objet recevant une requête*

L'ORB localise le code de l'implémentation appropriée et lui transmet les paramètres et le contrôle à travers une souche d'impémentation IDL ou dynamique, tous deux spécifiques à l'interface et à l'adaptateur d'objets. En effectuant la requête l'implémentation peut obtenir des services de l'ORB à travers l'adaptateur d'objets. L'implémentation doit choisir quel adaptateur d'objets utiliser. Cette décision est prise suivant la nature du service dont l'implémentation a besoin. Quand la requête est complète les valeurs de sortie et de contrôle sont retournées au client.

Processus d'installation

Les interfaces sont définies en IDL et, ou dans le dépôt d'interfaces. Ces définitions servent à générer les souches du coté des clients et des implémentations. L'information sur l'implémentation est fournie durant la période d'installation, stockée dans le dépôt d'implémentations et utilisée durant la livraison de la requête.

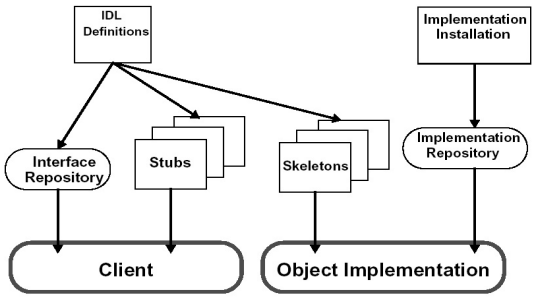


FIGURE 2.15 – *Dépôt d'interfaces et d'implémentations*

La figure 2.15 montre comment les informations relatives aux interfaces et implémentations sont rendues disponibles aux clients et aux implémentations.

2.3.5 Architecture de l'ORB

L'architecture d'un ORB n'est pas nécessairement implémentée sous forme d'un seul composant unique. Chaque implémentation d'ORB fournissant une interface appropriée est acceptable. Cette interface est organisée en trois catégories :

— Opérations communes à tous les ORB.
— Opérations spécifiques à des types particuliers d'objets.
— Opérations spécifiques à des styles particuliers d'implémentations.

Pour fournir un ensemble de services à des clients et implémentations dont les propriétés diffèrent, divers choix d'implémentation peuvent être faits. Il existe diverses implémentations d'ORB avec différentes représentations des références et différents mécanismes d'invocation. Aussi, il est possible qu'un client ait accès à deux références d'objets gérées par deux implémentations différentes d'ORB. Dans ce cas, il est de la responsabilité des ORB et non du client de distinguer leurs références respectives.

Le noyau de l'ORB fournit les représentations de base, des objets et des communications des requêtes. Les mécanismes de CORBA sont structurés sous forme de composants fonctionnellement dépendants du noyau, ce qui permet de fournir des interfaces et de masquer les différences entre les noyaux.

Les clients

Le client d'un objet accède à une référence de l'objet et invoque des opérations sur cet objet. Un client connaît la structure logique de l'objet uniquement grâce à son interface et expérimente son comportement au travers d'invocations. Bien que nous considérions un client comme étant un programme ou un processus lançant une requête à un objet, il est important de reconnaître, que quelque chose est un client relativement à un objet particulier. Par exemple, l'implémentation d'un objet peut être le client d'un autre objet.

Les clients voient généralement les objets et les interfaces de l'ORB à travers la projection ou «mapping» d'un langage mettant l'ORB au niveau du programmeur. Les clients sont grandement portables et doivent fonctionner avec toutes les instances qui implémentent l'interface désirée, sans modification du code source et ce, sur chaque ORB supportant la projection du langage désiré. Les clients n'ont aucune connaissance de l'implémentation de l'objet, de l'adaptateur d'objets utilisé par l'implémentation ou encore, de l'ORB y accédant.

Les implémentations

L'implémentation d'un objet fournit la sémantique de l'objet en définissant ses données d'instance et le code de ses méthodes. Souvent l'implémentation utilise d'autres objets et des logiciels additionnels pour implémenter le comportement de l'objet. Dans certains cas, la première fonction de l'objet est d'encapsuler des choses qui ne sont pas des objets.

Une variété d'implémentations d'objets peut être supportées, comme des serveurs séparés, des bibliothèques, une application encapsulée, une base de données orientée objet, etc. A travers l'usage d'adaptateurs d'objets traditionnels, il est possible de supporter virtuellement tous les styles d'implémentations d'objets. Généralement, l'implémentation des objets ne dépend pas de l'ORB ou de la façon qu'a le client de l'invoquer. Les implémentations d'objets doivent sélectionner les interfaces des services qui dépendent de l'ORB par le choix d'adaptateurs d'objets.

Les références

Une référence est l'information nécessaire pour dénommer un objet à l'intérieur d'un ORB. Les clients et les implémentations ont une notion opaque des références à travers la projection du langage et sont donc isolés de leur représentation. Deux implémentations d'ORB peuvent différer dans leurs choix de représentation des références. Les références des objets clients sont uniquement valides durant le cycle de vie de celui-ci. Pour un langage donné, tout ORB doit fournir le même projection des références. Cela permet à un programme écrit dans un langage particulier d'accéder aux références indépendamment d'un ORB particulier. La projection doit fournir d'autres façons d'accéder aux références de manière typée pour le confort du programmeur. Il existe une référence particulière qui se distingue de toutes les autres en ne désignant aucun objet.

Langage de définition d'interfaces (IDL)

L'OMG IDL définit les types d'objets en spécifiant leurs interfaces. Une interface consiste en un ensemble d'opérations nommées et acceptant des paramètres. Bien qu' IDL fournisse le cadre conceptuel pour décrire les objets manipulés par l'ORB, il n'est pas indispensable au fonctionnement de l'ORB. Tant que l'équivalent des informations IDL est disponible sous forme de routines souches ou de dépôt d'interfaces, l'ORB doit fonctionner correctement. IDL est le moyen par lequel une implémentation particulière communique, à ses clients potentiels, quelles opérations sont disponibles et comment elles doivent être invoquées. A partir des définitions IDL il est possible de projeter les objets CORBA dans un langage de programmation particulier ou autres systèmes à base d'objets.

La projection d'IDL

Différents langages de programmation, qu'ils soient orientés objet ou non, peuvent préférer accéder aux objets CORBA de différentes manières. Pour les langages orientés objet il serait souhaitable d'appréhender les objets CORBA comme des objets du langage. Même pour les langages non orientés objet il est judicieux de masquer la représentation exacte par l'ORB, des références d'objets, des noms de méthodes, etc. Une projection d'IDL vers un langage de programmation particulier doit être identique pour toutes implémentations d'ORB. La projection d'un langage inclut la définition des types de données spécifiques au langage et les interfaces des procédures d'accès aux objets au travers de l'ORB. Ceci inclut l'interface de la souche client (ce qui n'est pas nécessaire pour un langage orienté objet) l'interface d'invocation dynamique, le squelette d'implémentation, les adaptateurs d'objets et l'interface de l'ORB. La projection d'un langage définit aussi l'interaction entre les invocations d'objets et les processus légers ou «threads» de contrôle du client et de l'implémentation. La projection la plus courante fournit des appels synchrones, c'est-à-dire que l'appel d'une routine retourne une fois l'opération terminée. Des projection additionnelles peuvent être fournis pour permettre à un appel d'être lancé et que le contrôle soit rendu au programme. Dans de tels cas des routines additionnelles, spécifiques au langage, sont fournies pour synchroniser les processus légers de contrôle avec l'invocation de l'objet.

Les souches client (stubs)

Généralement les souches client présentent l'accès aux opérations d'un objet, définies en IDL, de sorte qu'il soit aisé aux programmeurs de prédire s'ils sont familiarisés avec IDL et

la projection d'un langage particulier. Les souches appellent le reste de l'ORB en utilisant des interfaces privées et optimisées pour le noyau d'un ORB particulier. A chaque ORB disponible correspond une souche différente. Dans ce cas il est nécessaire pour l'ORB et la projection du langage de coopérer pour associer les souches adéquates avec la référence d'un objet particulier.

L'interface d'invocation dynamique (DII)

Cette interface permet la construction dynamique d'invocations d'objets. Contrairement à l'appel d'une routine souche spécifique à une opération particulière sur un objet, un client peut spécifier l'objet à invoquer, l'opération à effectuer et l'ensemble de ses paramètres à travers une séquence d'appels. Le code client doit fournir les informations sur l'opération à effectuer ainsi que les types des paramètres passés (éventuellement obtenus durant l'exécution via le dépôt d'interfaces ou une autre source). La nature de l'interface d'invocation dynamique peut varier substantiellement d'une projection à l'autre.

Les souches d'implémentation (skeletons)

Suivant la projection et éventuellement suivant l'adaptateur d'objets, chaque type d'objets possède une interface sur les méthodes de son implémentation. Cette interface est conforme aux méthodes de l'implémentation et est appelée par l'ORB via la souche d'implémentation (skeleton). L'existence d'une souche n'implique pas l'existence d'une souche client, les clients peuvent aussi faire des requêtes via l'interface d'invocation dynamique. Il est possible d'écrire un adaptateur d'objets qui n'utilise pas les souches d'implémentation pour invoquer ses méthodes. Par exemple, il est possible de créer les implémentations de façon dynamique pour les langages tel que Smalltalk.

L'interface des souches d'implémentation dynamiques (DSI)

Cette interface permet la manipulation dynamique des invocations d'objets. Comparée à l'accès via une souche d'implémentation spécifique à une opération particulière, l'implémentation d'un objet est atteinte au travers d'une interface fournissant un accès au nom et aux paramètres de l'opération de manière analogue à l'interface d'invocation dynamique côté client. Les paramètres peuvent être déterminés de façon statique ou dynamique via le dépôt d'interfaces. Le code de l'implémentation doit fournir à l'ORB les descriptions de tous les paramètres de l'opération et l'ORB fournir les valeurs de tous les paramètres d'entrées nécessaires à l'exécution de l'opération. Après avoir exécuté l'opération, le code de l'implémentation doit fournir, à l'ORB, les valeurs de tous les paramètres de sortie ou une exception. La nature de l'interface de la souche dynamique peut varier substantiellement d'une projection de langage ou d'un adaptateur d'objets à l'autre mais sera toujours appelée par l'ORB. Les souches dynamiques peuvent être invoqués à la fois au travers des souches client et de l'interface d'invocation dynamique, du moment que la requête est bien construite.

Les adaptateurs d'objets (OA)

Un adaptateur d'objets est le moyen pour une implémentation d'accéder aux services de l'ORB. Chaque adaptateur doit posséder une interface appropriée à chaque type d'objets. Les services fournis par l'ORB à travers un adaptateur d'objets sont souvent la génération, l'interprétation des références d'objets, l'invocation de méthodes, la sécurité des interactions, l'activation et la désactivation des objets et des implémentations, la projection des références d'objets et l'enregistrement des implémentations. Les différentes granularités d'objets, cycles

de vie, règles et styles d'implémentations et autres propriétés rendent difficile, pour le noyau de l'ORB de fournir une seule interface pratique et efficace pour tous les objets. Grâce aux adaptateurs, il est possible pour l'ORB, de cibler des groupes particuliers d'implémentations ayant des besoins similaires et de leur fournir des interfaces qui leur soient adaptées.

L'interface de l'ORB

L'interface de l'ORB est la même pour tous les ORB, elle ne dépend ni des interfaces ni des adaptateurs d'objets. Parce que la plupart des fonctionnalités d'un ORB sont fournies au travers d'adaptateurs, de souches appelantes, appelées ou de l'interface d'invocation dynamique, il y a seulement quelques opérations communes à tous les objets. Ces opérations sont aussi bien utiles aux clients qu'aux implémentations.

Le dépôt d'interfaces (IR)

Le dépôt d'interfaces est un service qui fournit des objets persistants qui représentent l'information IDL sous une forme disponible durant l'exécution. Les informations du dépôt d'interfaces sont utilisées par l'ORB pour effectuer les requêtes. Ces informations permettent à un programme, rencontrant un objet pour lequel l'interface n'était pas connue lors de la compilation, de déterminer quelles opérations sont valides afin de les invoquer. En plus de son rôle dans le fonctionnement de l'ORB, le dépôt d'interfaces est le lieu commun pour stocker des informations supplémentaires, associées aux interfaces des objets de l'ORB. Par exemple, des informations de déboguage, des bibliothèques de souches appelantes et appelées, des routines pouvant formater ou rechercher des types particuliers d'objets peuvent être associées avec le dépôt d'interfaces.

Le dépôt d'implémentations

Le dépôt d'implémentations contient des informations permettant à l'ORB de localiser et d'activer les implémentations. Bien que la plupart des informations du dépôt d'implémentations soient spécifiques à un ORB ou à l'environnement d'exploitation, le dépôt d'implémentations est le lieu conventionnel pour enregistrer de telles informations. Habituellement, l'installation des implémentations et le contrôle des règles relatives à leur activation et à leur exécution sont faits à travers les opérations du dépôt d'interfaces.

En plus de son rôle dans le fonctionnement de l'ORB, le dépôt d'interfaces est un lieu commun pour stocker les informations supplémentaires associées aux implémentations des objets de l'ORB. Par exemple, les informations de déboguage, de contrôle administratif, d'allocation des ressources, de sécurité, etc. doivent être associées avec le dépôt d'implémentations.

2.3.6 Synthèse

CORBA est la spécification ouverte d'une architecture générale permettant l'interopérabilité des logiciels à base d'objets distribués sur des machines connectées par des réseaux hétérogènes. Contrairement aux spécifications de COM, les spécifications de CORBA sont libres. Alors que COM est fourni, avec le système d'exploitation Windows, dans la bibliothèque de composants, l'implémentation de CORBA réside dans l'implémentation d'un ORB. Hier cantonné à la plateforme Intel/Windows, des implémentations UNIX de COM existent aujourd'hui. De plus, une section des spécifications CORBA traite de l'interfaçage des composants COM. Le principe d'abstraction entre interfaces et implémentations permet aux applications de s'abstraire des

spécificités des architectures matérielles et logicielles. Cependant, l'abstraction n'est pas complète tant que les objets sont compilés. En effet, ces derniers sont compilés pour une plateforme d'exécution donnée et ne peuvent en migrer. Dans ce contexte, la compilation est un processus discutable face à l'interprétation.

2.3.7 CORBA et Java

En effet, la compilation confère à l'implémentation d'un objet des performances optimums mais un faible niveau de portabilité. Un objet interprété, tels que les objets du langage Java, est, au contraire, moins performant mais grandement portable. Contrairement aux objets compilés les objets Java peuvent migrer sur toutes plateformes hétérogènes implémentant la machine virtuelle Java. Il est important de noter que la technologie Java repose, elle aussi, sur le principe d'abstraction. En effet, celle-ci repose sur l'abstraction du micro processeur, la machine virtuelle, et l'abstraction de l'environnement, le noyau d'éxecution ou «runtime». Aussi, il est possible d'améliorer les performances des objets en intégrant les routines critiques au noyau d'éxecution. Ce gain est possible au prix du développement d'un «runtime» pour chaque plateforme. Enfin, une version de CORBA est intégrée, en standard, dans le noyau d'éxecution Java. Une architecture de composants logiciels basée sur CORBA existe également dans l'environnement Java sous le nom d'EJB (Entreprise Java Beans). L'approche Java est intéressante dans un contexte CORBA car bon nombre de fonctionnalités requises par CORBA font partie des caractéristiques de Java.

La gestion du cycle de vie

C'est le cas de la gestion du cycle de vie des objets, assurée de façon transparente par le ramasse miettes ou «garbage collector». Un objet est détruit dés qu'il n'est plus référencé, ce qui décharge le programmeur de cette tâche.

Les références

En Java, les objets sont manipulés uniquement par référence, contrairement à C++ où un objet peut être manipulé par copie, par pointeur et par référence. De plus, les notions de passage par pointeur et par référence ne sont pas sans ambiguïtés. La notion de référence en C++ étant un moyen syntaxique pratique dissimulant un passage par pointeur.

Le polymorphisme

Contrairement à C++, Java repose sur une stratégie de liaison tardive ou «late binding» entre opérations et méthodes d'une classe, alors que C++ repose sur une stratégie précoce ou «early binding». Ce qui confère, aux opérations Java, la propriété de polymorphisme implicite, contrairement à C++, ou celle-ci doit être explicitée à l'aide du mot clef «virtual».

Les mécanismes d'abstraction

La sémantique des opérations abstraites (opérations polymorphes ne possédant pas de corps dans la classe mère) est également améliorée grâce au mot clef «abstract». En C++, celle-ci s'exprime par la présence du mot clef «virtual», en début, et d'un «=0», en fin de déclaration, celle-ci est alors dite virtuelle pure. En Java, la notion de polymorphisme étant implicite, le mot clef «virtual» a disparu et la combinaison «virtual» et «=0» est remplacée, par «abstract». En C++, une classe ne possédant que des méthodes virtuelles pures est dite abstraite. En Java,

une classe abstraite ne possède que des méthodes abstraites ! Une classe ne possédant que des méthodes abstraites et hébergeant des données membres, est une classe abstraite et une classe ne possédant que des méthodes abstraites et n'hébergeant pas de données membres est une interface.

Les interfaces

La notion d'interface fait partie intégrante de la sémantique du langage. En C++, aucune syntaxe ne fixe cette sémantique. La syntaxe Java, par la présence des mots clef «abstract» et «interface» permet d'exprimer explicitement ces concepts.

La migration d'objets

La migration d'objets CORBA repose, dans l'environnement Java, sur les mécanismes de sérialisation des objets Java. Comme Java permet de représenter dynamiquement les classes d'objets par des objets de la classe «class», il est également possible de transférer des classes d'objets et d'en instancier des objets dynamiquement. Le compilateur Java, lui-même, peut être instancié afin de créer dynamiquement, à partir de leurs codes sources, de nouvelles classes permettant, après intégration, l'évolution des applications.

Le mécanisme de réflexion

Java offre également le mécanisme dit de réflexion. Celui-ci permet d'interroger dynamiquement les classes et les objets Java sur les méthodes qu'ils supportent et ainsi, d'instancier les objets et d'invoquer leurs méthodes. Ainsi, l'interface d'invocation dynamique de CORBA est implémentée, dans l'environnement Java, via le mécanisme de réflexion.

2.4 L'interopérabilité des systèmes d'IAO

Nous avons présenté deux approches de l'interopérabilité logiciel, le standard d'échange de données du produit STEP puis les approches orientées objet, à travers les architecture de médiateurs ou «middleware» COM et CORBA. Nous allons récapituler ces deux approches en essayant d'en extraire les fondements.

STEP fut créé en réponse aux problèmes d'échange de données dans le domaine de la CAO. En effet, la diversité des logiciels, intervenant dans le cycle de développement du produit, entraîne la diversité des représentations. Des solutions d'échanges spécifiques à chaque couple de logiciels n'étant pas gérables [20], [2], des formats neutres apparurent. STEP propose l'harmonisation des représentations du produit dans chaque corps de métier sous forme de schéma EXPRESS et propose un format neutre d'instanciation. STEP est ainsi le plus récent et le plus abstrait des formats neutres. Cependant, sa mise en oeuvre se révèle complexe car elle implique, pour chaque corps de métier (cf. protocoles d'applications), l'écriture de nombreuses bibliothèques de schémas et nécessite leur interfaçage avec les modèles internes à chaque logiciel.

En résumé, l'approche STEP de l'interopérabilité est donc focalisée sur les données, la représentation des objets du domaine. Elle propose d'implémenter dans le langage EXPRESS des bibliothèques de schémas métiers normalisés, définit la syntaxe d'instanciation des dits schémas et l'interface SDAI d'accès à ces données. Finalement, STEP définit des modèles normalisés et chaque logiciel est responsable de leur interfaçage avec ses modèles internes, au moyen de l'interface SDAI. STEP est donc une tentative de normalisation des données du produit.

L'approche orientée objet, au contraire, focalise sur l'aspect fonctionnel plutôt que sur l'aspect représentation. Ainsi, un objet est vu comme une entité fonctionnelle rendant un certain nombre de services et non comme une structure de données même si celui-ci possède la plupart du temps des données membres représentant son état interne. Aussi, des objets offrant des fonctionnalités similaires peuvent posséder différentes représentations internes. C'est l'abstraction des représentations autorisé par le principe d'encapsulation des données qui en interdit l'accès immédiat et le principe d'encapsulation des traitements ou polymorphisme qui dissocie l'interface d'un service, l'opération, de son implémentation, la méthode. Ainsi, la différence fondamentale entre entités de base de données et classes d'objets sont la présence de méthodes modélisant le comportement de l'objet tout en encapsulant ses données membres. Aussi, le polymorphisme permet l'encapsulation des traitements, donc l'abstraction complète de la représentation.

Ce constat n'est pas sans répercussions sur l'aspect méthodologique. Alors que STEP utilise IDEF1 ou EXPRESS_G pour la structuration des données, l'approche orientée objet utilise des méthodes adaptées à sa vision, comme UML [21], [22]. La méthode IDEFØ, par contre, reste utile à l'expression des besoins informatifs.

Il est donc naturel que la vision orientée représentation de l'approche STEP ait donner lieu à un standard d'échange de données. A notre avis, l'échange de données s'intègre dans une problématique plus générale, l'interopérabilité des systèmes. La capacité que les systèmes ont à coopérer et donc à échanger des données mais aussi des services. Il semble que les principes orientés objet soient en adéquation avec la problématique de l'interopérabilité et de l'échange de données.

Alors que STEP s'attache à définir et à normaliser le modèle produit sous forme de schémas relationnels interfacés avec les modèles natifs, il normalise la représentation. L'interface SDAI n'est en aucun cas assimilable à une interface CORBA puisqu'elle propose un ensemble d'opérations ne respectant pas le principe d'encapsulation. En effet, SDAI propose, par exemple, des opérations permettant de consulter et de modifier la valeur de n'importe quel attribut de n'importe quelle instance d'entité. L'approche orientée objet, au contraire, s'abstrait des représentations et s'attache à définir l'ensemble des fonctionnalités, ou services attendus, les interfaces que doivent implémenter les objets. C'est l'approche prise, récemment, par le comité Mfg DTF (Manufacturing Domain Task Force) de l'OMG en réponse au manque d'interopérabilité des systèmes d'IAO (Ingénierie Assistée par Ordinateur).

2.4.1 L'Approche de l'OMG Mfg DTF

En tant que commission chargée des technologies, spécifiques à un domaine, en anglais OMG DTC (Domain Technology Committee), la commission Mfg DTF s'attache à spécifier les technologies relatives à la production industrielle. Ses objectifs : intégrer, dans les différents processus de l'entreprise, une grande variété de logiciels. Cela passe par l'amélioration de l'interopérabilité des systèmes d'ingénierie et la définition d'un modèle du produit couvrant les différentes phases de son développement, voir tout son cycle de vie. En réponse au manque d'interopérabilité, le groupe de travail PPE (Product and Process Engineering Working Group) est chargé de normaliser les fonctionnalités des logiciels d'ingénierie au moyen d'interfaces CORBA. Il s'attache, en premier lieu, à normaliser les services des modeleurs géométriques, les services CAO ou «CAD Services» [23]. Ceux-ci doivent, en effet, définir les fondements d'interopérabilité nécessaires au

développement de services de plus haut niveau. Les premiers visés sont les services de gestion des données du produit, ou «PDM enablers» [24]. D'autres groupes travaillent à la spécification de services dans le domaine de la simulation distribuée DSS (Distributed Simulation) [25].

2.4.2 Cadre méthodologique

Comme il en a été discuté dans [26] les travaux de spécification doivent suivre le modèle OMG's MDA (Model Driven Architecture), l'architecture doit être déterminée par les modèles. Dans ce cadre, c'est le langage de modélisation orienté objet UML qui est adopté. Cependant, les travaux de l'OMG concernant les services de GDT ne sont pas sans rapport avec les travaux de la communauté STEP. Des divergences méthodologiques existant entre ces deux communautés, des efforts d'harmonisation sont nécessaires [27].

La gestion de l'information des entreprises est une discipline fondamentale qui nécessite des normes permettant l'interopérabilité des systèmes. Les principaux auteurs de telles normes sont la communauté STEP (officiellement ISO TC184/SC4) et l'OMG. Ces derniers développent et font la promotion de leurs normes respectives. Même si leurs objectifs sont proches, leurs approches diffèrent quant à leurs niveaux d'abstraction, leurs caractéristiques opérationnelles. Alors que l'OMG définit les interfaces des services offerts par les SGDT dans un environnement distribué et orienté objet, la norme STEP définit un standard de représentation des données du produit. Ce constat fait de STEP une norme importante pour la commission Mfg DTF. En effet, les travaux sur STEP ont permis de modéliser la sémantique de plusieurs domaines de la production industrielle et il serait dommage que ceux-ci ne soient pas réutilisés. Seulement, pour représenter ses modèles, la communauté STEP utilise le langage EXPRESS (ISO-10303-11) et l'OMG utilise le langage de modélisation unifié UML (Unified Modeling Language). Par conséquent, le Mfg DTF a encouragé le groupe chargé de l'analyse et de la conception ADTF (Analysis and Design Task Force) d'accommoder les sémantiques EXPRESS et UML. Pour ce faire, le méta modèle d'EXPRESS fut exprimé en UML et inversement [28], [29], [30]. Une équipe formée de représentants du NIST, d'IBM et d'EuroSTEP, est chargée de coopérer avec toutes équipes concernées par l'harmonisation entre UML et EXPRESS.

2.4.3 Les services CAO

Le projet «CAD services» regroupant, entre autres, des membres des sociétés Boeing, IBM/Dassault Systèmes, Ford, NASA, NIST, Unigraphics, Open Cascade, etc. s'intéresse à définir les interfaces des systèmes délivrant des informations géométriques et topologiques aux applications et outils de simulation et de fabrication. L'objectif n'est pas de définir des structures de données de bas niveau, mais d'établir une série d'interfaces de haut niveau. En effet, afin d'éviter les problèmes de conversion de données, le projet «CAD services» définit les interfaces CORBA chapeautant les fonctionnalités natives des modeleurs, leur conférant finalement, un ensemble de services de conception communs, tout en laissant à leurs éditeurs, la liberté d'implémentation.

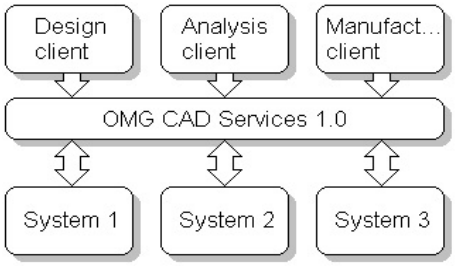


FIGURE 2.16 – *«OMG CAD Services»*

Les aspects cruciaux retenus sont : la capture de l'intention de conception, le lien avec la fabrication et la conception collaborative. L'architecture doit être évolutive afin d'intégrer de nouvelles fonctionnalités. Les spécifications des services de gestion des données techniques, ou «PDM Enablers» seront implémentées en tant que couches supérieures aux services CAO. Les services CAO représentent donc les fondations des services de plus haut niveau.

A ce jour, quelques prototypes implémentent les services CAO, c'est le cas du serveur CAO «CADServer» développé par Ford sur la base du modeleur I-DEAS. La société Open Cascade S.A. du groupe EADS Matra Datavision développe actuellement une implémentation ouverte des services CAO. Les documents IDL associés furent compilés avec succès sur plusieurs implémentations d'ORB tels TAO v.1.1.12, Orbix 2000 v.1.2, JacORB v.1.3.21, OmniORB v.3.0, Mico v.2.3.4.

Vue d'ensemble des interfaces

Les services CAO sont regroupés en sept modules interdépendants (diagramme des paquetages UML ci-dessous).

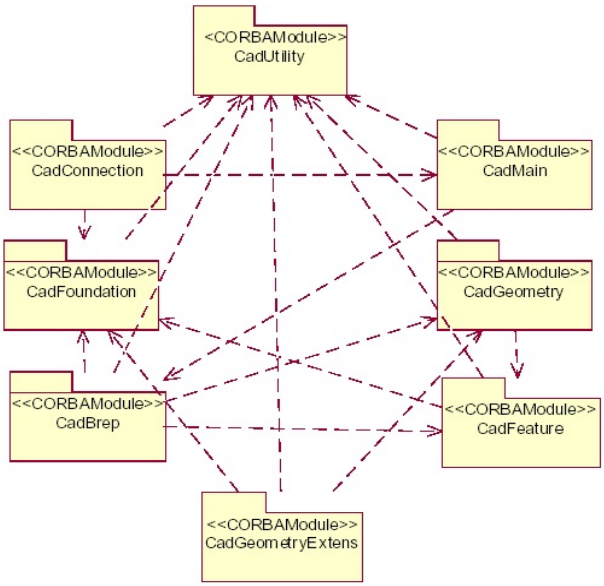


FIGURE 2.17 – *Diagramme des packages «OMG CAD Services»*

Les flèches pointillées représentent les relations de dépendance, les flèches sont orientées de l'entité dépendante vers l'objet de la dépendance.

Le module «CadConnection»

Ce module permet aux clients de se connecter à l'interface «Model» du module «CadMain». L'interface «CadServer» permet de se connecter à l'interface «CadSystem» pour obtenir des informations et activer le système natif sous jacent. Ce paquetage offre un mode de connexion uniforme entre les différents systèmes natifs.

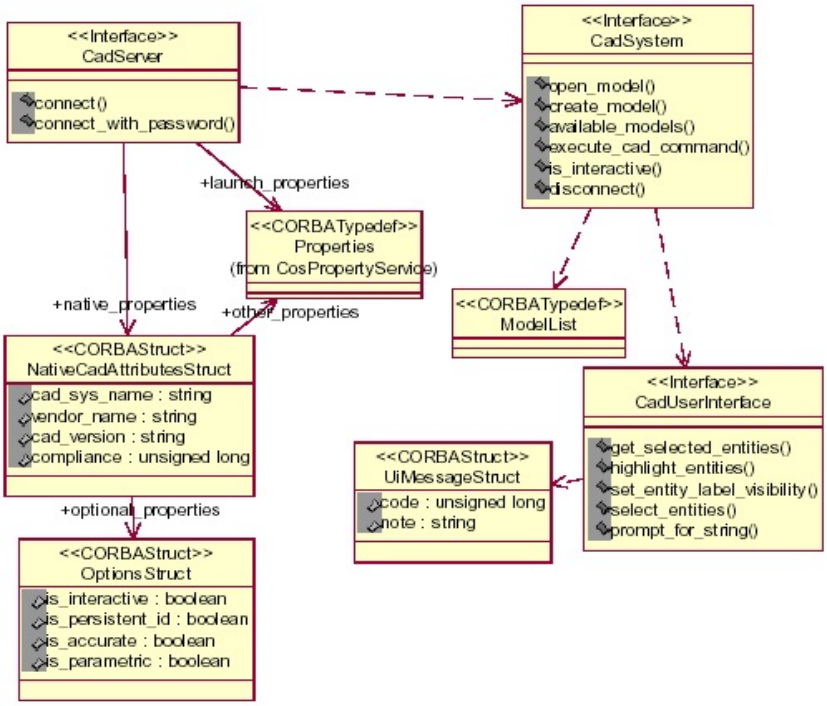


FIGURE 2.18 – *Module «CadConnection»*

L'interface «CadServer» permet de se connecter simultanément à différents systèmes implémentant les services CAO. L'interface «CadSystem» chapeaute les différentes implémentations des modeleurs géométriques. L'interface «CadUserInterface» permet l'interaction avec l'interface graphique du système natif, cette interface est optionnelle.

Le module «CadMain»

Ce module comprend les interfaces «Model» et «EntityFactory». La première regroupe les opérations sur les modèles CAO. Un modèle CAO représente la structure arborescente des pièces et assemblages, les entités géométriques, la représentation par les frontières, les caractèristiques ou «features», les textes et données associées. La deuxième fournit les services de création de multiples entités géométriques.

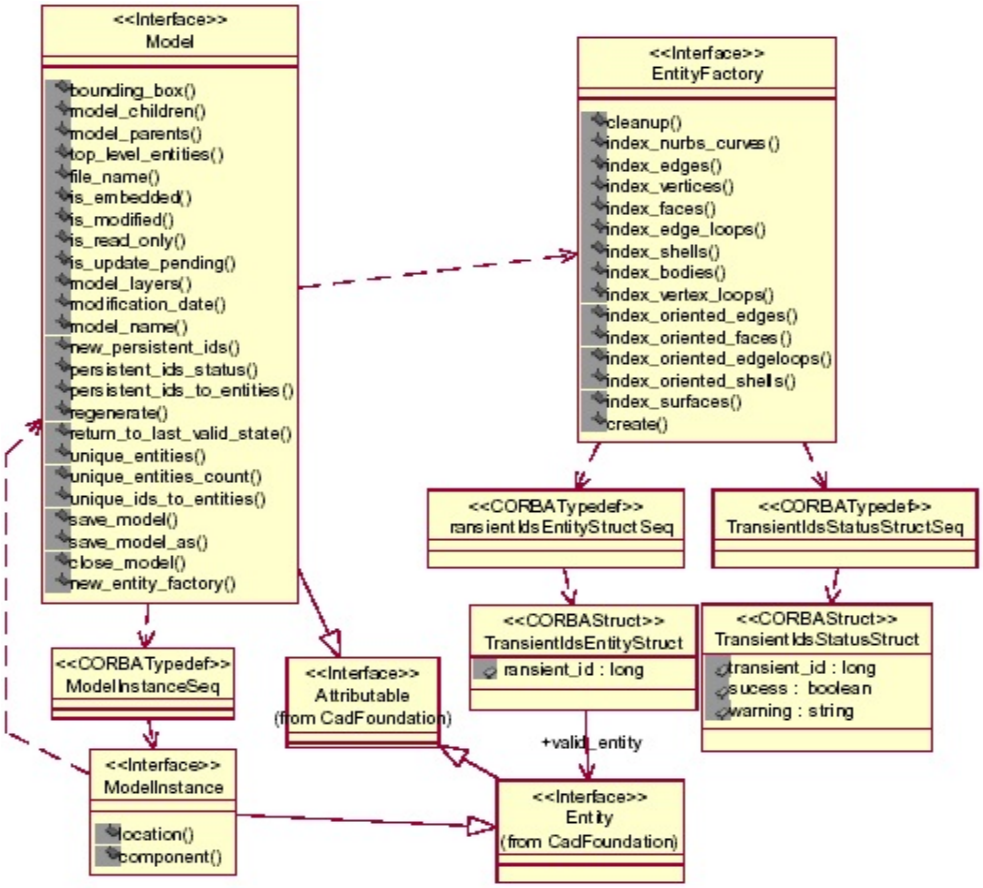


FIGURE 2.19 – *Module «CadMain»*

Ces entités sont basées sur le modèle NURBS mais, à cause des différences de représentation entre systèmes, le processus de création a lieu en deux temps. Premièrement les entités géométriques sont indexées via les opérations «index_xxx». Les index retournés peuvent être réutilisés pour indexer des entités géométriques interconnectées. Puis, les entités sont crées dans le modeleur natif, via l'opération «create».

Le module «CadFoundation»

Ce module définit des éléments et comportements partagés par toutes les entités géométriques ainsi que les interfaces permettant aux applications de les regrouper de manière spécifique, par exemple, sous forme de calques (entités partageant la même couleur).

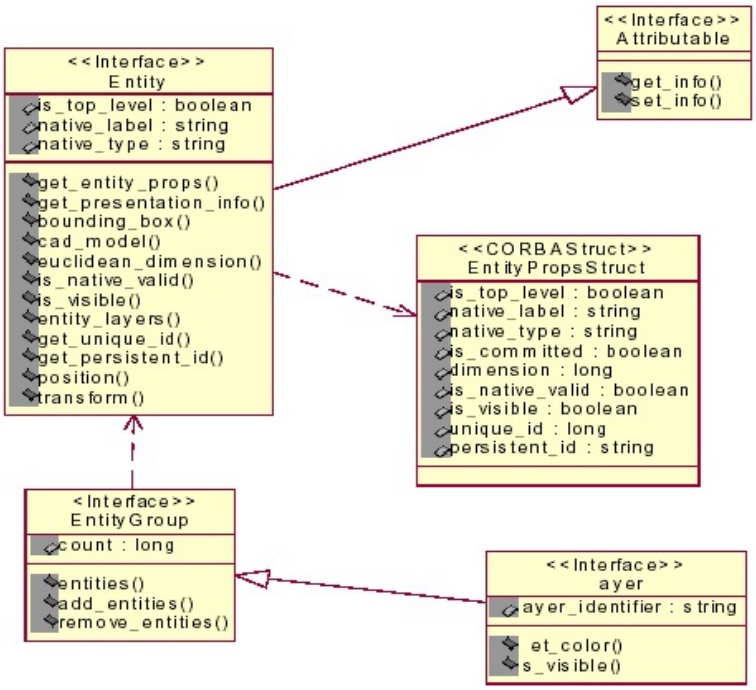


FIGURE 2.20 – *Module «CadFoundation»*

L'interface «Entity» définit les propriétés et comportements de base des objets CAO, lesquels sont hérités par les objets spécialisés. L'interface «Attributable» permet, aux entités l'implémentant, d'être décorées d'informations spécifiques aux applications. L'interface «EntityGroup» apporte des fonctionnalités permettant de grouper les entités géométriques suivant une sémantique spécifique aux applications. L'interface «Layer» permet de regrouper les entités partageant les mêmes propriétés de présentation.

Le module «CadGeometry»

Ce module contient les structures de données géométriques de base ainsi que les interfaces utilisées à travers les services CAO.

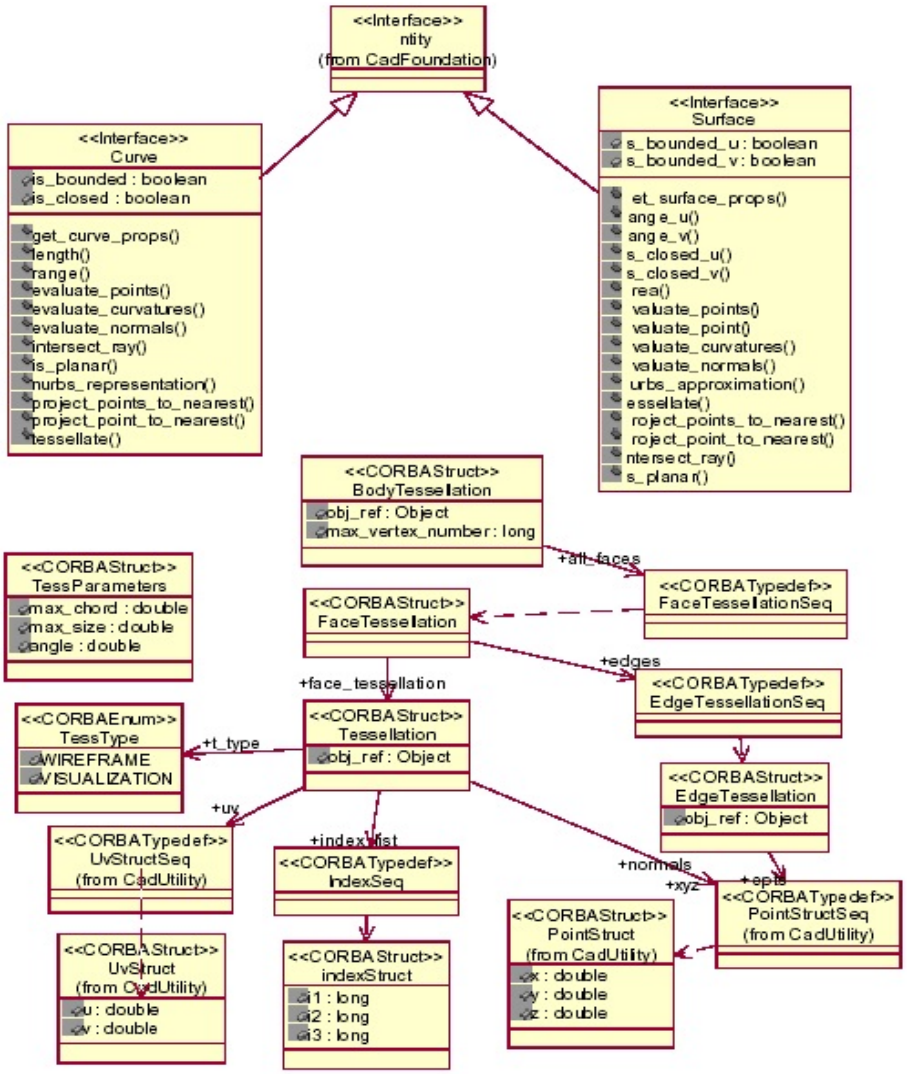


FIGURE 2.21 – *Module «CadGeometry»*

Les deux principales interfaces de ce module sont «Surface» et «Curve» qui dérivent toutes deux de l'interface «Entity» du module «CadFoundation».

Le module «CadGeometryExtens»

Ce module est une extension du module «CadGeometry», il contient les deux nouveaux sous modules «CadCurve» et «CadSurface». Les interfaces définies dans ces modules spécialisent un certain nombre d'entités.

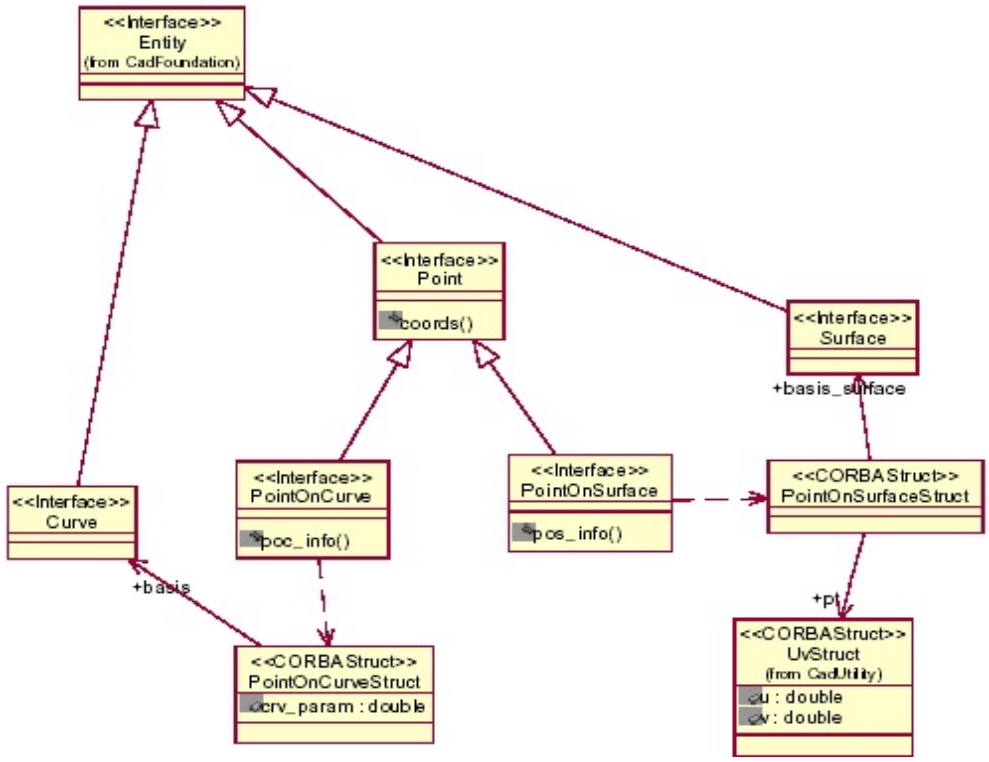


FIGURE 2.22 – *Module «CadGeometryExtens»*

Ce module spécialise l'interface «Point» et définit les sous interfaces «PointOnCurve» et «PointOnSurface».

Le module «CadGeometryExtens : :CadCurve»

Ce module spécialise l'interface «Curve» et définit les interfaces de courbes spécifiques.

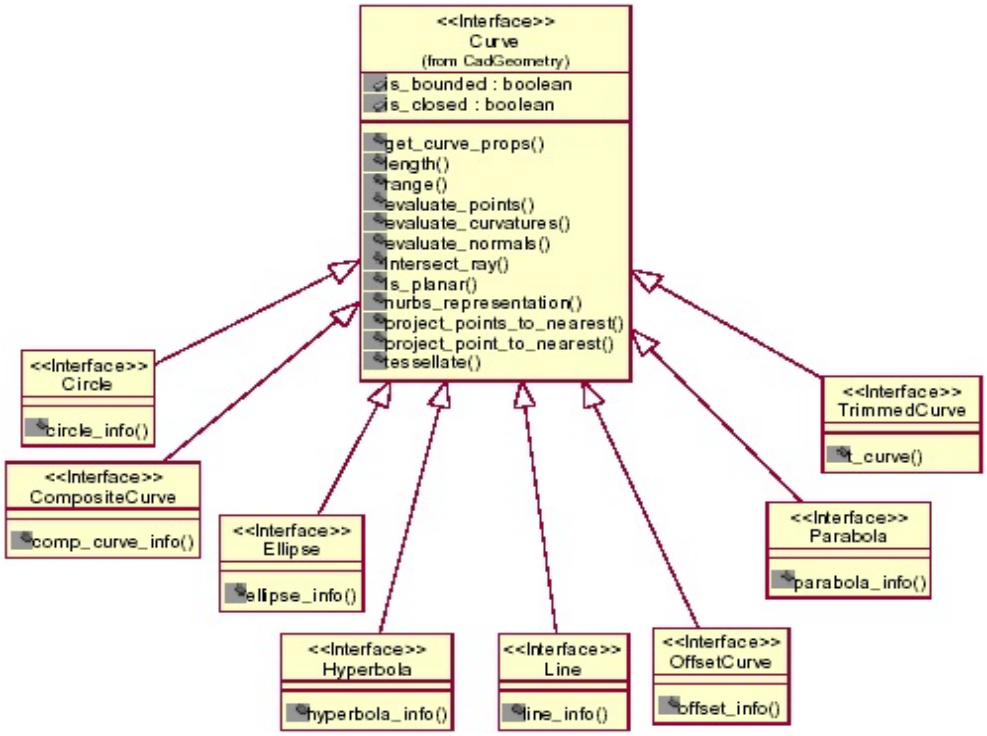


FIGURE 2.23 – *Module «CadGeometryExtens : :CadCurve»*

Le module «CadGeometryExtens : :CadSurface»

Ce module spécialise l'interface «Surface» et définit les interfaces de surfaces spécifiques.

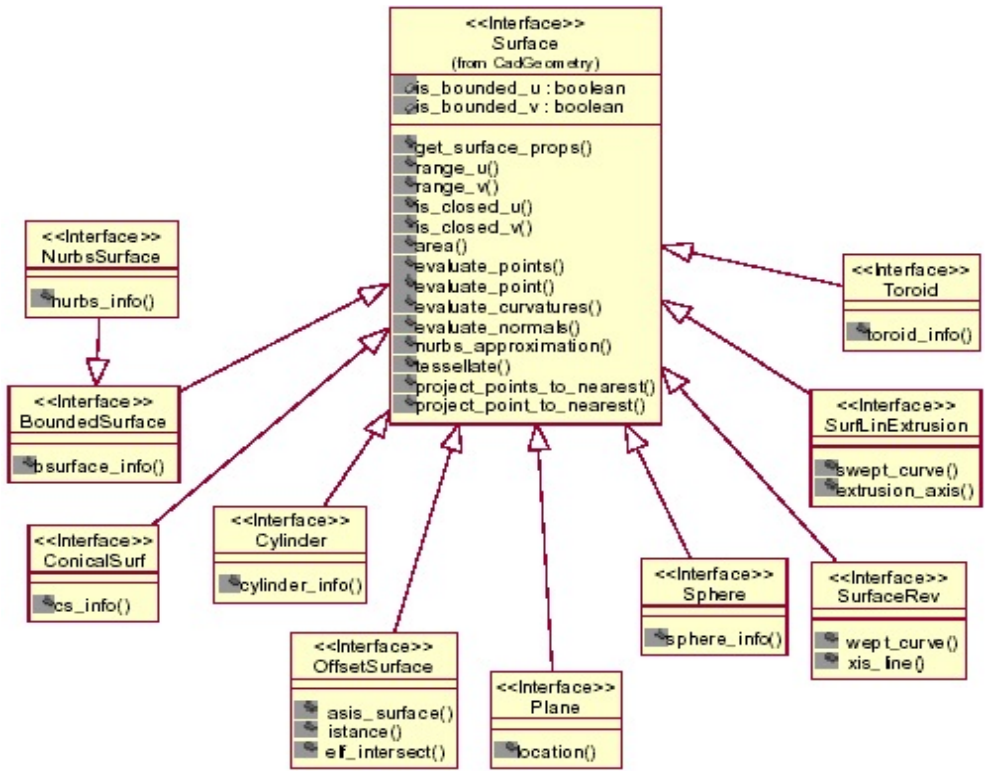


FIGURE 2.24 – *Module «CadGeometryExtens : :CadSurface»*

Le module «CadBRep»

Ce module contient le modèle solide représenté par les frontières. Celui-ci peut exposer ses caractéristiques paramétriques afin que la forme soit régénérée à travers les interfaces du module «CadFeature».

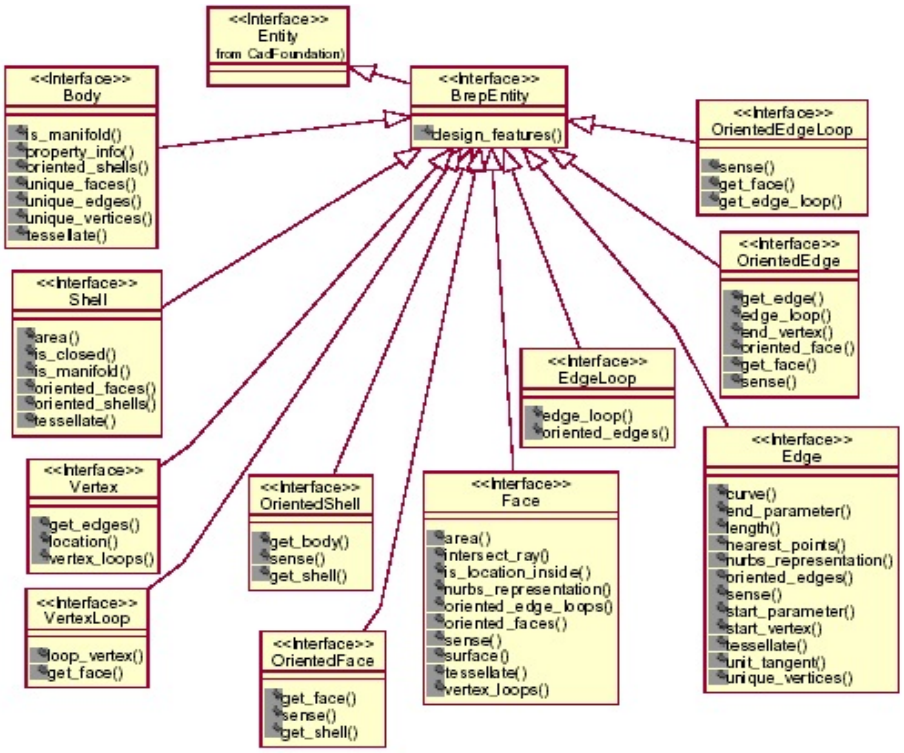


FIGURE 2.25 – *Module «CadBRep»*

L'interface «BRepEntity» dérive de l'interface «Entity» du module «CadFoundation». Elle contient les propriétés et comportements communs hérités par toutes les entités du module, en l'occurrence les caractèristiques (ou autres paramètres) utilisés durant la conception. L'interface «Body» définit les caractéristiques d'un corps formé d'un ensemble de volumes fermés. L'interface «Shell» définit les fonctionnalités des coques. Une coque ouverte est appelée une peau, ou «Skin», et une coque fermée doit toujours être utilisée par un corps. L'interface «OrientedShell» définit les fonctionnalités associées aux coques des corps. L'interface «Vertex» définit les fonctionnalités d'un point de l'espace, topologiquement indépendant. L'interface «VertexLoop» définit les fonctionnalités des contours délimités par les sommets des faces. L'interface «EdgeLoop» définit les fonctionnalités des contours délimitées par des arêtes. L'interface «OrientedEdgeLoop» définit, les fonctionnalités des contours d'arêtes orientées des faces. L'interface «Face» définit les fonctionnalités des régions triangulaires ou quadrilatérales délimitées par des arêtes. L'interface «OrientedFace» définit les fonctionnalités des faces d'une coque. L'interface «Edge» définit les fonctionnalités des arêtes. Une arête est une portion de courbe avec un point de départ et un point d'arrivée, si ces derniers sont confondus, on parle d'arête fermée. L'interface «OrientedEdge» définit les fonctionnalités des arêtes orientées.

Le module «CadFeature»

Ce module fournit les interfaces permettant de modifier les entités CAO natives. Elles permettent de supprimer des caractéristiques de conception variées et d'ensembles d'expressions paramétriques définissant la géométrie. Par exemple, la géométrie d'une boîte à laquelle sont associés les paramètres définissant ses dimensions. Si l'utilisateur altère ces paramètres, la géométrie doit être régénérée.

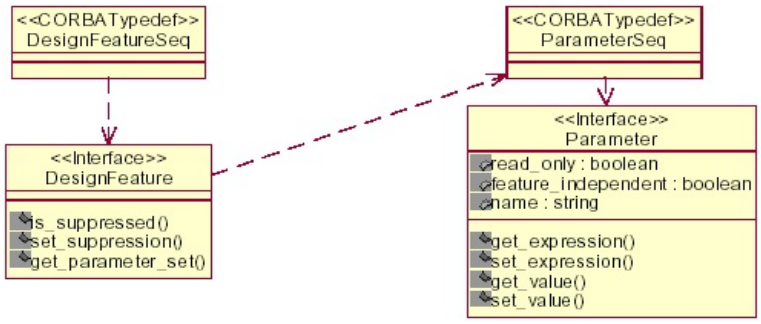


FIGURE 2.26 – *Module «CadFeature»*

La définition paramétrique de la géométrie est une structure arborescente décrivant l'historique de création des entités dans un ordre bien défini. L'interface «DesignFeature» définit les fonctionnalités des noeuds de cette structure. L'interface «Parameter» définie les fonctionnalités des paramètres.

Le module «CadUtility»

Ce module regroupe des définitions de types et de structures de données d'ordre général. Aucune interface n'est définit dans ce module.

2.4.4 Synthèse

Les services CAO représentent une couche d'abstraction fondamentale à l'interopérabilité des systèmes d'ingénierie. Les scénarios d'utilisation sont multiples. Pour le développeur, ces services l'isolent de l'API du système natif et lui garantissent la portabilité de ses applications. Dans un environnement collaboratif, la géométrie peut être fournie par un serveur CAO et utilisée par diverses applications d'ingénierie fonctionnant sur des systèmes hétérogènes. Celles-ci peuvent ajouter des attributs métiers afin d'effectuer des calculs et, éventuellement, intégrer leurs résultats, en tant qu'attributs dans le système de CAO. Une application peut, par exemple, simuler un véhicule dont les composants ont été conçus sur des systèmes hétérogènes. Les services CAO permettent l'interopérabilité des systèmes sans étape intermédiaire tel que le transfert de données via un fichier d'échange ou le transfert complet d'un ensemble de données. Enfin, les services CAO fournissent les fondements d'interopérabilité pour les services de plus haut niveau, tels que la gestion de données techniques et la simulation distribuée.

2.4.5 Les services PDM

En Août 1996, le Mfg DTF propose de normaliser les interfaces des services fournis par les systèmes de GDT (Gestion de Données Techniques), ou PDM (Product Data Management). On parle alors de « PDM Enablers». Ces interfaces, disponibles à travers l'ORB, doivent permettre de gérer les données du produit dans un environnement de systèmes hétérogènes et distribués.

Les services de GDT doivent offrir les mécanismes permettant de gérer les informations du produit durant son cycle de développement. Pour ce faire, sont pris en compte les acteurs et leurs rôles, les services, processus et flux de l'entreprise, les états d'un projet, les procédures de validation etc. Finalement, les mécanismes de l'entreprise sont modélisés, on parle alors d'entreprise virtuelle. Les principaux besoins identifiés sont : la demande d'intervention d'un ingénieur, la demande de modification, l'intégration de données relatives à la fabrication, au test, au diagnostique, à la maintenance, la gestion des documents et des versions, la définition de la structure du produit, la présentation sous différentes vues, l'amélioration de l'efficacité de gestion, la gestion des configurations. Le processus de développement du produit doit pouvoir être projeté en tant que scénario d'utilisation du SGDT. Le lecteur pourra obtenir plus de précisions dans [24].

2.5 Conclusion

Nous avons présenté, à travers les travaux des communautés STEP et OMG, les deux approches de l'interopérabilité des systèmes d'IAO. En s'adressant spécifiquement à l'interopérabilité des systèmes d'ingénierie, l'approche STEP est quelque peu monolithique, focalisée sur les données et ne prenant pas en compte les aspects comportementaux. L'approche de l'OMG est en revanche une approche en couches. Tout d'abord, l'architecture CORBA traite du problème d'interopérabilité des systèmes, en général. Sur cette base sont développées des solutions d'interopérabilité dans des domaine spécifiques tel que l'ingénierie. Ainsi, des interfaces CORBA spécifiques à chaque domaines sont définies, «CAD Services» pour la CAO, «PDM Enablers» pour la gestion de données techniques et «DSS» pour la simulation distribuée.

Aussi, il est important de souligner que ces deux approches ne se placent pas au même niveau d'abstraction. Alors que STEP normalise les représentations, l'OMG normalise les services. Même si l'approche STEP reste importante grâce aux nombreux domaines qu'elle a pu formaliser, elle souffre de son faible niveau d'abstraction. En effet, les différences de représentation sont à la base du problème d'interopérabilité et STEP propose l'harmonisation de celles-ci. Premièrement, cette démarche ne donne guère de liberté aux éditeurs de logiciels qui n'ont qu'à implémenter les modèles définis par STEP. De plus, cette approche focalise sur les données et non sur les responsabilités, les services. L'approche de l'OMG, est à notre avis, plus sûre car elle se situe à un niveau fonctionnel. En effet, les représentations sont d'avantage sujettes aux changements que ne le sont les fonctionnalités attendues des systèmes, même si celles-ci sont amenées à s'étendre.

Enfin, il faut prendre en compte l'aspect historique et culturel. En effet, les travaux de la communauté STEP ont débuté dans les années quatre vingts dans le domaine de la CFAO. Les concepts et méthodes utilisés, reflètent l'état des connaissances de cette époque. L'approche de l'OMG est très récente et basée sur les concepts orientés objets qui lui confèrent son haut niveau d'abstraction. Il est intéressant de suivre le développement des technologies dans leurs

contextes. STEP fut créé en réponse au problème crucial de l'échange des données dans les domaines de l'ingénierie. Domaines dans lesquels le volume de données est très conséquent et leur structuration très complexe. CORBA fut créé dans le souci de concevoir des applications à base de composants réutilisables et inter opérables fonctionnant sur des systèmes hétérogènes et distribués.

Comme nous l'avons déjà évoqué, la problématique de STEP est à notre avis, incluse dans la problématique CORBA. Mais CORBA apporte également d'autres perspectives importantes dans le monde de l'IAO. En effet, les applications d'IAO sont généralement volumineuses, complexes et donc difficiles à maintenir. CORBA offre le moyen de rationaliser la conception de logiciels en la fragmentant sous forme de composants. De plus, c'est le moyen de distribuer et de paralléliser des algorithmes très lourds tels que les simulations mécaniques.

Dans la première partie de l'état de l'art nous avons présenté la norme STEP et le cadre méthodologique utilisé. Cette norme ainsi que les méthodes qui la sous tendent ont été détaillées afin que le lecteur en appréhende précisément le niveau d'abstraction. Dans la deuxième partie, nous avons présenté l'architecture de composants logiciels COM et l'architecture générale CORBA pour l'interopérabilité des logiciels à base d'objets. L'objectif était de présenter concrètement la problématique de l'interopérabilité des composants logiciels et de montrer, à travers ces architectures, les concepts orientés objets.

L'exposé a été organisé de sorte que le lecteur appréhende, dans l'ordre chronologique, les approches menées. Celui-ci découvre, graduellement, le degré d'abstraction croissant des concepts et méthodes utilisées. Enfin, en montrant l'intérêt qu'ont les concepts orientés objets dans le développement d'applications évolutives et interopérables, cet état de l'art a pour but d'orienter les choix méthodologiques et architecturaux des personnes concernées dans le développement de systèmes d'IAO.

Chapitre 3

Le prototypage rapide

3.1 Introduction

La coopération des différents domaines de l'ingénierie passe par l'interopérabilité des systèmes associés. Le domaine du prototypage rapide n'échappe pas aux problèmes liés à l'hétérogénéité des systèmes utilisés. Qu'ils interviennent entre systèmes de CAO ou entre systèmes de CAO et systèmes de prototypage, les problèmes d'échange de données impliquent des reprises et corrections s'avérant fastidieuses, coûteuses et qui rendent le prototypage moins efficace. Nous présentons brièvement le domaine du prototypage rapide puis abordons plus en détail les procédés de fabrication par couches, les besoins informatifs des logiciels associés et les contraintes que doivent respecter les modèles CAO. Nous montrons que l'interopérabilité entre systèmes de CAO et systèmes de prototypage est uniquement basée sur l'échange de données et que les interfaces correspondantes ne sont pas adaptées à la prise en compte des contraintes métiers [31]. Nous proposons de reconsidérer l'interopérabilité CAO/prototypage d'un point de vue fonctionnel, c'est à dire comme des échanges de services et tentons d'identifier et de définir les composants et interfaces nécessaires au bon fonctionnement de la chaîne numérique. Nous verrons les avantages de cette approche dans l'intégration des données de prototypage dans le système d'information de l'entreprise.

3.2 Le prototypage rapide

Le prototypage rapide consiste à fabriquer au plus vite des prototypes de produits. Tous les outils et techniques permettant de réduire les temps de fabrication et les coûts de développement du produit entrent dans la catégorie des procédés de prototypage rapide : CAO, rétro conception, fabrication par couches, usinage rapide, duplication par moule souple, fonderie en matière perdue, etc. [3].

3.3 La fabrication par couches

Parmi les techniques de prototypage rapide les techniques de fabrication par couches, en anglais «layer manufacturing» sont capables de restituer physiquement des objets 3D définis en CAO plus rapidement qu'avec des procédés traditionnels et sans outillage. Qu'il s'agisse de

couches de résine polymérisée, de couches de poudre frittée, de couches découpées, collées, résultant de la projection de matière sur un liant et inversement ou de sections construites par dépôt de fil fondu, la fabrication se fait par apport itératif de couches de matière, par opposition aux techniques d'usinage qui procèdent par enlèvement de matière. On parle dans le premier cas de procédés additifs et dans le deuxième cas de procédés soustractifs [3]. Ces procédés permettent de fabriquer des pièces dont la complexité interdit l'usinage par des procédés traditionnels.

3.4 Principaux procédés de fabrication par couches

Notre objectif n'est pas de présenter de manière exhaustive les procédés existants, leurs avantages et inconvénients dans tel ou tel domaine d'application mais de montrer leurs caractéristiques communes et les principes sur lesquelles ils reposent.

La stéréolithographie (SLA)

Ce procédé est basé sur la photo polymérisation de résine. Un laser UV solidifie localement un polymère liquide photosensible. Guidé par un système de miroir galvanométrique, le laser parcourt et solidifie la section de l'objet à la surface du bain de résine. Après chaque couche, la plateforme portant l'objet descend dans le bain de résine.

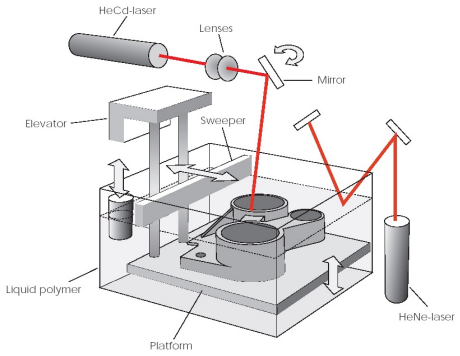


FIGURE 3.1 – *Principe de la stéréolithographie*

L'empilage des couches successives permet d'obtenir l'objet 3D. Des supports sont nécessaires pour rigidifier la structure. Enfin, la pièce subit une post polymérisation dans un four UV. Le choix de l'épaisseur des couches influence la précision, l'état de surface et le temps de fabrication.

Le masquage/flashage par lampe UV

Également basé sur la photo polymérisation de résine, le procédé «Solid Ground Curing» utilise un système de masque et une lampe de forte puissance comme source lumineuse.

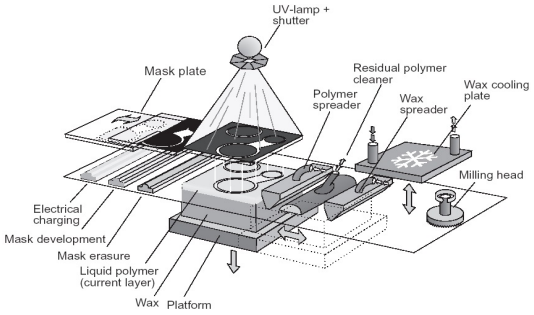


FIGURE 3.2 – *Principe du masquage flashage*

Tous les points d'une même section sont solidifiés simultanément. Le motif du masque est le négatif de la section en construction. Les points de la section sont irradiés par une puissante lampe UV. Le masque est une vitre sur laquelle un toner dépose une encre électrostatique sur le principe de la ionographie. Le polymère non solidifié est pompé et l'espace libéré est comblé par une fine couche de cire dont l'épaisseur est rectifiée par fraisage. Finalement, la cire est éliminée par fusion dans un four micro ondes ou dans un bain d'acide.

Le frittage sélectif laser (SLS)

Ce procédé est basé sur le frittage sélectif de poudre SLS (Selective Laser Sintering). Un puissant laser agglomère un matériau en poudre, éventuellement enrobé d'un liant, situé dans un réservoir. Le réservoir s'abaisse d'une distance égale à l'épaisseur d'une couche puis un rouleau dépose une nouvelle couche de poudre sur la section précédemment construite.

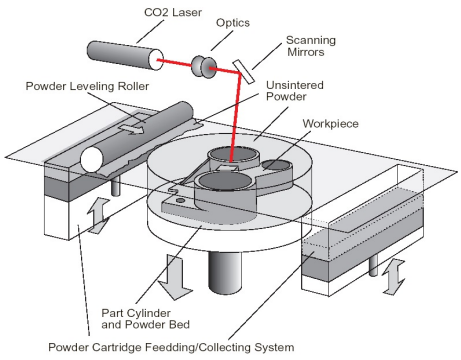


FIGURE 3.3 – *Principe du frittage sélectif de poudre*

Un traitement thermique complémentaire peut être nécessaire pour améliorer les propriétés physiques de la pièce, en réduisant notamment la porosité.

L'imprimante 3D

Ce procédé est basé sur la pulvérisation du matériau. Cette technique est un mélange de projection de liant et de frittage.

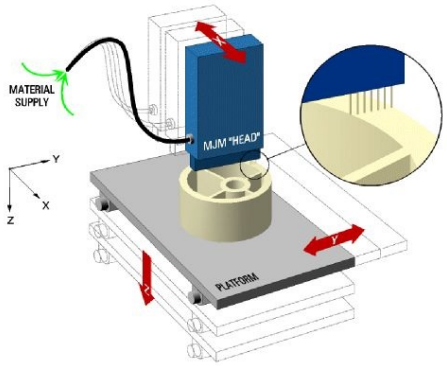


FIGURE 3.4 – *Le principe de l'imprimante 3D*

Une tête d'imprimante comprenant une ou plusieurs buses projette des gouttelettes de liant au dessus d'un bac de matériau en poudre. Le liant pénètre et agglomère la poudre qui devient rigide. Un post traitement thermique provoque l'évaporation du liant et le frittage véritable de la poudre.

Le procédé «Ballistic Particle Manufacturing» (BPM)

Le procédé BPM est basé sur la pulvérisation et l'agglomération de fines gouttes de cire liquide. La tête est articulée de manière continue dans les trois directions de l'espace afin d'appliquer le matériau suivant une direction qui soit le plus proche possible de la normale à la surface à construire. Ce procédé nécessite la construction de supports et n'a jamais vraiment été industrialisée.

Le procédé «Model Maker»

Également basé sur la pulvérisation du matériau, ce procédé utilise deux buses, l'une déposant un matériau thermoplastique constituant l'objet, l'autre une couche de cire servant de support. Les gouttelettes tombent et se solidifient au contact de la surface. Un système de fraisage est utilisé pour égaliser l'épaisseur des couches.

Le procédé «Multijet Manufacturing»

Également basé sur la pulvérisation du matériau, ce procédé fonctionne sur le principe du traceur. Une tête comprenant quatre vingt seize jets disposés en ligne dépose un matériau thermoplastique développé spécialement à cet effet. Une fois terminée la plate-forme descend pour laisser place au cycle suivant. L'avantage de ce procédé est sa capacité à s'intégrer dans les bureaux d'étude. Son logiciel permet à l'utilisateur d'envoyer, dans une file d'attente, plusieurs fichiers STL. Le positionnement et la génération des supports sont automatiques et la fabrication démarre dès que la machine se libère.

Le procédé «Fused Deposition Modeling» (FDM)

Le procédé FDM est basé sur l'extrusion du matériau. Une buse, guidée par trois axes numériques, fond et dépose un fil thermo fusible. Le filament se solidifie immédiatement.

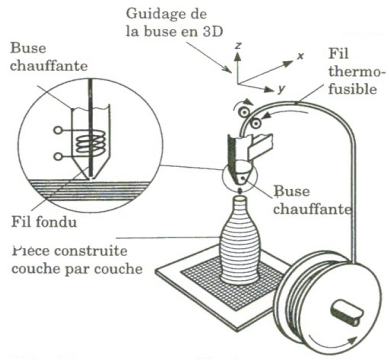


FIGURE 3.5 – *Le principe de l'extrusion*

Les parties de l'objet sont construites par empilement en spirale du fils. La buse dépose son fil dans un plan horizontal, puis la plate-forme s'abaisse pour laisser place au cycle suivant.

Le procédé «Laminated Object Manufacturing» (LOM)

Le procédé LOM est basé sur le laminage du matériau. Disponible sous forme de feuilles ou de plaques, le matériau est soit empilé, collé puis découpé soit découpé, empilé puis collé suivant la technique utilisée.

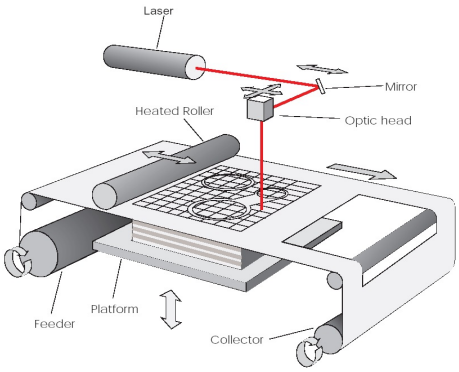


FIGURE 3.6 – *Le principe du découpage laminage*

Le procédé est basé sur l'empilement, le collage et le découpage laser de feuilles de papier enrobées d'un film plastique qui, fondues au passage d'un rouleau chauffant, se collent aux couches précédentes.

Le procédé de stratoconception

Ce procédé repose sur le principe d'empilement de plaques de matériau découpées. L'interface «stratoconcept» permet la flexibilité de l'outil de découpe, micro fraisage, laser, jet d'eau, MOCN (Machine Outil à Commande Numérique).

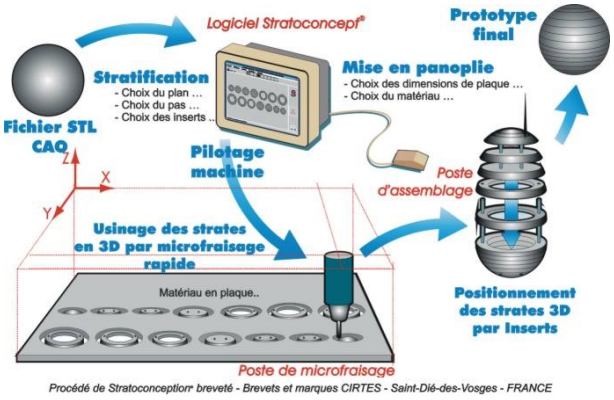


FIGURE 3.7 – *Le principe de la stratoconception*

Pour économiser la matière, le logiciel optimise le placement des sections dans la plaque. Les plaques de différents matériaux (bois, plastique, aluminium, acier) peuvent être assemblées par collage, soudage, etc. L'assemblage peut être guidé et renforcé par des inserts.

Le procédé «Selective Adhesive and Hot Press Process»

Ce procédé ressemble à une imprimante et utilise du papier comme matériau. Une résine en poudre est appliquée sur le papier, par un toner, puis un laser l'agglomère pour former la section. Une autre feuille est déposée et un rouleau vient presser le tout à haute température. Puis le contour est découpé et le papier superflu non aggloméré est retiré. Enfin, la pièce subit une finition manuelle.

Le procédé «JP Sytem 5»

Ce procédé est similaire à la stratoconception, il procède par empilement de papier adhésif. Les sections sont découpées à l'aide du couteau monté sur un système de traceur.

Caractéristiques communes

Tous ces procédés reposent sur le même principe, la construction successive des sections de l'objet. Ils comportent une partie opérative composée d'axes numériques pilotant différents organes : les miroirs, les outils de découpe, les toners, les micros fraises, les réservoirs de matière, les systèmes de raclage, etc. Un micro-ordinateur et un logiciel spécifique à chaque procédé permettent de calculer, à partir des données CAO, les sections de l'objet et les trajectoires de pilotage des différents organes.

Tous ces procédés requièrent, par conséquent, une description numérique des sections de l'objet. Celle-ci est calculée à partir d'une géométrie tridimensionnelle issue d'un système de CAO, c'est le processus de tranchage, en anglais «slicing process». Les sections sont perpendiculaires à la direction de construction et la distance qui les sépare correspond à l'épaisseur des couches.

La description bidimensionnelle des sections distingue les contours extérieurs et intérieurs et permet d'identifier les sites où le matériau doit être polymérisé, aggloméré, découpé ou collé suivant le procédé utilisé. Cette description est utilisée telle quelle par les procédés reposant sur le masquage/flashage et sur le découpage. Pour les procédés basés sur la polymérisation ou

le frittage elle sert au calcul des trajectoires de remplissage des sections par le laser, c'est le processus de remplissage, en anglais «filling process».

Pour l'ensemble des procédés, la durée et le coût de fabrication sont proportionnels à la finesse des couches. L'état de surface dépend également de cette épaisseur mais aussi de l'orientation de la pièce. Nous aurons l'occasion de présenter, au chapitre IV, un module de simulation de l'état de surface et un module de calcul du coût de fabrication.

3.5 Les systèmes de prototypage

Le rôle des systèmes de prototypage est de piloter les procédés de sorte qu'ils réalisent la géométrie souhaitée. Ces systèmes doivent disposer de la géométrie de l'objet puis effectuer différents traitements pour obtenir les trajectoires nécessaires au pilotage des différents organes. L'ensemble de ces traitements forme une chaîne numérique entre, systèmes de CAO et procédés de prototypage. Un certain nombre de modules logiciels composent les maillons de cette chaîne. Parmi ceux-ci le module de tranchage tient une place centrale. Nous présentons l'ensemble des modules de la chaîne en commençant par le module de tranchage.

Nous montrerons qu'il existe deux approches majeures pour implémenter un tel module. L'approche basée sur le tranchage direct qui consiste à calculer les sections à partir de la géométrie exacte ou bien l'approche polyédrique qui consiste à le faire à partir d'une géométrie polygonalisée. Cette dernière nécessite une étape supplémentaire de polygonalisation assurée par le module de maillage.

Le module de maillage étant soit intégré au système de CAO soit au système de prototypage, l'échange de la géométrie peut s'effectuer sous deux formes, exacte ou polyédrique et dans les deux cas, via un fichier d'échange. L'échange de la géométrie exacte se heurte au problème de compatibilité des représentations des systèmes hétérogènes au même titre que les échanges entre systèmes de CAO. De plus, celui-ci implique la duplication des données et des modèles correspondants. L'échange de la géométrie polyédrique, en revanche, se révèle relativement fiable, mais nous montrons que le format utilisé, de par la pauvreté des informations qu'il contient, n'est pas adapté au respect des contraintes géométriques imposées par le module de tranchage. Nous montrons, par ailleurs, qu'il est possible de définir un format plus adapté mais également plus concis.

Enfin, nous proposons de reconsidérer l'interopérabilité entre systèmes de CAO et systèmes de prototypage non pas comme des échanges de données mais comme une collaboration entre modules pouvant éventuellement être distribués. Pour cela, nous tentons de spécifier les fonctionnalités des différents modules sous forme de composants interopérables.

3.5.1 Module de tranchage

Pour piloter le procédé, le système de prototypage doit disposer des sections de l'objet. Celles-ci sont calculées, à partir de la géométrie de l'objet, par le module de tranchage, en anglais «slicer».

FIGURE 3.8 – *Géométrie avant et après tranchage*

Nous présentons brièvement, les deux approches de l'algorithme de tranchage puis les contraintes qu'il impose à la géométrie.

3.5.2 Tranchage direct et tranchage polyédrique

Même si certains travaux portent sur des méthodes de tranchage de géométries exactes (à base de surfaces de types bézier, spline, NURBS, etc.) ou tranchage direct [32], [33], [34], [35], [36], [37], les méthodes utilisées par la plupart des systèmes s'adressent à des géométries polyédriques à faces triangulaires.

En effet, en faisant intervenir des calculs d'intersection entre plans et surfaces paramétriques, les méthodes de tranchage direct se révèlent complexes et parfois instables. Les méthodes de tranchage de géométries polyédriques, ne faisant intervenir que des intersections entre plans et triangles, se révèlent en revanche, relativement simples et fiables mais imposent une phase de polygonisation qui occasionne une erreur de corde. La polygonalisation de la géométrie exacte est réalisée par le module de maillage.

3.5.3 Contraintes géométriques

Le calcul des sections de l'objet impose un certain nombre de contraintes sur la géométrie. Le respect de ces contraintes nécessite la collaboration de modules supplémentaires, tels que les modules de maillage, de positionnement et de correction. Ces modules étant souvent distribués entre plusieurs systèmes, des formats d'échanges sont utilisés à plusieurs niveaux.

En effet, le calcul des sections impose de connaître préalablement la direction de construction, ce qui revient à positionner l'objet dans l'espace de la machine, c'est le rôle du module de positionnement. De plus, il est impératif de disposer d'une géométrie fermée afin que les sections soient, elles aussi, bien fermées. Dans le cas contraire, des retours en conception ou des corrections sont nécessaires. Les modules de visualisation et de correction assistent l'utilisateur à cette tâche. D'après [38], 10% des fichiers STL créés à partir de modèles solides, possèdent des anomalies et 90% pour ceux créés à partir de modèles surfaciques.

Mais avant tout, le système de prototypage doit disposer de la géométrie de l'objet. L'échange de la géométrie entre systèmes de CAO et systèmes de prototypage rapide est conditionné par la localisation du module de tranchage.

3.5.4 L'échange de la géométrie

Les systèmes de prototypage permettent d'importer la géométrie issue de systèmes de CAO, soit sous forme exacte, soit polygonalisée. La plupart des systèmes actuels reposent sur le tranchage polyédrique, ce qui implique une phase intermédiaire de maillage et donc deux scénarios d'échange. La géométrie est soit polygonalisée sur le système de CAO, puis transmise, au système de prototypage, sous forme polyédrique, au format STL, soit transmise, du système de CAO, sous forme exacte et polygonalisée sur le système de prototypage. Afin d'enrichir les modèles de prototypage et intégrer les informations qu'ils contiennent dans un SGDT, certains travaux proposent, également, l'utilisation du standard d'échange STEP [39], [40], [41], [42], [43].

L'échange de la géométrie exacte

L'échange de la géométrie exacte fait intervenir des fichiers moins volumineux. Cependant, il s'effectue de la même manière qu'entre deux systèmes de CAO, par l'intermédiaire de formats natifs ou neutres et pose le problème de la compatibilité des représentations. Par exemple, le CTTM (Centre de Transfert de Technologies du Mans) reçoit 50% de fichiers STL, 30% de fichiers IGES, 10% de fichiers natifs et 10% de fichiers UNISURF. D'après [3], l'échange de données est un des problèmes les plus importants du prototypage rapide.

De plus, ce type d'interfaçage impose que le système de prototypage comporte un modeleur de géométrie exacte, ce qui implique une redondance de fonctionnalités avec le système de CAO. D'ailleurs, de nombreux systèmes de prototypage, en intégrant des fonctions de conception, en vue d'effectuer des retouches et corrections, tendent à se rapprocher des systèmes de CAO.

L'échange de la géométrie polyédrique

Initialement requise par les premiers procédés de fabrication par couches, la géométrie polyédrique à faces triangulaires est aujourd'hui encore largement utilisée. Générée, à partir du système de CAO, par un algorithme de maillage, celle-ci est transmise, sous forme de fichier, au format STL.

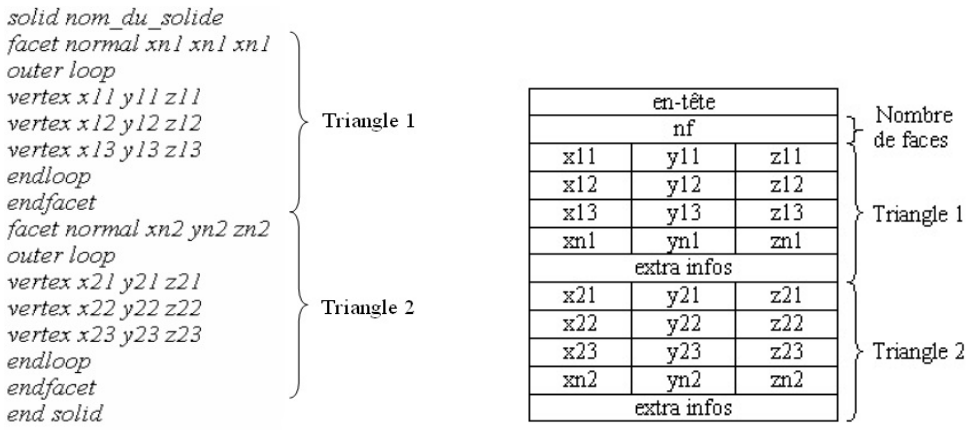
Un inconvénient majeur de la phase de maillage est la perte de précision et le volume des données générées. En effet, les surfaces exactes sont approximées par des facettes planes. L'erreur induite s'appelle l'erreur de corde. Le maillage est généré pour respecter une erreur de corde choisie par l'utilisateur. Malheureusement, la taille des fichiers générés est inversement proportionnelle à l'erreur de corde. Un compromis doit donc être fait entre la précision et la taille des fichiers.

L'échange de la géométrie tranchée

Certains systèmes proposent des formats d'échange de la géométrie tranchée. Suivant qu'il s'agit de tranchage direct ou polyédrique, les contours des sections sont représentés soit sous forme exacte, par des courbes paramétrées, soit par des polyèdres planaires, parmi ceux-ci les formats SLI et SLC de la société 3D Systems et le format CLI (Common Layer Interface) [44] de la société EOS. Certains systèmes utilisent également le format d'imprimante HPGL. Ces formats supposent que des échanges, entre systèmes, interviennent entre le calcul des sections et la fabrication. Dans ce cas, le module de tranchage s'exécute sur un système différent de celui qui pilote le procédé.

Le format STL

Introduit par la société 3D System [45], ce format encode une liste de triangles accompagnés de leurs vecteurs normaux. Les fichiers STL existent sous deux formes ASCII et binaire. Simple et lisible, du moins dans sa forme ASCII, ce format est fiable. Cependant, sa simplicité est, à notre avis, poussée à l'extrême car il ne contient pas, du moins de façon explicite, d'informations de nature topologique. Et pourtant celles-ci sont indispensables à la vérification de la fermeture de l'objet. Un certain nombre de travaux proposent des formats alternatifs [46], [47], [48], [49], [50].



(a) Ascii (b) Binaire

FIGURE 3.9 – *Structure du format STL*

Le format ASCII a l'avantage d'être lisible mais les nombreuses balises qu'il contient le rendent volumineux. De plus, celui-ci doit être interprété ce qui augmente le temps de chargement dans le système de prototypage rapide.

Le format binaire est constitué d'un en-tête de quatre vingts octets suivi du nombre de triangles codé sur quatre octets et de la liste des triangles. Un triangle est codé par ses trois sommets et son vecteur normal, soit douze nombres à virgule flottante simple précision, codés sur quatre octets chacun. Les concepteurs de ce format ont également prévu un espace de deux octets par triangle permettant de stocker des informations supplémentaires, celui-ci est généralement inutilisé. Ce format utilise la norme d'encodage binaire des nombres à virgules flottantes «IEEE 754 floating-point single precision» et ne présente pas de balises, ce qui le rend plus compact que son homologue ASCII et facilite son chargement.

Reconstruction topologique

Le format STL souffre d'une topologie maladroitement exprimée qui pénalise la taille des fichiers et leur traitement. Afin de vérifier la fermeture de l'objet, il est nécessaire de la reconstruire pour la rendre explicite [47], [51].

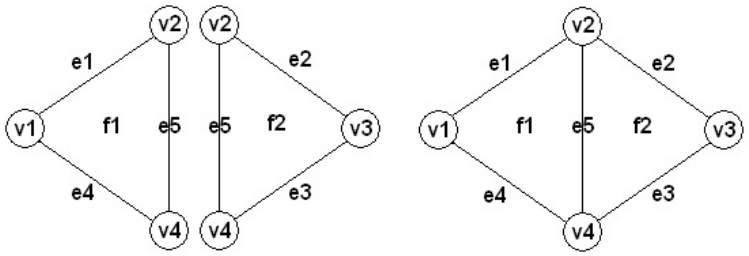


FIGURE 3.10 – *Avant et après reconstruction*

Si l'on considère deux faces triangulaires et adjacentes suivant une arête alors les extrémités de cette arête sont dupliquées dans le fichier STL. La reconstruction consiste à exprimer les triangles sous forme de sommets, arêtes et faces puis de fusionner les sommets et arêtes communes. La reconstruction est relativement coûteuse pour les maillages volumineux.

En pratique, il a été observé des écarts de l'ordre de 10e-16 mm entre sommets redondants. Ces sommets sont généralement situés sur les courbes communes aux surfaces adjacentes et l'écart observé provient des dispersions commises lors du maillage de chaque surface. Chaque surface étant maillée indépendamment à l'aide de son équation, les sommets situés sur le bord commun des surfaces sont donc calculés deux fois avec des erreurs différentes. L'algorithme de reconstruction doit intégrer ces dispersions dans un critère de vraisemblance faisant intervenir un petit intervalle de distance ε.

Format STL topologique

Nous montrons dans ce paragraphe qu'il est possible de créer un format moins volumineux et explicitant les informations topologiques. Sur le modèle du format STL nous définissons ce format sous forme ASCII et binaire.

```
<solid name="nom du solide">
<vertices>
x1 y1 z1
x2 y2 z2
</vertices>
<edges>
0 1
1 2
2 0
</edges>
<faces>
0
1
2
</faces>
</solid>
```

(a) Ascii

en-tête		
nv		
ne		
nf		
x1	y1	z1
x2	y2	z2
x3	y3	z3
v1	v2	
v2	v3	
v3	v1	
e1	e2	e3
xn	yn	zn

Nombre de sommets, arêtes et faces
v1, v2, v3
e1, e2, e3
f1

(b) Binaire

FIGURE 3.11 – *Le format STL topologique*

La forme ASCII de ce format commence par la balise <solid> dans laquelle le nom de l'objet peut être spécifié puis la balise <vertices> renferme les coordonnées des sommets, la

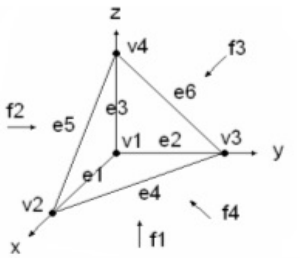
balise <edges> les arêtes et la balise <faces> les faces. Une arête est décrite par un couple de références à ses extrémités et une face par trois références à ses arêtes. Les sommets et arêtes sont référencés par leur ordre d'apparition dans leur balise respective.

La forme binaire de ce format conserve l'en-tête de quatre vingts octets suivi des nombres de sommets, arêtes et faces codés sur quatre octets chacun. Ensuite, figurent les listes de, sommets, arêtes et faces. Un sommet est codé par trois nombres, à virgule flottante simple précision, codés sur quatre octets chacun. Les arêtes et les faces sont codées, respectivement, par deux et trois entiers codés sur quatre octets chacun.

Présentons l'exemple d'un fichier STL ASCII renfermant un tétraèdre de côté unitaire. Les sommets sont notés v_i, les arêtes e_i et les faces f_i.

```
solid nom_du_solide
facet normal 0.0 0.0 -1.0
outer loop
vertex 0.0 0.0 0.0
vertex 1.0 0.0 0.0
vertex 0.0 1.0 0.0
endloop
endfacet
facet normal 0.0 -1.0 0.0
outer loop
vertex 0.0 0.0 0.0
vertex 1.0 0.0 0.0
vertex 0.0 0.0 1.0
endloop
endfacet
facet normal -1.0 0.0 0.0
outer loop
vertex 0.0 0.0 0.0
vertex 0.0 1.0 0.0
vertex 0.0 0.0 1.0
endloop
endfacet
facet normal 1.0 1.0 1.0
outer loop
vertex 1.0 0.0 0.0
vertex 0.0 1.0 0.0
vertex 0.0 0.0 1.0
endloop
endfacet
end solid
```

(a) Standard




```
<solid name = "nom du solide">
<vertices>
0.0 0.0 0.0
1.0 0.0 0.0
0.0 1.0 0.0
</vertices>
<edges>
0 1
0 2
0 3
1 2
1 3
2 3
</edges>
<faces>
0 1 3 0.0 0.0 -1.0
0 2 4 0.0 -1.0 0.0
1 2 5 -1.0 0.0 0.0
3 4 5 1.0 1.0 1.0
</faces>
</solid>
```

(b) Topologique

FIGURE 3.12 – *Tétraèdre unitaire aux formats STL et STL topologique ASCII*

La figure 3.12 montre que le format topologique est plus concis que le format classique mais dans quelle proportion ?

STL topologique vs STL

Il serait intéressant de comparer l'encombrement des deux formats afin d'apprécier le gain associé à l'utilisation du format STL topologique. Seulement, celui-ci dépend de la nature même de la géométrie qu'il contient. En effet, la taille d'un fichier standard dépend, uniquement, du nombre de faces alors que celle d'un fichier topologique dépend du nombre de sommets, arêtes

et faces. Le gain ne pouvant être établi pour toute géométrie, nous présentons les résultats obtenus pour une géométrie particulière, à savoir, la sphère maillée par une méthode polaire.

Il est également difficile d'évaluer la taille des fichiers ASCII, de façon exacte, car l'encombrement de la représentation ASCII d'un nombre à virgule flottante dépend de sa valeur (ex : «0.0» et «3,14159265358979323846264338327 95»). Cependant, l'expérience montre que celui-ci n'excède pas treize octets, nous utiliserons donc cette valeur. A noter que la présence de caractères de contrôle biaise également l'évaluation.

	standard	topologique
ascii	$t = 199f + 94$	$t = 36v + 8e + 48f + 156$
binaire	$t = 50f + 84$	$t = 12v + 8e + 24f + 92$

TABLE 3.1 – *Encombrement des deux formats dans le cas général*

Considérons la peau d'une sphère approchée par des faces triangulaires suivant une méthode polaire. Nous pouvons ordonner ces maillages suivant leur degré de polygonalisation. Ainsi, le plus petit maillage porte le degré de polygonalisation zéro, il possède cinq sommets, neuf arêtes et six faces. Il est possible d'établir les équations régissant le nombre de sommets, d'arêtes et de faces de tels maillages en fonction de leur degré de polygonalisation.

$$\begin{cases} v(i) &= i + 5 \\ e(i) &= 3i + 9 \\ f(i) &= 2i + 6 \end{cases} \quad \text{avec } i : \text{degré de polygonalisation } (i \geq 0)$$

Exprimons les nombres de sommets et d'arêtes en fonction du nombre de faces.

$$\begin{cases} v(f) &= \frac{f}{2} + 2 \\ e(f) &= \frac{3}{2} f \end{cases}$$

Nous obtenons l'encombrement des deux formats dans le cas de la sphère.

	standard	topologique
ascii	$t = 199f + 94$	$t = 78f + 228$
binaire	$t = 50f + 84$	$t = 42f + 116$

TABLE 3.2 – *Encombrement des deux formats dans le cas de la sphère*

Exprimons les gains fais entre les formats STL standard et topologique ASCII et binaire.

standard	$g = \frac{tstlascii - tstlbin}{tstlascii} = \frac{149f+10}{199f+94} \simeq 75\%$
topologique	$g = \frac{ttopascii - ttopbin}{ttopascii} = \frac{36f+112}{78f+228} \simeq 46\%$

TABLE 3.3 – *Gain entre formats STL ASCII et binaire*

ascii	$g = \frac{Tstlascii - Ttopascii}{Tstlascii} = \frac{121f+134}{199f+94} \simeq 61\%$
binaire	$g = \frac{Tstlbin - Ttopbin}{Tstlbin} = \frac{8f-32}{50f+84} \simeq 16\%$

TABLE 3.4 – *Gain entre formats STL standard et topologique*

D'une manière générale, les fichiers binaires sont moins volumineux et traités plus rapidement que les fichiers ASCII.

Qu'il soit ASCII ou binaire, le format topologique est plus avantageux que son homologue standard.

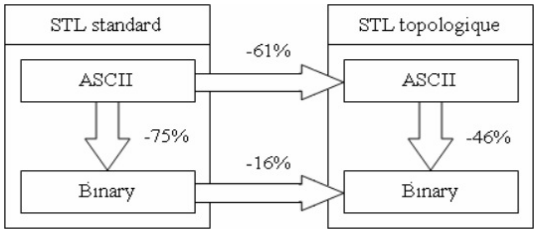


FIGURE 3.13 – *Gains entre formats STL et STL topologique ASCII et binaire*

Nous avons montré que le format STL pouvait être amélioré. Le format topologique que nous avons présenté explicite les informations topologiques tout en réduisant la taille des fichiers, ce format est, en quelque sorte, plus riche et plus concis. Enfin, l'existence du format STL témoigne de la non adéquation des solutions d'échange.

3.5.5 Module de vérification et de correction

Il est primordial de disposer de la topologie de l'objet pour s'assurer de sa validité. Les trous représentent, en effet, des anomalies topologiques pouvant être isolées. Par exemple, chaque arête doit posséder, exactement, deux faces incidentes et chaque sommet doit posséder autant d'arêtes que de faces incidentes. Dans le cas contraire, le maillage présente des anomalies, qui, si elles ne sont pas corrigées, poseront des problèmes lors du calcul des sections. Un module de vérification et de correction doit permettre de visualiser et de corriger de telles anomalies [52].

3.5.6 Module de maillage

Nous avons vu l'importance de la topologie et l'inadéquation du format STL, lequel nécessite une reconstruction coûteuse. Ceci nous amène à une réflexion sur les modules de maillage en prototypage rapide. Le format STL laisse à supposer que ces modules ne supportent pas la topologie. Nous ne savons pas, en pratique, si tel est le cas. Dans l'hypothèse du contraire, il serait alors absurde que celle-ci soit perdue lors d'un échange au format STL, puis reconstruite par le système cible. Il est donc nécessaire que le module de maillage et le format d'échange supportent la topologie.

3.5.7 Module de remplissage

Le module de remplissage, en anglais «filler», calcule les trajectoires de balayage de chaque section.

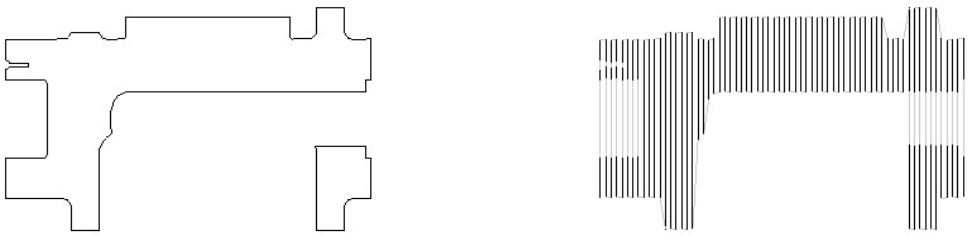


FIGURE 3.14 – *Section avant et après remplissage*

Il est nécessaire aux procédés de stéréolithographie ou de frittage lesquels polymérisent ou agglomèrent les sections points par points.

3.5.8 Autres Modules

Nous avons présenté les modules communs à la plupart des systèmes de prototypage mais des modules plus ou moins spécifiques existent également. C'est le cas du module de pilotage du procédé, du module de génération des supports, nécessaires aux procédés basés sur la polymérisation de résine. Certains systèmes permettent également, le découpage de géométries trop volumineuses, ou encore, la génération d'empreintes de moules. Des systèmes de numérisation et de reconstruction de surfaces permettent de produire les modèles numériques d'objets existants, on parle de rétro conception. La collaboration de ces modules, distribués au sein d'applications spécialisées, a forcé la création d'interfaces à différents niveaux.

3.5.9 Synthèse

Le développement de produit passe par la collaboration de divers secteurs de l'ingénierie. Dans ce cadre, un certain nombre d'applications, aux fonctionnalités spécialisées, doivent collaborer. Situées à différents niveaux fonctionnels de nombreuses interfaces d'échange furent développées. Mais, uniquement basée sur l'échange de données, l'interopérabilité des systèmes d'IAO, se heurte à l'hétérogénéité des représentations.

Nous nous sommes attaché à l'interopérabilité entre systèmes de CAO et systèmes de prototypage et avons montré qu'un certain nombre de traitements était nécessaire pour passer de la géométrie CAO au pilotage du procédé. De la géométrie exacte aux trajectoires de l'outil en passant par la géométrie polyédrique et la géométrie des tranches, un certain nombre de modules fonctionnellement dépendants interviennent.

Distribués entre systèmes, ces modules collaborent en échangeant des données. Cette distribution variant entre couples de systèmes, l'échange de la géométrie s'effectue sous plusieurs formes, exacte, polyédrique ou tranchée.

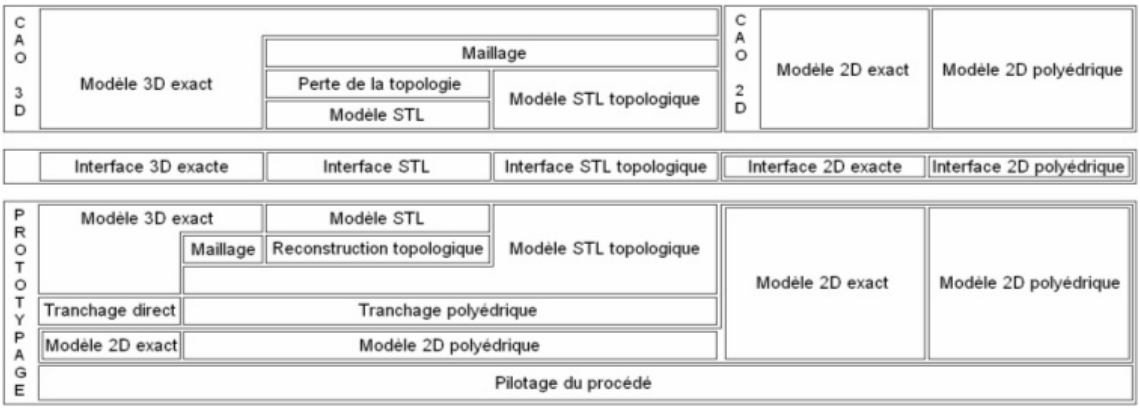


FIGURE 3.15 – *Interopérabilité entre systèmes de CAO et de prototypage*

Qu'ils interviennent à quelque niveau que se soit, les échanges de données impliquent leur duplication et posent le problème de la compatibilité des représentations. Il est clair que les problèmes de compatibilité sont proportionnels à la complexité des représentations ce qui est, à notre avis, le point faible de l'échange de la géométrie exacte.

3.5.10 Solution proposée

L'applicatif des domaines de l'ingénierie est composé d'un certain nombre d'applications spécialisées et fonctionnellement dépendantes. De la même manière, chaque application est composée d'un certain nombre de modules spécialisés et fonctionnellement dépendants. Dans les deux cas, c'est le principe de la décomposition fonctionnelle qui consiste à fractionner un problème difficile en sous problèmes plus simples. Mais qu'il s'agisse des services d'une entreprise ou des composants d'une application il est nécessaire que ces derniers puissent communiquer afin de collaborer. La collaboration des services d'ingénierie passe donc par l'interopérabilité d'applications hétérogènes et par conséquent, de modules distribués entre ces systèmes. Mais, basée sur les échanges de données, celle-ci pose les problèmes de la duplication et de la compatibilité des représentations.

Nous proposons de reconsidérer l'interopérabilité entre systèmes de CAO et systèmes de prototypage sur la base d'une architecture de composants logiciels collaborant via des interfaces bien définies. Cette approche permettrait aux applications d'IAO de collaborer à plusieurs niveaux et de profiter des ressources matérielles et logicielles distribuées sur diverses machines tout en rationalisant leur développement.

Chapitre 4

Proposition de composants et services de prototypage rapide

4.1 Introduction

Depuis l'avènement de l'informatique dans les domaines de l'ingénierie (conception, simulation, fabrication, gestion etc.), diverses catégories de logiciels ont vu le jour. Chacune rendant des services spécifiques à un corps de métiers, les systèmes de CAO pour la conception de produits, les codes de calcul pour la simulation de leurs caractéristiques mécaniques, les systèmes de FAO pour leur fabrication. Cependant même s'ils ont des responsabilités spécifiques, ces systèmes sont loin d'être isolés les uns des autres. En effet, en intervenant à différents niveaux du cycle de développement, ils ne peuvent se détacher des dépendances fonctionnelles associées et doivent collaborer.

Pour des raisons historiques cette collaboration s'effectuait, et s'effectue encore, par des échanges de données au travers de supports de masse et suivant différents formats d'échange. En fournissant les modèles géométriques du produit, les systèmes de CAO tiennent une place centrale dans le processus de développement. Cependant les besoins informatifs des différentes filières ont donné lieu à de nombreuses évolutions des modèles de données correspondants. Il en résulte des formats d'échange basés sur un certain nombre de représentations différentes.

Parmi eux, les formats natifs propriétaires assurent les échanges entre systèmes identiques. Mais ces derniers, en plus de ne pas être ouverts, subissent l'évolution constante des systèmes qui les supportent. En revanche, les formats neutres devaient, comme leur nom l'indique, proposer des solutions d'échange indépendantes des spécificités des systèmes. Cependant, en encodant la structure même des données, ces derniers suivent difficilement l'évolution des représentations (CSG, BRep polyédriques et exactes, courbes et surfaces de types Béziers, splines, NURBS, etc.).

Via ses protocoles d'application, la norme STEP propose la normalisation des schémas de données échangées entre filières. Cependant ces échanges, tels qu'ils existent, ont l'irréductible inconvénient d'impliquer la duplication des données. En effet, entre CAO et simulation, par exemple, la géométrie est intégralement transférée ce qui implique sa duplication mais aussi le support par les systèmes de simulation des modeleurs correspondants.

Il serait à notre avis plus judicieux que celle-ci puisse être consultée plutôt que transférée intégralement. De plus, l'utilisation de catégories de services polymorphes permettrait l'abstraction des différentes représentations, ce qui ne remet pas en cause le transfert de certaines données, essentiellement les paramètres nécessaires à l'invocation de services ainsi que les données résultantes.

L'avènement des réseaux informatiques et le développement des technologies orientées objet ouvrent aujourd'hui, à travers les normes de médiateurs que sont COM et CORBA, de nouvelles perspectives en matière d'interopérabilité des systèmes informatiques. Quelle que soit la nature de l'applicatif, ces derniers assurent en effet l'échange des données nécessaires à l'invocation de services inter-applications, inter-plateformes et inter-réseaux. En connectant les systèmes, ils permettent d'éviter la duplication des données et modèles, et rationalisent ainsi les interactions entre filières. A noter qu'en permettant de consulter plutôt que de transférer l'intégralité des données, les médiateurs apportent également, un progrès en matière de sécurité et de confidentialité. De plus, en fournissant la transparence face à l'évolution des représentations internes, les standards de médiateurs sont, à notre avis, une alternative prometteuse aux formats d'échange. Aussi, sans remettre en cause les efforts importants de spécification de la communauté STEP, les travaux en matière d'interopérabilité des systèmes d'ingénierie devraient se tourner vers la spécification des responsabilités et interfaces de leurs composants.

Ainsi en définissant les interfaces des composants des systèmes délivrant des données de nature géométrique, les services CAO encapsulent les spécificités des systèmes correspondants. Tout système périphérique faisant usage de données géométriques en provenance de systèmes de CAO devrait utiliser ces interfaces. Les systèmes de prototypage, responsables du calcul des sections, devraient, à notre avis, suivre cette règle. Réciproquement, les services des systèmes de prototypage devraient être accessibles et les données qu'ils renferment être réutilisées.

Ainsi, une architecture de composants logiciels orientés prototypage permettrait d'interfacer, à plusieurs niveaux, divers systèmes de prototypage avec divers systèmes de CAO et divers procédés de fabrication. Il serait ainsi possible aux systèmes de prototypage de consulter les caractéristiques de haut niveau que proposent les systèmes de CAO (les «features») plutôt que d'acquérir, via un fichier d'échange, une géométrie parfois dépouillée voir corrompue. Enfin, l'approche médiateur fournirait le cadre nécessaire à la maintenance d'associations, au sein d'un SGDT, des données issues des divers systèmes. Nous proposons donc de spécifier les composants et interfaces des systèmes de prototypage.

4.1.1 Architecture proposée

Comme nous l'avons déjà évoqué, les systèmes de prototypage sont chargés de calculer, à partir de la géométrie CAO, les données nécessaires au pilotage du procédé. La géométrie CAO étant la propriété du système de CAO, celle-ci doit être consultée via les services CAO et non transférée intégralement.

Dans le cas du tranchage direct, le modèle CAO est consulté directement par le module de tranchage afin d'obtenir les informations nécessaires au calcul. Les sections sont ensuite générées dans le modèle de section, à travers les services de son interface.

Dans le cas du tranchage polyédrique, une phase de maillage est indispensable. Dans ce cas, le module de maillage accède directement au modèle CAO et génère le maillage dans le modèle polyédrique qui lui sera consulté par le module de tranchage pour le calcul des sections.

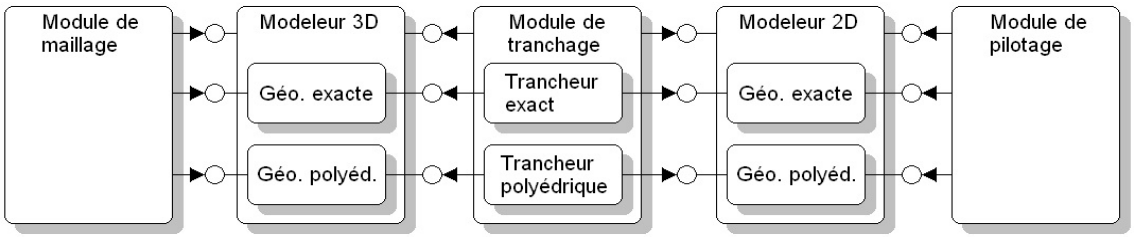


FIGURE 4.1 – *Architecture de composants orientés prototypage*

Dans le cas des procédés basés sur la polymérisation et le frittage, le module de remplissage consulte le modèle de sections pour générer les trajectoires du laser dans le modèle de trajectoires. Celles-ci serviront à leur tour, au pilotage du procédé.

Ces composants pouvant être distribués entre divers systèmes, les différents modules et modeleurs peuvent être répartis de différentes manières entre CAO et prototypage. Ainsi, les composants peuvent être implémentés totalement ou partiellement par l'un ou l'autre des systèmes.

Nous proposons de spécifier les composants relatifs au tranchage polyédrique à l'exception du module de maillage. A des fins de réutilisation de l'existant, nous proposons également des composants additionnels dédiés à supporter l'interface d'échange STL.

4.1.2 Le modèle STL standard

Ce modèle représente les données des solides telles qu'elles sont structurées dans le format STL. Dans ce format, un solide est représenté par une liste de faces triangulaires topologiquement indépendantes. Ce modèle est nécessaire à la réutilisation des fichiers STL existants.

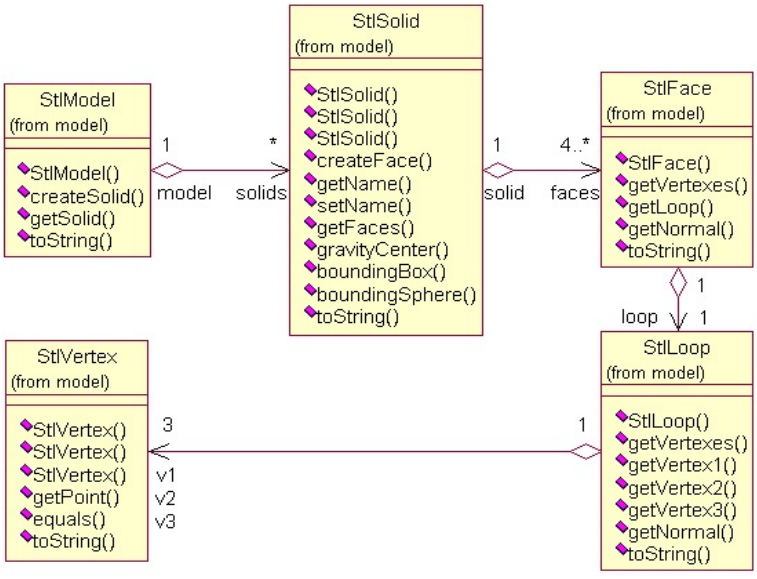


FIGURE 4.2 – *Modèle STL standard*

La classe «StlModel» définit les modèles STL, responsables de la création et du stockage des solides STL. Un solide ne peut être partagé par plusieurs modèles.

La classe «StlSolid» définit les solides composés d'un minimum de quatre faces triangulaires. Les solides ne peuvent partager leurs faces, ils sont par conséquent, responsables de leur stockage et de leur création. Ces dernières peuvent être consultées grâce à l'accesseur correspondant. Cette classe propose des opérations permettant, d'obtenir le volume, le centre de gravité, la boîte et la sphère englobante.

La classe «StlFace» définit les faces des solides. Une face est composée d'un contour de trois sommets et d'un vecteur normal. Une face possède un unique contour et un contour ne peut être partagé par plusieurs faces, celui-ci est donc créé à la construction de la face. Les sommets, le contour et la normale d'une face peuvent être consultés grâce aux accesseurs correspondants.

La classe «StlLoop» définit les contours des faces. Un contour est formé de trois sommets. Les contours ne peuvent partager leurs sommets (cf. absence de topologie). Les sommets et la normale d'un contour peuvent être consultés grâce aux accesseurs correspondants.

La classe «StlVertex» définit les sommets des solides. Un sommet encapsule un point 3D. Un sommet ne peut être partagé par plusieurs faces (cf. absence de topologie). Le point encapsulé par le sommet peut être consulté grâce à l'accesseur correspondant. L'opération «equal» permet de déterminer si deux sommets sont superposés, c'est-à-dire voisin d'une très petite distance ε. Cette opération est utilisée pour reconstruire l'information topologique.

4.1.3 Chargement et sauvegarde au format STL

Même si les systèmes de prototypage devraient accéder à la géométrie au travers des services CAO, le support d'une interface d'échange STL permettrait de réutiliser les données existantes. Nous présentons les composants additionnels permettant le chargement et la sauvegarde de solides dans le modèle STL.

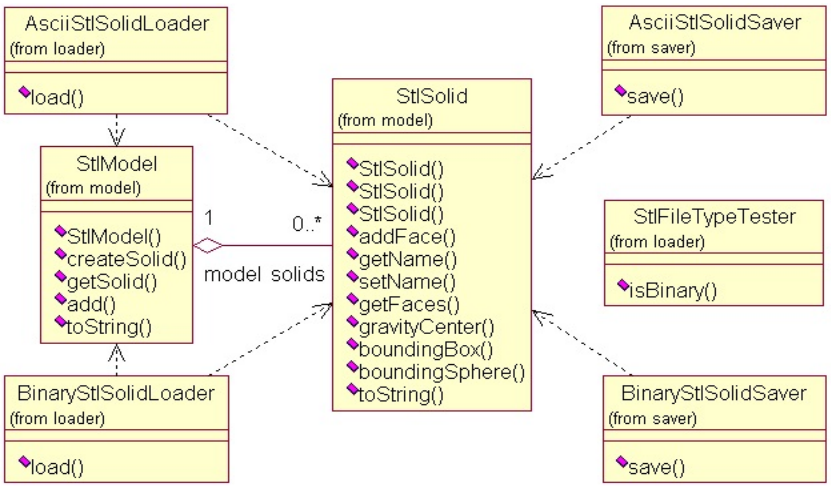


FIGURE 4.3 – *Chargement et sauvegarde des solides au format STL*

La classe «StlFileTypeTester» permet de déterminer le type, ASCII ou binaire, d'un fichier au format STL. Les classes «Ascii/BinaryStlSolidLoader/Saver» offrent les services de

chargement et de sauvegarde des solides aux formats STL, ASCII et binaire.

4.1.4 Modèle STL topologique

Ce modèle représente des solides polyédriques à faces triangulaires mais, contrairement au modèle STL classique, la topologie y est explicite. Nous utilisons la représentation par les frontières ou BRep (Boundary Representation) [53]. L'interface de cette classe est utilisée par le composant responsable du maillage.

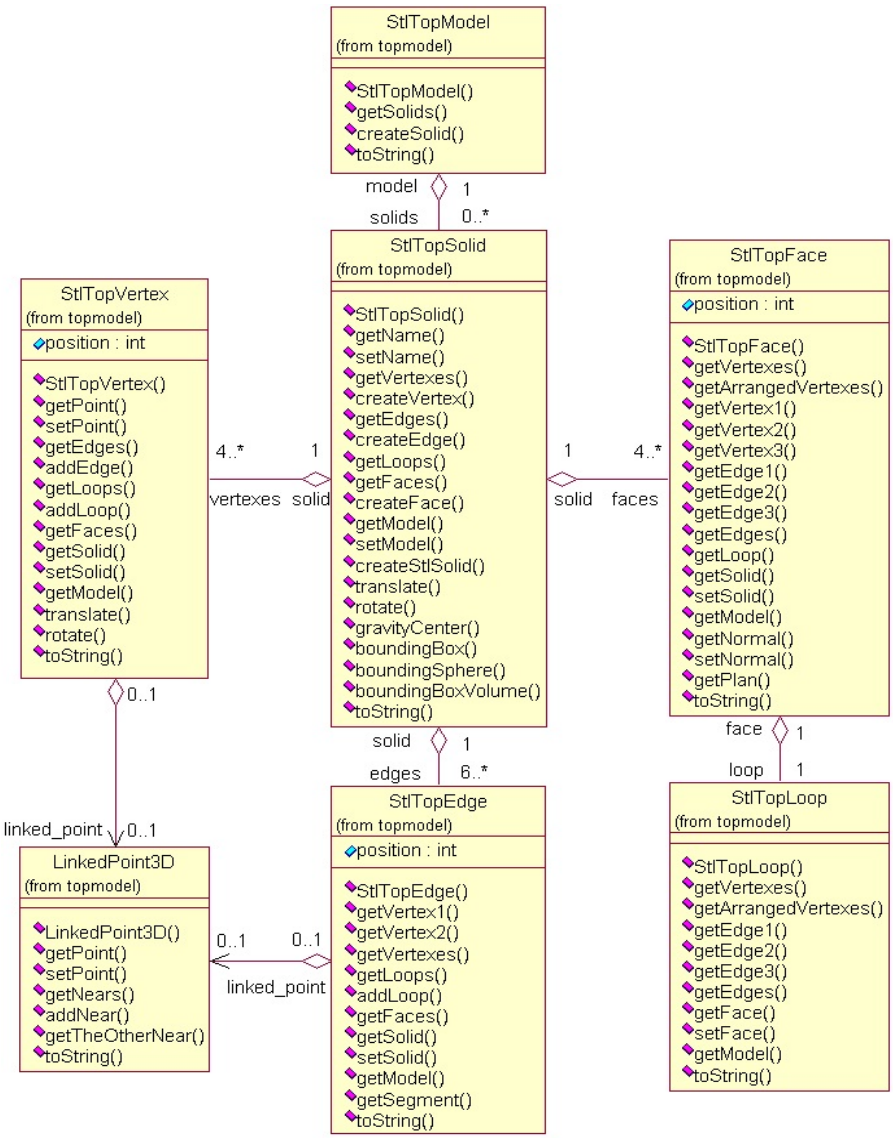


FIGURE 4.4 – *Modèle STL topologique*

Les instances de la classe «StlTopModel» sont responsables de la création et du stockage des solides. Un solide ne peut être partagé par plusieurs modèles.

La classe «StlTopSolid» définit les solides polyédriques à faces triangulaires représentés par les frontières. Un solide est composé d'un minimum de quatre sommets, six arêtes et quatre faces. Les solides ne peuvent partager leurs frontières, ils sont par conséquent responsables de leur stockage et de leur création. Celles-ci peuvent être consultées grâce aux accesseurs correspondants. Cette classe propose des opérations permettant d'obtenir le volume, le centre de gravité, la boîte et la sphère englobante ainsi que la translation et la rotation.

La classe «StlTopFace» définit les faces des solides. Une face est composée d'un contour de trois arêtes et d'un vecteur normal. Une face possède un unique contour ne pouvant être partagé par plusieurs faces, celui-ci est donc créé à la construction de la face. Les sommets, arêtes, le contour et la normale d'une face peuvent être consultés grâce aux accesseurs correspondants.

La classe «StlTopLoop» définit les contours des faces. Un contour est formé de trois arêtes pouvant être partagées par plusieurs contours (deux dans le cas 2-manifold). Les arêtes d'un contour ainsi que la face qu'il délimite peuvent être consultées grâce aux accesseurs correspondants.

La classe «StlTopEdge» définit les arêtes des solides. Une arête est composée de deux sommets pouvant être partagés par plusieurs arêtes, contours et faces. Les extrémités d'une arête, les contours et faces qui la partagent peuvent être consultés grâce aux accesseurs correspondants.

La classe «StlTopVertex» définit les sommets des solides. Un sommet encapsule un point 3D et peut être partagé par plusieurs arêtes, contours et faces. Ces derniers peuvent être consultés grâce aux accesseurs correspondants.

La classe «LinkedPoint» définit les éléments d'une liste, doublement chaînée, de points 3D utilisés par le module de tranchage. Le trancheur associe aux sommets et arêtes coupant le plan de tranchage, un élément de liste renfermant ce point. La topologie du solide permet de chaîner ces éléments et de construire ainsi les contours des sections.

4.1.5 Chargement et sauvegarde au format STL topologique

L'interopérabilité devrait être assurée par des interfaces logicielles plutôt que par des fichiers d'échange. Cela ne signifie pas pour autant que la sauvegarde de la géométrie sous forme de fichiers soit inutile. Celle-ci semble même indispensable à l'archivage sur des supports de masse, mais elle ne devrait pas constituer une solution d'échange entre systèmes hétérogènes. Il est en effet concevable que chaque système puisse archiver ses données dans un format qui lui est propre, tant que celles-ci peuvent être rechargées au sein d'objets supportant les interfaces appropriées. Nous présentons les composants permettant le chargement et la sauvegarde des solides au format STL topologique.

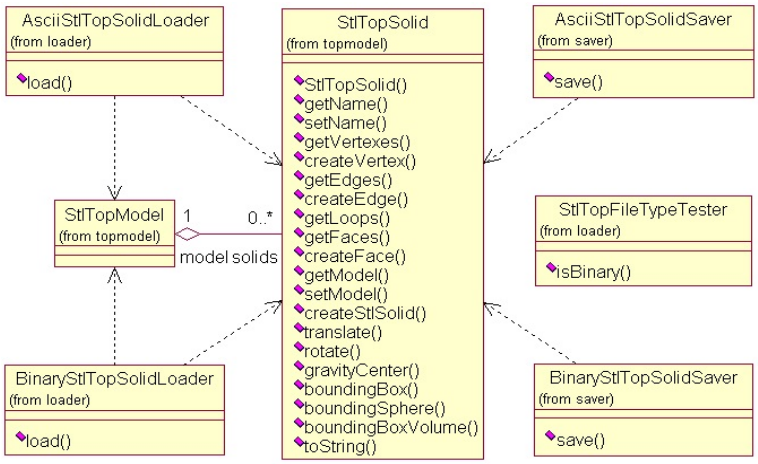


FIGURE 4.5 – *Chargement et sauvegarde des solides au format STL topologique*

La classe «StlTopFileTypeTester» permet de déterminer le type, ASCII ou binaire, d'un fichier au format STL topologique. Les classes «Ascii/BinaryStlTopSolidLoader/ Saver» offrent les services de chargement et de sauvegarde des solides au format STL topologique, ASCII et binaire.

4.1.6 Reconstruction topologique

Pour réutiliser les fichiers STL existants, il est nécessaire de convertir le modèle standard vers le modèle topologique.

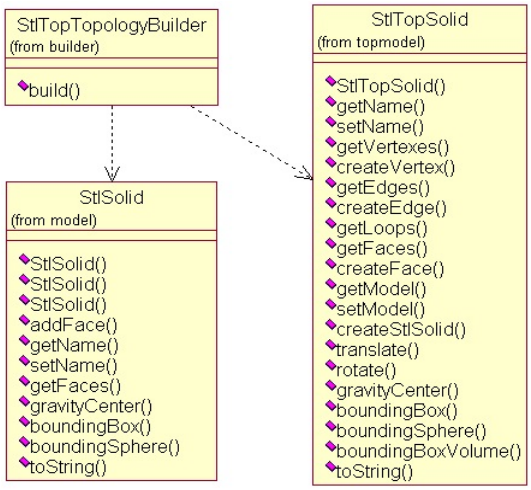


FIGURE 4.6 – *Reconstruction topologique*

La classe «StlTopTopologyBuilder» est chargée expliciter l'information topologique dans le modèle topologique à partir des redondances géométriques du modèle standard.

4.1.7 Le modèle de tranche

Ce modèle représente l'objet comme un ensemble de tranches. Une tranche est un polyèdre planaire non connexe représenté par les frontières. Son interface doit être utilisée par le trancheur.

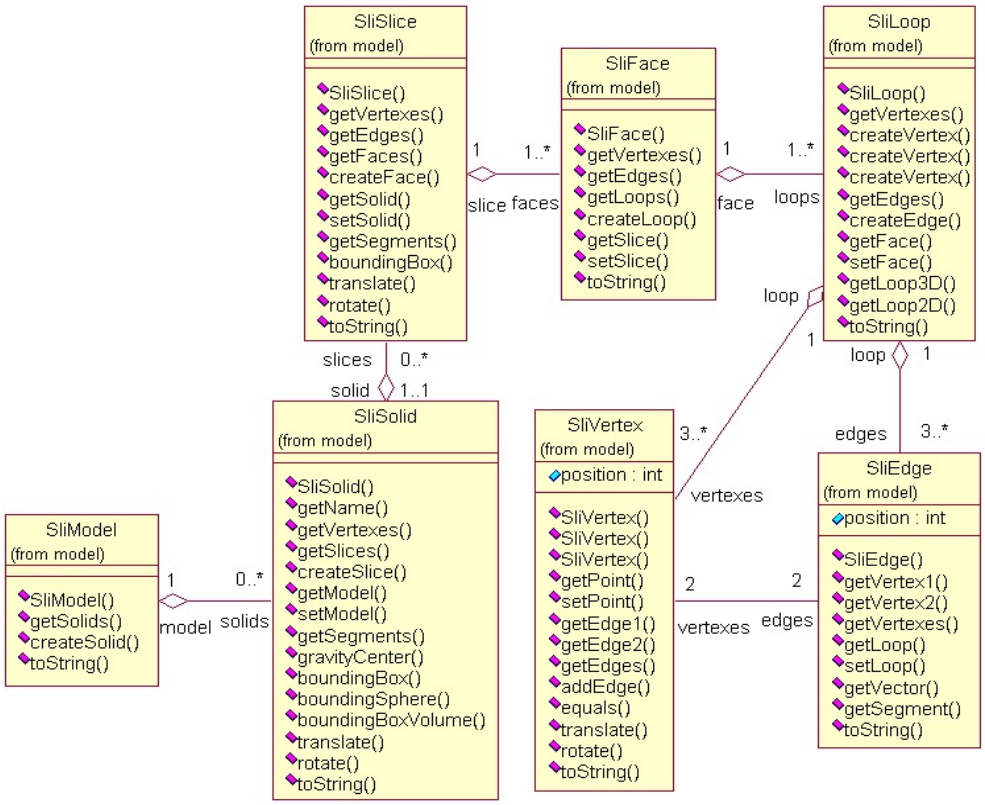


FIGURE 4.7 – *Le modèle de tranche*

Les instances de la classe «SliModel» sont responsables de la création et du stockage des solides tranchés. Un solide ne peut être partagé par plusieurs modèles.

La classe «SliSolid» définit les solides tranchés. Un solide est composé de plusieurs tranches. Les solides ne peuvent partager leurs tranches et sont responsables de leur création et de leur stockage. Ces dernières peuvent être consultées grâce aux accesseurs correspondants. Cette classe propose des opérations permettant d'obtenir le volume, le centre de gravité, la boîte et la sphère englobante ainsi que la translation et la rotation.

La classe «SliSlice» définit les sections (ou tranches) des solides. Une tranche est composée d'un ensemble non connexe, de faces polyédriques planes. Les tranches ne peuvent partager les faces qui les composent et sont responsables de leur création et de leur stockage. Ces dernières peuvent être consultées grâce aux accesseurs correspondants.

La classe «SliTopFace» définit les faces des tranches. Une face est composée d'un contour extérieur et de plusieurs contours intérieurs. Les faces ne peuvent partager les contours qui les composent et sont responsables de leur création et de leur stockage. Ces dernières peuvent être consultées grâce aux accesseurs correspondants.

La classe «SliTopLoop» définit les contours d'arêtes des faces. Un contour est formé d'un minimum de trois arêtes. Les contours ne peuvent partager les sommets et arêtes qui les composent et sont responsables de leur création et de leur stockage. Ces dernières peuvent être consultées grâce aux accesseurs correspondants. Les contours de sommets 3D et 2D peuvent également, être calculés.

La classe «SliTopEdge» définit les arêtes des contours. Une arête est composées de deux extrémités et chaque extrémité est partagée par deux arêtes. Les extrémités d'une arête et le contour qu'elle délimite peuvent être consultés grâce aux accesseurs correspondants.

La classe «SliTopVertex» définit les sommets des contours. Un sommet encapsule un point 2D $\frac{1}{2}$ et est partagé par deux arêtes incidentes. Les arêtes incidentes d'un sommet ainsi que le contour qu'elles délimitent peuvent être consultés grâce aux accesseurs correspondants.

4.1.8 Trancheur

Le calcul des sections joue un rôle central dans un système de prototypage. Nous appelons «trancheur», en anglais «slicer», le composant responsable de ce calcul. Celui-ci doit consulter le modèle solide pour obtenir les informations nécessaires et générer le résultat dans le modèle de tranche.

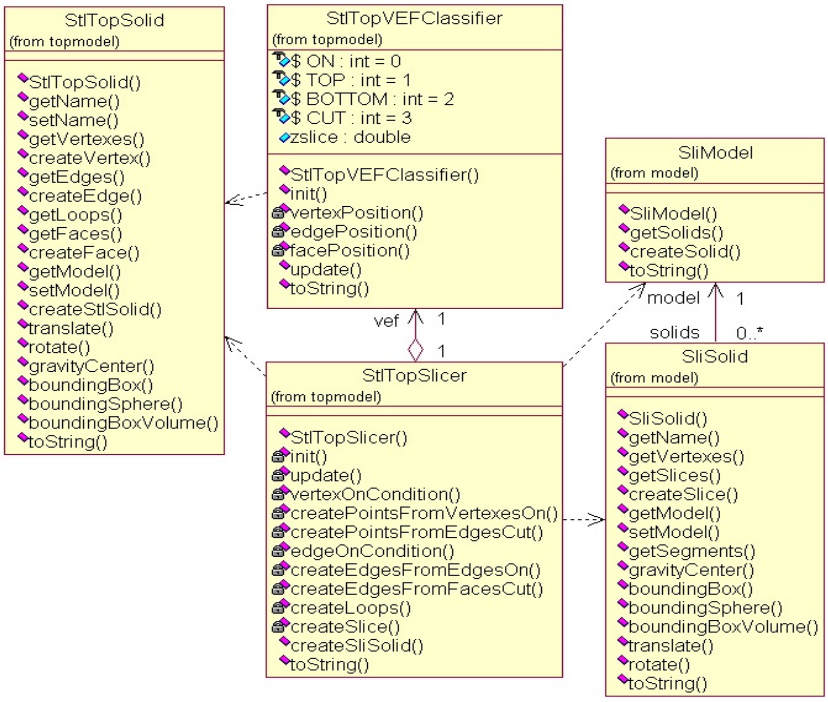


FIGURE 4.8 – *Trancheur*

La classe «StlTopSlicer» définit les trancheurs responsables du calcul des sections d'un solide. Pour effectuer son calcul le trancheur doit identifier les frontières du solide, coupant le plan de tranchage. Pour ce faire il obtient les sommets, arêtes et faces via l'interface de la classe «StlTopSolid» puis procède à leur classification suivant leurs positions par rapport au plan de tranchage.

Les sommets d'une tranche sont issus de l'intersection des arêtes coupées par le plan de tranchage et des sommets qui, situés dans ce plan, représentent des intersections franches du solide (singularités au niveau des sommets). Des instances de la classe «LinkedPoint» renfermant ces intersections sont créées et associées à ce type de sommets et arêtes.

Les arêtes d'une tranche sont issues de l'intersection des faces coupées et des arêtes, qui situées dans le plan, représentent des intersections franches du solide (singularités au niveau des arêtes). Une face coupée possède soit deux arêtes coupées, soit une arête coupée et une singularité au sommet et donc deux instances, préalablement calculées, de la classe «LinkedPoint». Les extrémités d'une arête singulière représentent également des singularités et possèdent chacune une instance de la classe «LinkedPoint». Ces instances sont chaînées deux à deux et les arêtes et contours sont créés. Puis l'arborescence des niveaux d'inclusion des contours permet leur réorientation, la création des faces puis du solide via les interfaces des classes «SliModel» et «SliSolid».

La classe «StlTopVEFClassifier» définit les classificateurs utilisés par les trancheurs. Un trancheur est composé d'un unique classificateur. Les sommets sont classés en trois catégories, ceux qui sont situés au dessus, au dessous et sur le plan de tranchage. Les arêtes et faces sont classées en quatre catégories, au dessus, au dessous, dans et coupée.

4.1.9 Le modèle de trajectoires outils

Ce modèle décrit les trajectoires de balayage de chaque section d'un solide. Ce modèle est nécessaire aux procédés basés sur la polymérisation et le frittage afin de polymériser ou agréger le matériau. Nous appelons cette phase, phase de remplissage.

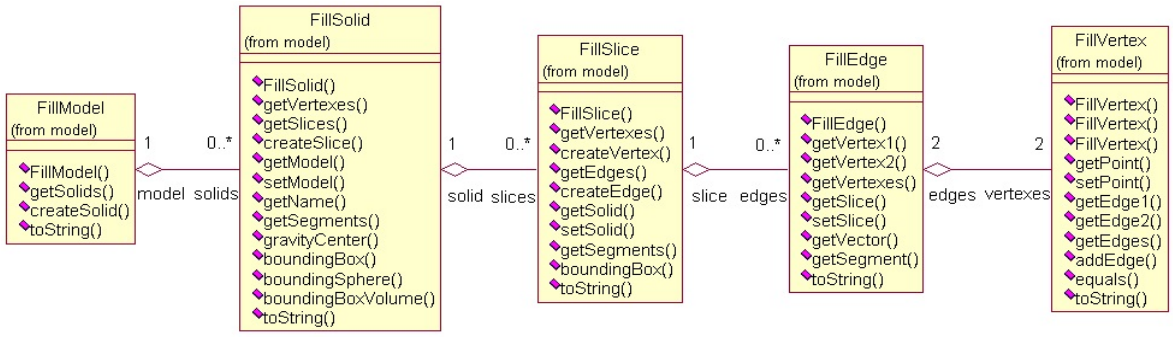


FIGURE 4.9 – *Le modèle de trajectoire*

Les instances de la classe «FillModel» sont responsables de la création et du stockage des trajectoires outils relatives à la fabrication de solides.

La classe «FillSolid» définit les trajectoires outils relatives à l'ensemble des tranches du solide. Un solide ne peut être partagé par plusieurs modèles.

La classe «FillSlice» définit les trajectoires outils relatives à la fabrication d'une section d'un solide. Le balayage d'une section est une ligne brisée pour laquelle, les segments représentent la trajectoire du laser allumé ou éteint (dégagements).

Les classes «FillEdge» et «FillVertex» définissent respectivement les portions rectilignes et les changements de direction des trajectoires de l'outil.

4.1.10 Le remplisseur

Le remplisseur ou «filler» est responsable du calcul des trajectoires de remplissage des sections des solides. Celui-ci consulte le solide tranché pour obtenir les informations nécessaires et générer les trajectoires dans le modèle de remplissage. Ce composant calcule les sections des tranches avec un ensemble de droites. Ce calcul est l'équivalent 2D du tranchage.

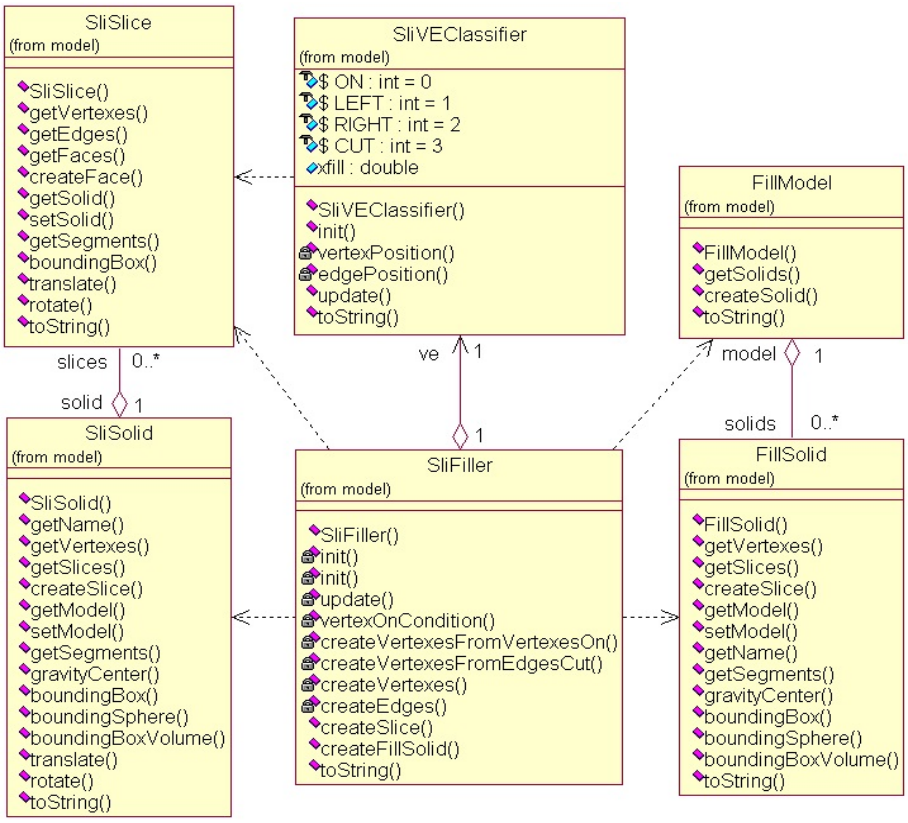


FIGURE 4.10 – *Remplisseur*

La classe «SliFiller» définit les remplisseurs responsables du calcul des sections des tranches d'un solide avec la droite de remplissage. Pour effectuer son calcul le remplisseur doit identifier les frontières des tranches coupant la droite de remplissage. Pour ce faire il obtient les sommets et arêtes via l'interface de la classe «SliSolid» puis procède à leur classification suivant leurs positions par rapport à la droite de remplissage.

Les sommets de la trajectoire sont issus de l'intersection des arêtes coupées par la droite de remplissage et des sommets qui, situés sur la droite, représentent des intersections franches de la tranche (singularités). Les arêtes représentant les trajectoires du laser éteint ou allumé sont créées alternativement entre ces sommets. Les trajectoires sont créées via les interfaces des classes «FillModel» et «FillSolid». La classe «SliVEClassifier» définit les classificateurs utilisés par les remplisseurs. Un remplisseur est composé d'un unique classificateur. Les sommets sont classés en trois catégories, ceux qui sont situés au dessus, au dessous et sur la droite de remplissage. Les arêtes sont classées en quatre catégories, au dessus, au dessous, dans et coupées.

4.2 Conclusion

En fournissant la base de l'interopérabilité, une architecture de composants orientés prototypage rapide permettrait de développer des services de plus haut niveau. Il serait alors possible de maintenir une association entre les données de prototypage et les données CAO. Ainsi, les caractéristiques de conception seraient accessibles par le système de prototypage et inversement, les caractéristiques de prototypage pourraient être intégrées dans le système de CAO. Les services de prototypage permettraient également d'intégrer les données du domaine dans un SGDT, à un niveau de granularité plus fin que celui du fichier.

Même si le rôle de la spécification des services est de s'abstraire de toute implémentation, celle-ci n'en demeure pas moins importante. Nous présentons au chapitre V, une formalisation des algorithmes de tranchage et de remplissage et présentons une implémentation au sein d'une maquette logicielle. Puis nous illustrons le développement de services de plus haut niveau à travers l'implémentation d'un module de simulation de l'état de surface, d'un module d'aide au positionnement et d'un module de calcul de coût.

Chapitre 5

Mise en oeuvre des services de prototypage

5.1 Introduction

Dans ce chapitre, nous présentons une maquette logicielle validant les spécifications des composants de tranchage et de remplissage présentés au chapitre IV. Ces derniers forment le coeur de l'architecture et représentent les fondations des services de plus haut niveau. Nous présentons les concepts théoriques des algorithmes de tranchage et de remplissage puis la mise en oeuvre, au sein d'une application Internet, d'un service d'aide au placement basée sur la simulation de l'effet d'escalier et d'un service d'estimation du coût de fabrication.

Au chapitre IV, nous avons présenté les services de prototypage en faisant abstraction de leur implémentation. Celle-ci est néanmoins importante et se doit d'être fiable et performante. Comme pour la plupart des algorithmes géométriques, une implémentation fiable de l'algorithme de tranchage se révèle complexe et nécessite le traitement de cas algorithmiques singuliers. En effet comme pour les opérations booléennes, l'algorithme de tranchage peut aboutir en une géométrie non régulière et un effort de formalisation important est nécessaire à la prise en compte de telles singularités. De plus, les algorithmes de tranchage et de remplissage consistent en une intersection booléenne de solides respectivement 3D et 2D avec un plan et une droite de tranchage. Nous présentons donc les concepts théoriques qui, développés dans le cadre des opérations booléennes, nous permettent de définir et de mettre en oeuvre le tranchage régularisé 2D et 3D. Finalement, nous montrons les redondances de traitements que présentent les méthodes basées sur l'approche descendante et introduisons l'approche ascendante permettant de les éviter.

5.2 Algorithmes géométriques

Depuis les années soixante, de nombreux efforts sont menés dans l'étude des modèles et des algorithmes géométriques [54], [55]. Parmi ceux-ci, les opérations booléennes, ou ensemblistes permettent de faire évoluer une géométrie par ajout et par retrait de matière. Ils permettent également de détecter les interférences ou encore de simuler la fabrication. Ces algorithmes sont probablement des plus délicats et des plus complexes du domaine de la modélisation

géométrique. Aussi d'importants efforts de formalisation ont été menés afin d'améliorer notre compréhension des cas les plus complexes et de fournir de fiables implémentations. L'algorithme de tranchage consiste à calculer l'intersection d'un solide (sous-ensemble tridimensionnel de l'espace euclidien) avec une surface plane non limitée (sous-ensemble bidimensionnel de l'espace euclidien). Cet algorithme n'est donc pas sans affinité avec l'intersection ensembliste même si dans notre cas les opérandes n'ont pas la même dimension.

5.3 Opérations booléennes

La plupart des modeleurs contemporains supportent les opérations ensemblistes à travers différentes représentations (BRep, CSG, hybride, etc.). Formalisées dans les années soixante dix, dans le cadre du projet d'automatisation de la production de l'université de Rochester et implémentées au sein du système PADL [56], [57], [58], les opérations ensemblistes ont longtemps fait l'objet de rapports internes [56], [59], [60], [61]. Depuis, un nombre relativement restreint de publications ont vu le jour. Situées à un niveau proche de l'implémentation, leur intelligibilité souffre bien souvent d'un faible niveau d'abstraction [62]. Dans les années quatre vingts les notions théoriques et les algorithmes développés à l'université de Rochester ont enfin été publiés [53], [63], [64], [54], [55], [62], [65], [66], [67], [68], [69]. Ces travaux présentent les principales représentations solides utilisées (BRep, CSG, hybride) et le cadre théorique nécessaire à la définition des opérations ensemblistes ainsi que certaines stratégies d'implémentation.

Nous présentons les concepts théoriques développés dans le cadre des opérations ensemblistes sur les solides représentés par les frontières ou BRep (Boundary Representation). L'algorithme correspondant est connu sous le nom d'évaluation et de fusion des frontières (Boundary evaluation and merging) [62]. Afin qu'il soit réutilisable, il est essentiel que la composition ensembliste de deux solides soit également un solide, on parle dans ce cas, de loi de composition interne. Ainsi, l'ensemble des opérations ensemblistes formerait une algèbre fermée sur le corps des solides. Cependant les opérations ensemblistes classiques ne préservent pas la solidité et doivent être remplacées par une version dite régularisée. La théorie relative aux opérations ensemblistes est aujourd'hui bien connue et peut être trouvée dans [60], [63].

L'union, l'intersection et la différence régularisées sont notées $\cup^*, \cap^*, \setminus^*$. L'union régularisée coïncide avec l'union standard, la différence régularisée assure qu'un solide possède toutes ses faces (que celles-ci forment une peau bien fermée) et l'intersection diffère de sa version standard dans les cas de superposition des frontières.

Nous présentons les concepts mathématiques en faisant abstraction des aspects d'implémentation. Ces concepts s'appliquent aussi bien aux solides à faces planes, qu'à faces incurvées, qu'ils soient eulériens (2-manifold) ou non manifold. De plus amples détails peuvent être trouvés dans [60], [55].

5.3.1 Notion de solide

On appelle couramment solide bidimensionnel ou tridimensionnel, un sous-ensemble fini et non nul de l'espace euclidien, respectivement bidimensionnel ou tridimensionnel. Soit un solide S, le complément $\overline{S}$ est défini comme étant l'ensemble des points de l'espace, n'appartenant pas à S.

FIGURE 5.1 – *Un solide bidimentionnel S et son complément $\overline{S}$*

5.3.2 Notion de frontière

Les notions d'intérieur iS, d'extérieur eS et de frontière ∂S peuvent être définies.

$$\begin{cases} p \text{ in } S & \leftrightarrow & p \in iS \\ p \text{ on } S & \leftrightarrow & p \in \delta S \\ p \text{ out } S & \leftrightarrow & p \in eS \end{cases}$$

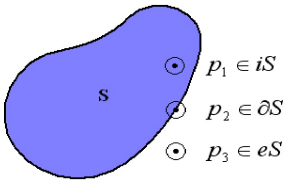


FIGURE 5.2 – *Notions d'intérieur, de frontière et d'extérieur*

— Un point appartient à iS si les points de son voisinage appartiennent à S.
— Un point appartient à ∂S si les points de son voisinage appartiennent à S et à $\overline{S}$.
— Un point appartient à eS si les points de son voisinage appartiennent à $\overline{S}$.

5.3.3 Solide ouvert et fermé

Un solide est dit respectivement, ouvert ou fermé, s'il inclut ou non sa frontière.

$\partial S \not\subset S$ $\qquad$ $\partial S \subset S$

$p \in S \leftrightarrow p \in iS$ $\qquad$ $p \in S \leftrightarrow p \notin eS$

FIGURE 5.3 – *Solide ouvert ou fermé*

On note k l'opération de clôture consistant à inclure la frontière ∂S à l'ensemble S.

5.3.4 Opérations booléennes classiques

Ce sont les combinaisons ensemblistes classiques $\otimes = \cup, \cap, \setminus$ entre sous-ensembles de l'espace euclidien. Soit A et B deux solides bidimensionnels et $S = A \otimes B$ une de leurs combinaisons. Exprimons les conditions d'appartenance d'un point p pour chacun des opérateurs.

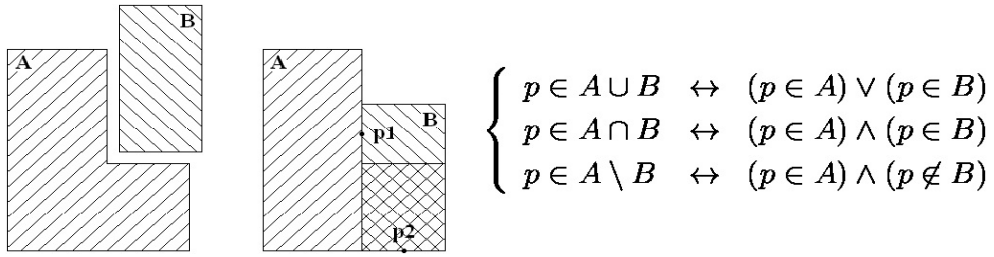
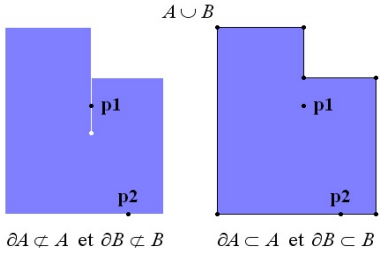


FIGURE 5.4 – *Notions d'intérieur, de frontière et d'extérieur*

Exprimons, pour chacun des opérateurs également, la classification de p par rapport à S ainsi que la représentation graphique du résultat de l'opération dans le cas d'opérandes ouvert ou fermé.

Union

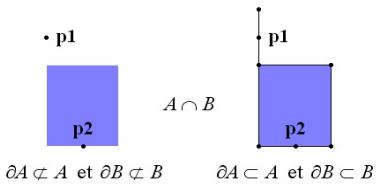


$S = A \cup B$	p *in* B	p *on* B	p *out* B
p *in* A	p *in* S	p *in* S	p *in* S
p *on* A	p *in* S	?	p *on* S
p *out* A	p *in* S	p *on* S	p *out* S

FIGURE 5.5 – *Union non régularisée*

Dans le cas ouvert, le point p_1 est sur une coupure de S alors qu'il est intérieur à S dans le cas fermé. Le point p_2 est dans les deux cas sur la frontière de S, même si celle-ci est exclue de S dans le cas ouvert.

Intersection

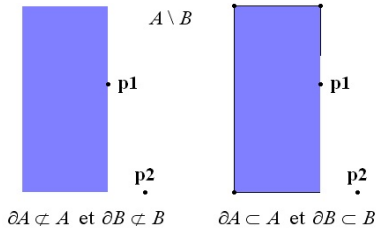


$S = A \cap B$	p *in* B	p *on* B	p *out* B
p *in* A	p *in* S	p *on* S	p *out* S
p *on* A	p *on* S	?	p *out* S
p *out* A	p *out* S	p *out* S	p *out* S

FIGURE 5.6 – *Intersection non régularisée*

Dans le cas ouvert, le point p_1 est extérieur à S alors qu'il est sur une frontière pendante de S dans le cas fermé. Le point p_2 est dans les deux cas sur la frontière de S même si celle-ci est exclue de S dans le cas ouvert.

Différence



$S = A \setminus B$	p *in* B	p *on* B	p *out* B
p *in* A	p *out* S	p *on* S	p *in* S
p *on* A	p *out* S	?	p *on* S
p *out* A	p *out* S	p *out* B	p *out* S

FIGURE 5.7 – *Différence non régularisée*

Dans les deux cas, le point p_1 est sur la frontière de S et le point p_2 est extérieur à S. Dans le cas fermé certaines frontières sont incluses à S alors que d'autres en sont exclues.

Dans tous les cas, on s'aperçoit qu'un point situé simultanément sur les frontières de A et de B est parfois intérieur, extérieur ou sur le bord de S, la classification suivant A et B ne suffit donc pas. Cette difficulté est connue sous le nom d'ambiguïté On/On et peut être levée en regard des voisinages relatifs aux deux opérandes. Mais avant tout, S présente des frontières hétérogènes (partiellement inclues à S), des coupures, ou des arêtes pendantes. Une arête pendante, ou une face dans le cas 3D, est un sous-ensemble de points dont le voisinage est de dimension inférieure à celle des opérandes. Une coupure est le complément d'une entité pendante. Les ensembles possédant de telles pathologies sont dits non réguliers.

5.3.5 Définitions

La notion d'intérieur, de frontière, d'extérieur ainsi que les différentes pathologies évoquées peuvent être définies rigoureusement.

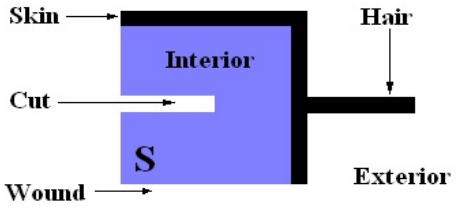


FIGURE 5.8 – *Notions d'intérieur, d'extérieur, de frontière et différentes «pathologies»*

— Frontière ∂S :
Sous-ensemble des points adjacents à S et à son complément $\overline{S}$.
— Intérieur $iS = S \setminus \partial S$:
Sous-ensemble des points dont le voisinage appartient à S.

— Extérieur $eS = \overline{S} \setminus \partial S$:
Sous-ensemble des points dont le voisinage appartient à $\overline{S}$.
— Peau (Skin) $sS = \partial iS \cap S$:
Sous-ensemble de ∂S intérieur à S et adjacent à iS.
— Plaie (Wound) $wS = \partial eS \cap \overline{S}$:
Sous-ensemble de ∂S intérieur à $\overline{S}$ et adjacent à eS.
— Cheveu (Hair) $hS = \partial S \setminus \partial iS$:
Sous-ensemble de ∂S intérieur à S et non adjacent à iS.
— Coupure (Cut) $cS = \partial S \setminus \partial eS$:
Sous-ensemble de ∂S intérieur à $\overline{S}$ et non adjacent à eS.

5.3.6 Régularisation

Les opérations booléennes classiques ne garantissent pas la régularité de leurs résultats et ne forment donc pas une loi de composition interne sur les ensembles réguliers. Pour cela, il faut régulariser leurs résultats. La régularisation d'un ensemble consiste à éliminer les entités pendantes, à combler les coupures et à exclure ou à inclure ses frontières pour obtenir un résultat ouvert ou fermé.

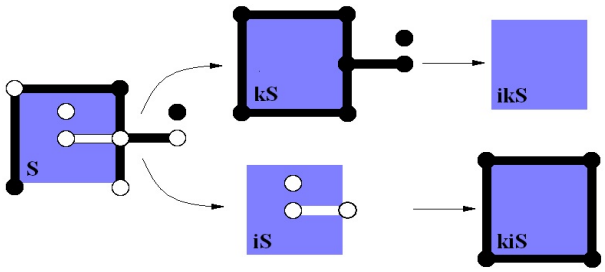


FIGURE 5.9 – *Régularisation*

Un ensemble est dit régulier et ouvert s'il est égal à l'intérieur de sa clôture et il est dit régulier et fermé s'il est égal à la clôture de son intérieur.

5.3.7 Voisinages

Le voisinage $N(p, S, r)$ d'un point p relativement à un solide S est défini comme l'intersection régularisée de S avec une boule de centre p et de rayon infinitésimal r.

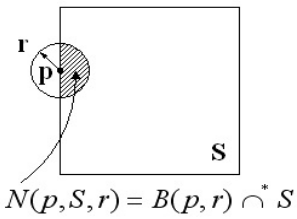


FIGURE 5.10 – *Voisinage d'un point p relativement à un solide S*

Le voisinage $N(p, S, r)$ d'un point p relativement à un solide $S = A \otimes^* B$ est obtenu par combinaison booléenne régularisée des voisinages de p relativement à A et B.

$$N(p, S, r) = N(p, A, r) \otimes^* N(p, B, r).$$

5.3.8 Opérations booléennes régularisées

Les opérations booléennes régularisées notées $\otimes^* = \cup^*, \cap^*, \setminus^*$ résultent en des ensembles réguliers et fermés. Les combinaisons booléennes régularisées sont définies comme étant la clôture de l'intérieur des combinaisons booléennes classiques.

$$A \otimes^* B = ki(A \otimes B)$$

Pour lever, les ambiguïtés On/On la classification $C(X,\ S)$ d'un ensemble X relativement à un ensemble de référence S est augmentée d'informations sur le voisinage dans les cas de superpositions $X\ on\ S$.

$$C(X,\ S) = X\ in\ S,\ (X\ on\ S,\ N(X\ on\ S,\ S)),\ X\ out\ S$$

Le voisinage $N(X\ on\ S,\ S)$ de X relativement à S est obtenu par combinaison régularisée de ses voisinages relativement à A et à B soit :

$$N(X\ on\ S,\ S) = N(X\ on\ S,\ A) \otimes^* N(X\ on\ S,\ B)$$

Union

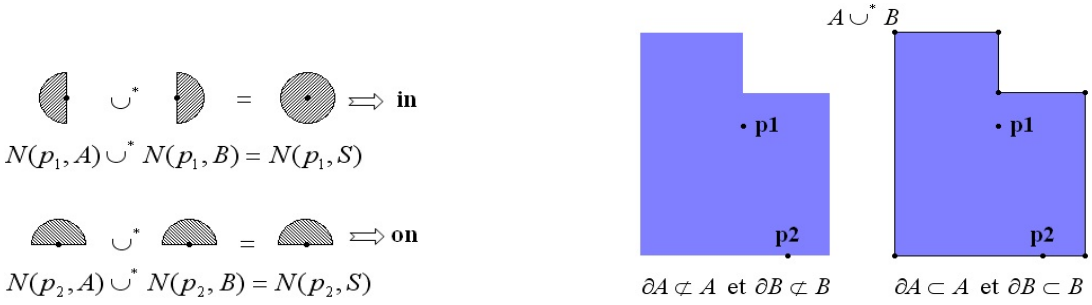


FIGURE 5.11 – *Union régularisée*

Le voisinage $N(p_1, S)$ est un disque plein, p_1 est donc intérieur à S. Le voisinage $N(p_2, S)$ est un demi disque, p_2 est donc sur la frontière de S.

Intersection

FIGURE 5.12 – *Intersection régularisée*

Le voisinage $N(p_1, S)$ est vide, p_1 est donc extérieur à S. Le voisinage $N(p_2, S)$ est un demi disque, p_2 est donc sur la frontière de S.

Différence

FIGURE 5.13 – *Différence régularisée*

Le voisinage $N(p_1, S)$ est un demi disque, p_1 est donc sur la frontière de S. Le voisinage $N(p_2, S)$ est vide, p_2 est donc à l'extérieur de S.

L'évaluation des opérations booléennes se déroule en deux étapes, la segmentation et la classification des frontières des opérandes puis la construction des frontières du résultat.

5.3.9 Classification des sous-ensembles

La classification des sous-ensembles est une généralisation de l'algorithme de découpage ou «clipping» bien connu en informatique graphique. Une entité X est classée comme étant à l'intérieur, sur la frontière ou à l'extérieur de l'ensemble de référence S.

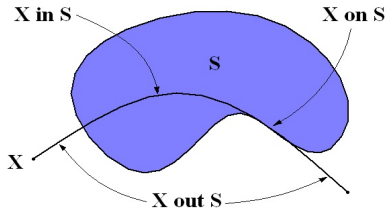


FIGURE 5.14 – *Classification d'une courbe 2D par rapport à un solide 2D*

Dans le cadre des opérations booléennes régularisées les sous-ensembles doivent conservés la topologie de l'ensemble. Une courbe est donc segmentée en sous courbes, une face en sous faces et un solide en sous solides.

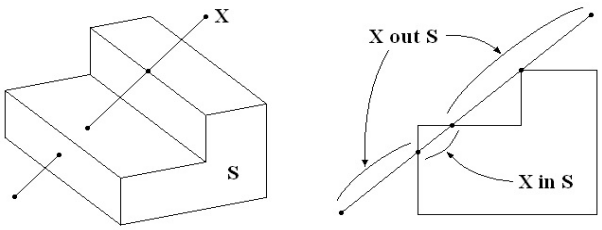


FIGURE 5.15 – *Classification d'un segment X par rapport à un solide S*

Le segment X coupe le solide de référence S en trois points mais l'ensemble X *on* S est vide car il ne contient aucun sous-ensemble topologiquement équivalent à X, techniquement :

$$\begin{cases} X\ in\ S & = & k_x i_x(X \cap i_s S) \\ X\ on\ S & = & k_x i_x(X \cap \delta_s S) \\ X\ out\ S & = & k_x i_x(X \cap e_s S) \end{cases}$$

Avec k_x et i_x dénotant respectivement la clôture et l'intérieur de X relativement à sa topologie et i_s, δ_s, e_s respectivement l'intérieur, la frontière et l'extérieur de S relativement à sa topologie. Les propriétés mathématiques des sous-ensembles sont bien connues et peuvent être trouvées dans [61], [63], [64].

5.3.10 Combiner les classifications

Nous avons vu que la classification d'un ensemble X par rapport à S est obtenue par combinaison des classifications de X par rapport à A et B. Dans les cas d'ambiguïtés On/On, la classification est obtenue par combinaison des voisinages de X par rapport à A et B. Le

voisinage des points intérieurs ou extérieurs est respectivement une boule pleine ou vide. Dans le cas des points situés sur la frontière il faut distinguer trois cas :

- Le voisinage tridimensionnel des faces, voisinage relatif à un solide tridimensionnel, des points intérieurs aux faces. Il est déterminé par les normales des faces et la convention faites sur le coté matière.
- Le voisinage tridimensionnel des arêtes, voisinage relatif à un solide tridimensionnel, des points intérieurs aux arêtes. Il est déterminé par des secteurs contenant la matière autour de l'arête. Chaque secteur est représenté par les surfaces qui le limitent. Il est déterminé par les normales et tangentes des faces adjacentes. Une surface divise l'espace 3D en deux demi-espaces et une courbe sépare une surface en deux demi-surfaces. Ces derniers sont déterminés par les normales et tangentes aux faces incidentes de l'arête. Dans le cas d'une arête partagée par plus de deux faces, soit dans le cas d'une géométrie non eulérienne (non 2-manifold) le voisinage peut être représenté par une liste de secteurs [70], [71], [72].
- Le voisinage tridimensionnel des sommets, voisinage des sommets d'un solide relativement à lui même. Il peut être représenté par la notion de secteurs généralisés délimités par au moins trois surfaces [73]. Cependant leur représentation est complexe et n'est pas nécessaire à tout algorithme.

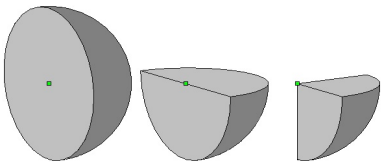


FIGURE 5.16 – *Voisinages tridimensionnels des faces, arêtes et sommets*

Les voisinages bidimensionnels, relativement aux faces et non aux solides sont également utiles. On peut distinguer deux cas :

- Le voisinage bidimensionnel des arêtes, voisinage relatif à une face, des points intérieurs aux arêtes. C'est l'analogue bidimensionnel du voisinage tridimensionnel des faces et peut être représenté par la normale à l'arête indiquant le coté où se trouve la face. Cette information est souvent représentée par orientation de l'arête. C'est le cas du modèle BRep à arêtes ailées (winged-edge BRep) [70], [73].
- Le voisinage bidimensionnel des sommets, voisinage des sommets d'une face relativement à celle-ci. C'est l'analogue bidimensionnel du voisinage tridimensionnel des arêtes et peut être représenté par les secteurs limités par les courbes incidentes.

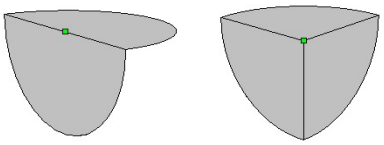


FIGURE 5.17 – *Voisinages bidimensionnels des arêtes et sommets*

La combinaison des voisinages tridimensionnels des faces superposées consiste à effectuer des opérations logiques sur les bits de signe représentant le côté matière. La combinaison des voisinages tridimensionnels des arêtes superposées est relativement complexe. Il consiste à ordonner les surfaces incidentes autour de l'arête et à sélectionner les secteurs appropriés suivant l'opération booléenne à effectuer. Dans le cas de surfaces incurvées de mêmes tangentes la courbure doit être utilisée.

5.3.11 Classification des points

Le lecteur pourra trouver dans [74] plusieurs algorithmes de classification des points relativement à un solide. Parmi ceux-ci, l'algorithme du lancer de rayon consiste à calculer les intersections avec le solide d'une demi-droite issue du point à classer. Si le nombre d'intersections est nul ou pair le point est extérieur sinon il est intérieur. Cependant, il est possible que certaines intersections soient des singularités. Dans ce cas, il faut prendre en compte le voisinage 3D de ces arêtes ou sommets, ce qui est relativement coûteux. Une approche alternative consiste à itérer l'algorithme jusqu'à ce qu'il n'y est pas de singularité et de consulter le voisinage 3D de la face contenant l'intersection la plus proche de l'origine.

5.3.12 Classification des arêtes

La classification des arêtes s'effectue en regard du type de transition des intersections de l'arête et du solide. Celui-ci est déterminé par examen des voisinages 3D des intersections. Tout changement de classification s'effectue en des points d'intersection mais tous les points d'intersection ne représentent pas un changement de classification.

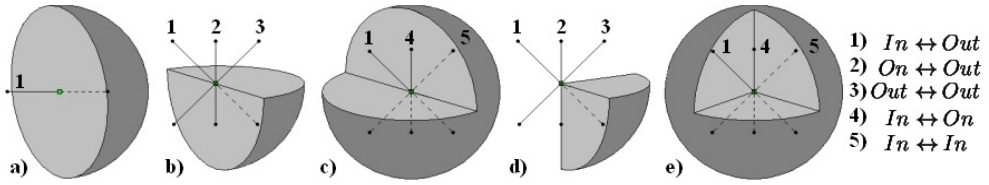


FIGURE 5.18 – *Types de transitions*

Le type de transition des intersections intérieures aux faces est déterminé en regard du vecteur directeur de l'arête et de la normale à la face (fig. 5.18a). Cependant les intersections peuvent se situer à l'intérieur d'arêtes (fig. 5.18 b et c) ou de sommets (figures 5.18 d et e), on parle dans ces deux cas de singularités. La résolution des singularités se fait en regard des voisinages 3D des arêtes et sommets. Les singularités aux sommets sont plus difficiles à résoudre mais, heureusement, celles-ci peuvent être évitées par classification des milieux des segments.

La classification des arêtes consiste à calculer ses intersections avec les faces du solide de référence. Si une arête est contenue dans la surface porteuse d'une des faces du solide, cette surface est ignorée dans le calcul d'intersection. Les points d'intersection sont obtenus avec les faces adjacentes. La classification des sous segments contenus dans la surface d'une face du solide est déduite en regard des voisinages 2D des intersections. Dans le cas de singularités aux sommets, un algorithme de lancer de rayon 2D permet de classer les milieux des segments.

La classification des arêtes consiste à calculer ses intersections avec les faces du solide de référence et de les organiser de manière à ce qu'elles délimitent des sous-ensembles de classification homogène, déterminée au regard des voisinages ou par classification des milieux, évitant ainsi les singularités.

5.3.13 Classification des faces

De manière analogue, la classification des faces consiste à calculer leurs intersections avec les faces du solide de référence et de les organiser de manière à ce qu'elles délimitent des sous-ensembles de classification homogène, déterminés en regard des voisinages ou par classification d'un point intérieur.

5.3.14 Évaluation et fusion des frontières

Il faut tout d'abord identifier les faces candidates, c'est-à-dire les faces susceptibles d'appartenir au résultat de l'opération booléenne. L'ensemble des frontières d'une combinaison booléenne est inclus dans l'ensemble des frontières de ses opérandes, techniquement $\delta(A \otimes B) \subseteq \delta A \cup \delta B$. Les faces candidates sont donc l'ensemble des faces de A et de B. Pour construire les faces de la combinaison booléenne, il faut identifier les arêtes délimitant les sous-ensembles, de classification homogène, des faces de A et B. Celles-ci étant sur la frontière de S, les arêtes qui les délimitent le sont également.

Cependant, certaines arêtes ne délimitent pas forcément des faces dites maximales, c'est-à-dire des sous-ensembles les plus larges possibles des faces de S. Ces arêtes sont caractérisées par un voisinage 3D de la forme d'une demi-boule. Celles-ci peuvent donc être exclues.

Les arêtes du solide $S = A \otimes B$ se divisent en deux catégories, les arêtes propres et les arêtes intersections. La classification des arêtes propres d'un opérande relativement à lui même est triviale et doit être combinée à sa classification relativement à l'autre opérande.

Les arêtes intersections sont les intersections d'intérieurs de faces. Considérer l'intérieur des faces permet de ne pas confondre les arêtes intersections avec les arêtes propres situées à l'intérieur de faces.

5.3.15 Efficacité et amélioration

Les performances de cet algorithme peuvent être améliorées par différents filtres basés sur la localisation des entités. De plus, la classification des arêtes propres peut être déduite de la classification des arêtes intersections et inversement mais ces techniques ne sont pas généralisables aux entités incurvées et compliquent considérablement les algorithmes correspondants.

Enfin, cet algorithme possède des redondances de traitements. En effet, une arête appartenant à au moins deux faces, si une intersection existe entre deux arêtes celle-ci est calculée quatre fois. De plus, un sommet appartenant à un nombre indéfini de faces, si une intersection entre une arête et un solide intervient en un sommet, celle-ci est calculée un nombre indéfini de fois dans le cadre des calculs de son intersection avec les faces adjacentes au sommet. Ces redondances peuvent être évitées par le marquage des entités traitées. Une autre approche consisterait à traiter prioritairement de telles singularités (intersections d'arêtes intervenant

en des sommets ou à l'intérieur d'arêtes) puis de traiter, par la suite seulement, les intersections d'arêtes avec les faces du solide. Notre apport se situe dans le développement d'une telle approche dans le cadre de l'algorithme de tranchage.

5.3.16 Conclusion

Nous avons omis plusieurs aspects connexes importants. L'algorithme décrit précédemment produit un BRep dans lequel certaines arêtes peuvent être répétées ou superposées et où les relations de connectivité ne sont pas explicites. Ces tâches peuvent être effectuées à posteriori ou maintenues durant l'évaluation, ce qui complique significativement le code associé. La complexité maximale de l'algorithme est d'ordre polynomial alors que la complexité moyenne n'est pas connue. Cet algorithme est le plus fondamental et le plus délicat du domaine de la modélisation géométrique. La classification et la combinaison des voisinages sont des notions fondamentales de l'algorithme. Certains aspects restent néanmoins méconnus. En effet, quelle influence ont les informations de connectivité sur les performances de l'algorithme ! Est il plus efficace de les maintenir ou de les reconstruire à posteriori ? Est il plus simple et plus efficace d'éviter ou de résoudre les singularités ?

5.4 Algorithme de tranchage

Les algorithmes de tranchage 2D et 3D consistent à calculer l'intersection d'un solide respectivement 2D avec une courbe et 3D avec une surface non limitée. L'algorithme de tranchage est donc analogue à l'intersection booléenne mais avec des opérandes de dimensions différentes. A noter que, dans le cadre du prototypage rapide, l'algorithme de tranchage 3D est chargé, du calcul des sections de l'objet et l'algorithme de tranchage 2D, du calcul des trajectoires de remplissage des sections. Comme pour l'intersection booléenne classique, le tranchage classique peut produire un résultat non régulier, celui-ci doit donc être remplacé par sa version régularisée.

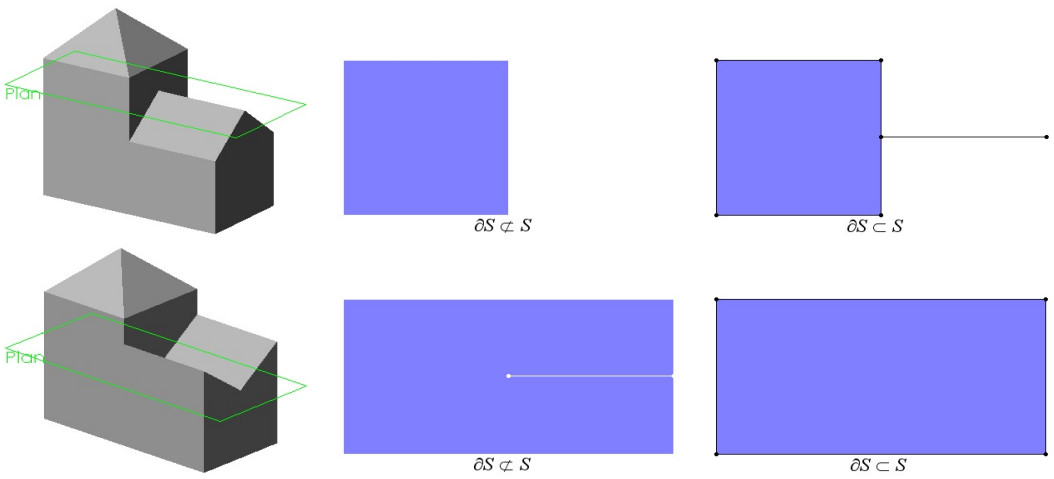


FIGURE 5.19 – *Tranchage classique de solides considérés successivement ouverts et fermés*

Les sections résultantes contiennent des arêtes pendantes ou des coupures. Celles-ci sont issues de singularités ne représentant pas des transitions de classification de type $S \leftrightarrow \overline{S}$. Comme pour les opérations booléennes celles-ci doivent être régularisées par combinaisons des voisinages.

5.4.1 Limitations

Pour des raisons de simplicité nous nous limitons aux solides polyédriques à faces triangulaires (format STL). Nous avons déjà évoqué le tranchage de solides représentés par des surfaces, ces algorithmes sont bien plus complexes et délicats [75]. Les travaux dans ce domaine portent essentiellement sur les calculs d'intersection entre surfaces, sur la détection des transitions et des contours [35]. Le tranchage polyédrique à l'avantage de ne faire intervenir que des calculs d'intersection entre plans et faces triangulaires et la qualité d'être numériquement non ambiguë. Nous nous limitons également aux solides eulériens c'est-à-dire non connectés ou encore 2-manifold. Un solide est dit non eulérien si le voisinage de tous les points de sa frontière est homéomorphique à une portion de boule. Un solide est connecté ou non manifold si le voisinage d'au moins un point de sa frontière est homéomorphique à plusieurs portions de boule.

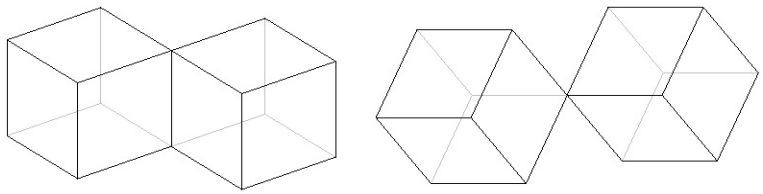


FIGURE 5.20 – *Solides connectés ou non eulériens (non manifolds)*

Du point de vue théorique, Les solides connectés sont valides et traités grâce à des représentations adéquates des voisinages [70], [73], [71], [72]. Du point de vue de la conception, leur utilité reste discutable, certains systèmes les autorisant (Catia) d'autres non (SolidWorks, Pro-Engineer).

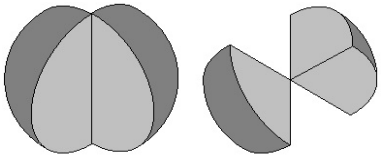


FIGURE 5.21 – *Voisinages respectifs d'une arête et d'un sommet non manifold*

La figure 5.21 représente les voisinages d'arêtes et de sommets non manifolds, tous deux étant homéomorphiques à deux secteurs de boule. Les algorithmes traitant ce genre de géométries traitent les voisinages correspondants comme un ensemble de voisinages 2-manifolds [76]. Notre algorithme s'attache aux voisinages 2-manifold mais les concepts développés peuvent être réutilisés dans les cas non manifolds.

Le tranchage polyédrique restant massivement utilisé et les géométries connectées intervenant exceptionnellement en conception, ces limitations restent peu restrictives. Au contraire, elles devraient pouvoir être exploitées afin d'améliorer les performances et la fiabilité des algorithmes correspondants.

5.4.2 Approche proposée

Comme pour les opérations booléennes, une approche consisterait à calculer les intersections avec la droite ou le plan de tranchage des différentes entités, dans l'ordre décroissant de leur dimension, faces, arêtes et sommets. Nous qualifions cette approche, d'approche descendante. Comme nous l'avons évoqué à la section 5.3.15, cette approche a l'inconvénient de la redondance des traitements entraînant une perte de performances et nécessitant la fusion des diverses instances d'une même entité. Comme nous l'avions évoqué, ces redondances peuvent être évitées dans une approche ascendante consistant à calculer les intersections avec le plan de tranchage des différentes entités, dans l'ordre croissant de leur dimension, sommets, arêtes et faces. Une telle approche traite donc prioritairement les cas de singularité. Notre apport se situe donc dans l'élimination, des redondances des traitements et du post traitement associé, par l'utilisation d'une approche ascendante traitant prioritairement les singularités.

5.4.3 Classification des sous-ensembles

Si l'on note A la droite ou le plan de tranchage et B le solide à trancher, la section résultante S est le sous-ensemble régularisé du plan ou de la droite intérieur au solide. Exprimons la classification suivant S d'un point en fonction de sa classification par rapport à A et B. Il faut noter qu'une surface et une courbe ne définissent pas d'intérieur ni d'extérieur respectivement au sens 3D et 2D du terme mais en définissent au sens 2D et 1D. De plus celles-ci ne sont pas limitées et ne possèdent donc pas de bords. Un point devrait donc être classé soit à l'intérieur, soit à l'extérieur d'une surface ou d'une courbe mais relativement à sa topologie. Cependant afin de ne pas mélanger des notions relatives à des topologies différentes (celle de l'entité de tranchage : surface ou courbe et celle de l'objet à trancher : solide 3D et 2D) nous faisons l'abus de langage suivant : un point est sur la surface ou la courbe de tranchage s'il lui est intérieur relativement à sa topologie.

$S = A \cap^* B$	p *in* B	p *on* B	p *out* B
p *on* A	p *in* S	?	p *out* S
p *out* A	p *out* S	p *out* B	p *out* S

TABLE 5.1 – *Classification d'un point p suivant la section S*

Comme pour les opérations booléennes les cas d'ambiguïté On/On ne représentent pas forcément des transitions de classification de type $S \leftrightarrow \overline{S}$. Ces ambiguïtés doivent être résolues par combinaison des voisinages de p par rapport à A et B.

$$N(p,\ S) = N(p,\ A) \cap^* N(p,\ B)$$

5.4.4 Combinaison des voisinages

Le voisinage $N(p,\, A)$ est toujours un segment en 2D et un disque en 3D. Pour le voisinage $N(p,\, B)$ rappelons les cas à distinguer en 2D et 3D :

— Le voisinage bidimensionnel des sommets situés à l'intérieur d'arêtes de B.
— Le voisinage bidimensionnel des sommets de B.
— Le voisinage tridimensionnel des points situés à l'intérieur de faces de B.
— Le voisinage tridimensionnel des sommets situés à l'intérieur d'arêtes de B.
— Le voisinage tridimensionnel des sommets de B.

5.4.5 Originalité de notre approche

L'originalité de notre approche est de prendre en compte la spécificité du voisinage $N(p,\, A)$ pour déterminer le type de transition dans les cas de singularité. Pour cela, nous procédons à l'augmentation de la classification $C(X\, on\, B,\, A) = On,\, Out$ des entités de B par rapport à la droite ou au plan de tranchage A. Cette nouvelle classification nous permettra de décrire qualitativement le voisinage, relativement à A, des singularités et de déterminer, par combinaison du voisinage, relativement à B, le type de transition.

En 2D, il faut classer les sommets et arêtes de B par rapport à la droite de tranchage A. Pour cela, nous procédons à l'orientation arbitraire de la droite afin que celle-ci délimite l'espace 2D en deux demi-espaces droit et gauche. Ainsi, la classification Out des sommets, notés V, est étendue en $Right$ et $Left$.

$$C(V\, on\, B,\, A) = On,\, Right,\, Left$$

Pour les arêtes notées E, la classification est étendue de manière analogue mais un quatrième cas doît être considéré. Contrairement aux sommets, les arêtes peuvent être situées de part et d'autre de la droite de tranchage. Dans ce cas, l'arête coupée est classée Cut.

$$C(E\, on\, B,\, A) = On,\, Right,\, Left,\, Cut$$

En 3D, il faut classer les sommets, arêtes et faces de B par rapport au plan de tranchage A. Comme en 2D, il faut orienter le plan afin qu'il délimite deux demi-espaces inférieur et supérieur. Ainsi, la classification Out des sommets est étendue en Top et $Bottom$.

$$C(V\, on\, B,\, A) = On,\, Top,\, Bottom$$

Pour les arêtes et les faces, notées respectivement E et F, la classification est étendue de manière analogue à celle des arêtes en 2D.

$$C(X\, on\, B,\, A) = On,\, Top,\, Bottom,\, Cut \quad avec\, X = E,\, F$$

Pour harmoniser les notations 2D et 3D, nous ferons volontairement en 2D, un abus de langage en considérant les demi-espaces droit et gauche comme étant respectivement supérieur et inférieur.

$$C(X\, on\, B,\, A) = \begin{cases} On,\, Top,\, Bottom & pour\, X = V \\ On,\, Top,\, Bottom,\, Cut & pour\, X = E,\, F \end{cases}$$

Cette classification sera utilisée indifféremment en 2D et en 3D.

5.5 Tranchage 2D

Il s'agit de calculer les sous-ensembles régularisés, et appartenant à B, de la droite de tranchage A (arêtes). Ces arêtes sont délimitées par les intersections de la droite avec la frontière δB (sommets). Ces sommets se situent soit à l'intérieur d'arêtes classées Cut soit sur des sommets, classés On (singularités), et représentant une transition de classification de type $B \leftrightarrow \overline{B}$. Alors que le premier cas est trivial, le deuxième nécessite de déterminer le type de transition aux sommets. Notre approche consiste à déterminer ce type de transition en regard de la classification, par rapport à la droite de tranchage, des sommets et arêtes de B. Nous verrons, par la suite, que pour décrire qualitativement les types de voisinages dans les cas de singularité, la classification des arêtes devra être augmentée.

5.5.1 Classification des sommets

Si l'on considère une droite orientée, un observateur marchant sur le plan et suivant cette droite classerait les sommets alentours comme étant sur, à droite ou à gauche de la droite. Par abus de langage nous dirons que les points situés à droite et à gauche se situent respectivement au dessus et au dessous de la droite. Nous définissons la fonction position π d'un sommet $v(x_v, y_v, z_v)$ par rapport à la droite $d(p, \overrightarrow{d})$ définie par un point de passage p et un vecteur directeur $\overrightarrow{d}$.

$$\begin{array}{rcl} \pi : \mathbb{R}^2 & \rightarrow & \{On,\ Top,\ Bottom,\ Cut\} \\ v & \rightarrow & \pi(v) \end{array} \qquad \pi(v) = \begin{cases} On & \Leftrightarrow & \overrightarrow{pv} \wedge \overrightarrow{d} = 0 \\ Top & \Leftrightarrow & \overrightarrow{pv} \wedge \overrightarrow{d} > 0 \\ Bottom & \Leftrightarrow & \overrightarrow{pv} \wedge \overrightarrow{d} < 0 \end{cases}$$

Dans le cadre du prototypage rapide la droite de tranchage représente la trajectoire de balayage de l'outil. Celle-ci est orientée suivant l'axe $\overrightarrow{x}$ ou $\overrightarrow{y}$ de la machine. Nous choisissons arbitrairement l'axe $-\overrightarrow{y}$, la classification devient :

$$\pi(v) = \begin{cases} On & \Leftrightarrow & x_v = x_p \\ Top & \Leftrightarrow & x_v < x_p \\ Bottom & \Leftrightarrow & x_v > x_p \end{cases}$$

Cette considération nous permet de minimiser les traitements et d'éviter les arrondis numériques qui seraient intervenus lors des calculs de distance entre sommets et droite.

5.5.2 Classification des arêtes

Rappelons que nous nous attachons au tranchage polyédrique uniquement. Dans ce contexte, la classification d'une arête linéaire peut être déterminée d'après la classification de ses extrémités. Nous définissons donc la fonction position π d'une arête non orientée $e(v_1, v_2)$ par rapport à la droite $d(p, \overrightarrow{d})$ ainsi :

$$\begin{array}{rcl} \pi : \mathbb{R}^2 \times \mathbb{R}^2 & \rightarrow & \{On,\ Top,\ Bottom,\ Cut\} \\ e & \rightarrow & \pi(e) \end{array}$$

cas	$\pi(v_1)$	$\pi(v_2)$	$\pi(e)$
1	*On*	*On*	*On*
2	*On*	*Top*	*Top*
3	*On*	*Bottom*	*Bottom*
4	*Top*	*Top*	*Top*
5	*Top*	*Bottom*	*Cut*
6	*Bottom*	*Bottom*	*Bottom*

TABLE 5.2 – *Classification des arêtes non orientées*

Notons que nous n'avons pas fait figurer tous les couples possibles, certains étant, d'un point de vue géométrique, identiques par permutation. Profitons en pour exprimer le nombre de cas possibles.

Soit n variables pouvant chacune prendre p valeurs, le nombre de n-uplets invariants par permutations est de $f(n,p) = f(n-1,p) + f(n,p-1)$ avec $f(1,p) = p$ et $f(n,1) = 1$. Soit dans notre cas, v_1 et v_2 à valeurs dans $\{On, Top, Bottom\}$ soit $f(2,3) = 6$ couples possibles.

Cependant, cette classification reste insuffisante à la détermination des types de transition de classification dans les cas de singularités. Nous proposerons plus tard, son augmentation.

5.5.3 Transition de classification aux sommets classés «On»

Les intersections de la droite de tranchage avec le solide se situent soit à l'intérieur d'arêtes classées *Cut*, soit sur des sommets, classés *On* (singularités) et représentant des transitions de classification de type $B \leftrightarrow \overline{B}$. Alors que le premier cas est trivial, le deuxième nécessite de déterminer le type de transition aux sommets.

Pour cela, nous examinons les combinaisons possibles entre les positions des arêtes incidentes aux sommets classés *On*. Comme B est 2-maniflold, un tel sommet possède deux arêtes incidentes ne pouvant être classées *Cut*, soit $f(2,3) = 6$ combinaisons.

cas	$\pi(e_1(v))$	$\pi(e_2(v))$
1	*On*	*On*
2	*On*	*Top*
3	*On*	*Bottom*
4	*Top*	*Top*
5	*Top*	*Bottom*
6	*Bottom*	*Bottom*

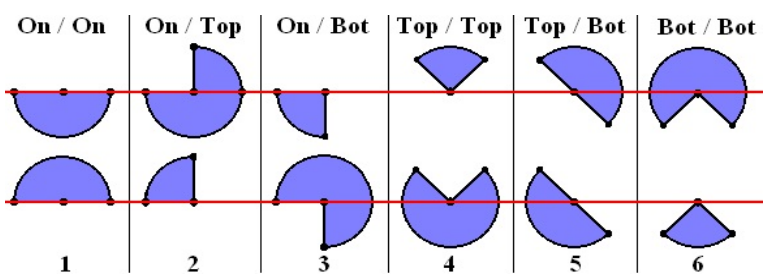


FIGURE 5.22 – *Combinaisons des classes d'arêtes incidentes et voisinages d'un sommet classé «On» pour les deux orientations possibles*

La classification actuelle ne prend pas en compte l'orientation du contour. La figure 5.22 montre les voisinages relatifs aux deux orientations possibles notées o_1 en haut et o_2 en bas. Présentons les types de transitions associés à ces deux éventualités.

cas	$\pi(e_1(v))$	$\pi(e_2(v))$		o_1	o_2
1	*On*	*On*		$On \leftrightarrow On$	$On \leftrightarrow On$
2	*On*	*Top*		$On \leftrightarrow In$	$On \leftrightarrow Out$
3	*On*	*Bottom*		$On \leftrightarrow Out$	$On \leftrightarrow In$
4	*Top*	*Top*		$Out \leftrightarrow Out$	$In \leftrightarrow In$
5	*Top*	*Bottom*		$Out \leftrightarrow In$	$In \leftrightarrow Out$
6	*Bottom*	*Bottom*		$In \leftrightarrow In$	$Out \leftrightarrow Out$

TABLE 5.3 – *Type de transition d'un sommet classé «On» pour les deux orientations éventuelles*

Sur la figure 5.3, les cas 1, 4, 6 ne représentent pas des transitions de classification. Le cas 5 représente bien une transition entre sous-ensembles, de la droite, intérieurs et extérieurs au solide. Les cas 2 et 3 représentent des transitions entre la frontière et l'intérieur ou l'extérieur, ceux-ci doivent donc être interprétés d'après l'hypothèse d'appartenance de la frontière au solide suivant si le solide est considéré ouvert ou fermé.

$$\delta B \not\subset B \Rightarrow \begin{cases} In & \rightarrow & B \\ On & \rightarrow & \overline{B} \\ Out & \rightarrow & \overline{B} \end{cases} \quad et \quad \delta B \subset B \Rightarrow \begin{cases} In & \rightarrow & B \\ On & \rightarrow & B \\ Out & \rightarrow & \overline{B} \end{cases}$$

Exprimons donc les transitions vis-à-vis du solide et de son complément.

cas	o_1	o_2
2	$\overline{B} \leftrightarrow B$	$\overline{B} \leftrightarrow \overline{B}$
3	$\overline{B} \leftrightarrow \overline{B}$	$\overline{B} \leftrightarrow B$

$\delta B \not\subset B$

cas	o_1	o_2
2	$B \leftrightarrow B$	$B \leftrightarrow \overline{B}$
3	$B \leftrightarrow \overline{B}$	$B \leftrightarrow B$

$\delta B \subset B$

TABLE 5.4 – *Transitions de classification entre B et $\overline{B}$*

Dans le cas ouvert, les cas 2 et 3 sont des transitions entre le solide et son complément, pour les orientations respectives o_1 et o_2 et inversement dans le cas fermé.

5.5.4 Classification augmentée des arêtes

Les cas 2 et 3 nécessitant de connaître l'orientation du contour, il nous faut prendre en compte l'orientation d'une des deux arêtes incidentes, l'orientation de l'autre pouvant en être déduite. Nous choisissons pour les cas 2 et 3, d'augmenter la classification des arêtes classées *On*.

$$\pi(e\ on\ A) = \begin{cases} On- & \Leftrightarrow & \overrightarrow{e}.\overrightarrow{d} < 0 \\ On+ & \Leftrightarrow & \overrightarrow{e}.\overrightarrow{d} > 0 \end{cases}$$

Pour définir complètement le voisinage il faut ajouter à l'orientation du contour, la convention sur le coté de la matière. Nous choisissons la convention «matière à gauche». Ainsi les

orientations o_1 et o_2 correspondent respectivement aux classifications $On+$ et $On-$ des arêtes initialement classées On.

5.5.5 Création des sommets

Afin d'éviter les redondances de traitement, nous traitons prioritairement les singularités. Ainsi, nous créons dans le modèle 2D, les sommets issus de sommets, classés On, et représentant des transitions de type $B \leftrightarrow \overline{B}$ puis ceux issus d'intersections de la droite avec l'intérieur d'arêtes classées Cut. Dans ce dernier cas, il suffit de calculer l'intersection de la droite de tranchage avec les droites porteuses de ces arêtes.

5.5.6 Création de la tranche

En 2D, la tranche est l'ensemble des sous-ensembles régularisés et appartenant au solide, de la droite de tranchage (segments). Ces segments sont délimités par les intersections de la droite avec la frontière du solide (points). Ces points se situent soit à l'intérieur d'arêtes classées Cut soit sur des sommets classés On, et représentant des transitions de type $B \leftrightarrow \overline{B}$. Selon l'approche ascendante nous traitons prioritairement les singularités. Les sommets issus de singularités représentant un type de transition $B \leftrightarrow \overline{B}$ sont créés puis les sommets issus d'intersections de la droite et d'arêtes classées Cut. Ces derniers sont calculés par intersection de la droite de tranchage et des droites porteuses des arêtes classées Cut. Enfin, ces intersections sont triées le long de la droite et les sous-ensembles intérieurs au solide créés.

5.6 Tranchage 3D

Il s'agit de calculer les sous-ensembles régularisés et appartenant à B du plan de tranchage A. Ces régions sont délimitées par un ensemble de contours formés de sommets et d'arêtes. Ces contours sont les intersections du plan de tranchage A avec la frontière δB. Pour construire ces contours nous devons identifier de quelles entités, leurs sommets et arêtes sont les intersections.

Les arêtes sont issues d'intersections, avec le plan de tranchage, d'intérieurs de faces classées Cut ou d'arêtes, classées On (singularités) et représentant des transitions de type $B \leftrightarrow \overline{B}$. Alors que le premier cas est trivial, le deuxième nécessite de déterminer le type de transition. Ce problème est équivalent à la résolution des singularités aux sommets en 2D et doit être traité au regard des positions des faces incidentes à l'arête classée On. Nous ne le présentons donc pas.

Les sommets sont issus d'intersections avec le plan de tranchage d'intérieurs d'arêtes classées Cut ou de sommets classés On et représentant des transitions de type $B \leftrightarrow \overline{B}$. Le premier cas est trivial et le deuxième nécessite de déterminer le type de transition, ce qui est relativement plus difficile que dans le cas des singularités au niveau d'arêtes.

De manière analogue au tranchage 2D, notre approche consiste à déterminer les types de transitions en regard de la classification, par rapport au plan de tranchage, des sommets, arêtes et faces de B. Comme pour la 2D, les types de voisinages des sommets seront décrits qualitativement grâce à la classification augmentée des faces.

5.6.1 Classification des sommets

Les sommets sont classés comme étant dans, au dessous, en dessous du plan de tranchage. Définissons la fonction position π d'un sommet $v(x_v, y_v, z_v)$ par rapport au plan de tranchage $\wp(p, \overrightarrow{n})$ défini par un point de passage p et un vecteur normal $\overrightarrow{n}$.

$$\begin{array}{rcl} \pi : \mathbb{R}^3 & \rightarrow & \{On, Top, Bottom, Cut\} \\ v & \rightarrow & \pi(v) \end{array} \qquad \pi(v) = \begin{cases} On & \Leftrightarrow & \overrightarrow{pv}.\overrightarrow{n} = 0 \\ Top & \Leftrightarrow & \overrightarrow{pv}.\overrightarrow{n} > 0 \\ Bottom & \Leftrightarrow & \overrightarrow{pv}.\overrightarrow{n} < 0 \end{cases}$$

Le plan de tranchage étant orienté suivant l'axe $\overrightarrow{z}$ du procédé de prototypage :

$$\pi(v) = \begin{cases} On & \Leftrightarrow & z_v = z_p \\ Top & \Leftrightarrow & z_v > z_p \\ Bottom & \Leftrightarrow & z_v < z_p \end{cases}$$

Cette considération nous permet de minimiser les traitements et d'éviter les arrondis numériques qui seraient intervenus lors de calculs de distance entre sommets et plan.

5.6.2 Classification des arêtes

Rappelons que nous nous attachons au tranchage polyédrique uniquement. Dans ce contexte, la classification d'une arête linéaire peut être déterminée d'après la classification de ses extrémités. De manière analogue à la 2D, la classification des arêtes en 3D est déduite de la classification des sommets.

5.6.3 Classification des faces

De même, la position des faces est déduite des positions de ses sommets. Une face f se situe soit dans, au dessus, au dessous ou encore de part et d'autre du plan de tranchage.

$$\begin{array}{rcl} \pi : \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3 & \rightarrow & \{On,\ Top,\ Bottom,\ Cut\} \\ v & \rightarrow & \pi(v) \end{array}$$

Le nombre de 3-uplets formés des positions des sommets d'une face est $f(3,3) = 10$.

cas	$\pi(v_1)$	$\pi(v_2)$	$\pi(v_3)$	$\pi(f)$
1	*On*	*On*	*On*	*On*
2	*On*	*On*	*Top*	*Top*
3	*On*	*On*	*Bottom*	*Bottom*
4	*On*	*Top*	*Top*	*Top*
5	*On*	*Top*	*Bottom*	*Cut*
6	*On*	*Bottom*	*Bottom*	*Bottom*
7	*Top*	*Top*	*Top*	*Top*
8	*Top*	*Top*	*Bottom*	*Cut*
9	*Top*	*Bottom*	*Bottom*	*Cut*
10	*Bottom*	*Bottom*	*Bottom*	*Bottom*

TABLE 5.5 – *Classification des faces*

Comme pour les arêtes, la classification des faces est augmentée dans les cas de singularités.

$$\pi(f \text{ on } A) = \begin{cases} On- & \Leftrightarrow & \overrightarrow{n_f}.\overrightarrow{n} < 0 \\ On+ & \Leftrightarrow & \overrightarrow{n_f}.\overrightarrow{n} > 0 \end{cases}$$

5.6.4 Transition de classification aux sommets classés «On»

Pour déterminer les types de transition des singularités aux sommets nous devrions, comme en 2D, examiner la position, non pas des arêtes, mais des faces incidentes. Rappelons qu'en 2D, il s'agissait d'établir les combinaisons possibles entre les positions des arêtes incidentes soit des couples de positions. Malheureusement un sommet 3D possède un nombre indéfini de faces incidentes, nous ne pouvons donc énumérer, dans le cas général, les combinaisons (n-uplets) possibles entre leurs positions.

Toutefois si ce type de combinaisons est suffisant pour décider du type de transition, il n'est pas nécessaire. En effet, seule la connaissance des combinaisons entre classes de positions est nécessaire. Nous aurions pu introduire et utiliser cette propriété dès le cas 2D, mais dans ce cas, les arêtes incidentes étant au nombre de deux, il nous semblait plus intuitif d'utiliser les combinaisons de positions plutôt que les combinaisons de classes de positions. Introduisons les combinaisons de classes en reprenant l'exemple du cas de singularité aux sommets en 2D.

cas	$\pi(e_1(v))$	$\pi(e_2(v))$		*On*	*Top*	*Bottom*		o_1	o_2
1	*On*	*On*		1	0	0		$On \leftrightarrow On$	$On \leftrightarrow On$
2	*On*	*Top*		1	1	0		$On \leftrightarrow In$	$On \leftrightarrow Out$
3	*On*	*Bottom*		1	0	1		$On \leftrightarrow Out$	$On \leftrightarrow In$
4	*Top*	*Top*		0	1	0		$Out \leftrightarrow Out$	$In \leftrightarrow In$
5	*Top*	*Bottom*		0	1	1		$Out \leftrightarrow In$	$In \leftrightarrow Out$
6	*Bottom*	*Bottom*		0	0	1		$In \leftrightarrow In$	$Out \leftrightarrow Out$

TABLE 5.6 – *Combinaisons de positions et combinaisons de classes de positions*

Les combinaisons sont formées des classes auxquelles appartient au moins une des faces incidentes. Dans le tableau 5.6 un «1» figure dans les colonnes de ces classes. En 3D, les classes de positions des faces incidentes sont *On*, *Top*, *Bottom* et *Cut*. De plus, si il existe au moins une face classée *Cut*, la transition est toujours de type $B \leftrightarrow \overline{B}$. Nous excluons donc de l'analyse les combinaisons où figure la classe *Cut*.

cas	*On*	*Top*	*Bottom*		o_1	o_2
1	1	0	0		$On \leftrightarrow On$	$On \leftrightarrow On$
2	1	1	0		$On \leftrightarrow In$	$On \leftrightarrow Out$
3	1	0	1		$On \leftrightarrow Out$	$On \leftrightarrow In$
4	0	1	0		$Out \leftrightarrow Out$	$In \leftrightarrow In$
5	0	1	1		$Out \leftrightarrow In$	$In \leftrightarrow Out$
6	0	0	1		$In \leftrightarrow In$	$Out \leftrightarrow Out$
7	1	1	1		$Out \leftrightarrow In$	$In \leftrightarrow Out$

TABLE 5.7 – *Combinaisons des classes de faces incidentes à un sommet classé «On»*

Les cas 1 à 6 sont analogues au cas 2D. En effet, le cas 5 représente une transition de type $B \leftrightarrow \overline{B}$ et les cas 2 et 3 doivent être considérés suivant l'hypothèse d'appartenance de la frontière et en regard de la classification étendue des faces classées *On*. A noter, que dans le cas 2-manifold, la classe *On* ne peut contenir que des faces $On-$ ou $On+$ et non les deux. Les orientations o_1 et o_2 correspondent respectivement aux classifications $On-$ et $On+$ des faces initialement classée *On*.

En 3D, un septième cas est possible puisque, contrairement à la 2D où les deux arêtes incidentes ne peuvent appartenir, simultanément, à plus de deux classes, les faces incidentes peuvent appartenir aux trois classes. Ce cas représente une transition de type $B \leftrightarrow \overline{B}$.

5.6.5 Création des sommets

Dans le cadre de l'approche ascendante, pour éviter les redondances de traitement, nous traitons prioritairement les singularités. Nous créons donc, dans le modèle 2D les sommets issus de sommets, classés *On* et représentant des transitions de type $B \leftrightarrow \overline{B}$ puis ceux issus d'intersections du plan avec l'intérieur d'arêtes classées *Cut*. Dans ce dernier cas, il suffit de calculer l'intersection du plan de tranchage avec les droites porteuses des arêtes classées *Cut*.

Dans les deux cas nous gardons un lien entre l'entité coupée et le sommet intersection. Ainsi, il est possible d'obtenir les sommets intersections associés aux arêtes classées Cut et aux sommets, classés On et représentant des transitions de type $B \leftrightarrow \overline{B}$.

5.6.6 Création des arêtes

De la même manière, nous traitons prioritairement les singularités. Ainsi, nous créons dans le modèle 2D, les arêtes issues d'arêtes, classés On et représentant des transitions de type $B \leftrightarrow \overline{B}$ puis celles issues d'intersections du plan avec l'intérieur de faces classées Cut.

Dans le premier cas, comme les arêtes classées On représentent des transitions de type $B \leftrightarrow \overline{B}$, ses extrémités en représentent également et sont donc associées à des sommets intersections précédemment créés. Nous créons donc l'arête reliant ces sommets. Dans le deuxième cas, les faces classées Cut possèdent soit deux arêtes classées Cut, soit une arête classée Cut et un sommet classé On et représentant une transition de type $B \leftrightarrow \overline{B}$. Deux sommets intersections précédemment créés, sont donc associés à chacune de ces faces. Nous créons donc l'arête reliant ces sommets intersections.

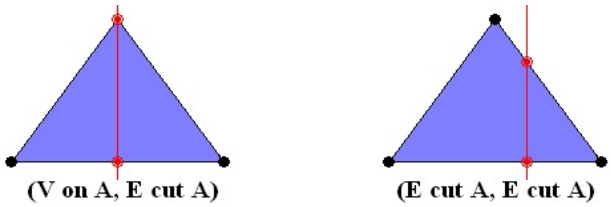


FIGURE 5.23 – *Création des arêtes issues de faces classées «Cut»*

C'est sur ce point que se différencie l'approche ascendante de l'approche descendante. En effet, dans l'approche descendante, l'intersection du plan et de la frontière aurait été calculée comme une succession d'intersections de faces et l'intersection de chaque face comme une succession d'intersections d'arêtes. Ainsi, les sommets issus d'intérieurs d'arêtes et les arêtes issues de singularités au niveau d'arêtes auraient été calculés deux fois et les sommets issus de singularités au niveau des sommets auraient été calculés un nombre indéfini de fois. Ce type d'approche a l'inconvénient de la redondance des traitements et nécessite un post-traitement consistant à fusionner les résultats redondants.

Dans l'approche ascendante, les constituants de l'intersection du plan et de la frontière sont calculés en ordre croissant de leur dimension, sommets et arêtes. Ces sommets et arêtes sont respectivement issus d'intersections d'entités de dimensions différentes. Les sommets sont issus d'intersections d'intérieurs d'arêtes et de sommets représentant des transitions de type $B \leftrightarrow \overline{B}$. Les arêtes sont issues d'intersections d'intérieurs de faces et d'arêtes représentant des transitions de type $B \leftrightarrow \overline{B}$. Afin d'éviter le recouvrement des traitements, les sommets et arêtes sont respectivement créés dans l'ordre croissant des dimensions des entités dont elles sont les intersections. Les sommets issus de singularités au niveau des sommets sont créés avant ceux issus d'intersections d'intérieurs d'arêtes. Les arêtes issues de singularités au niveau d'arêtes sont créées avant celles issues d'intersections d'intérieurs de faces. En prenant soin de conserver une association entre entités coupées et entités intersections, cette approche permet de créer des

arêtes partageant naturellement leurs sommets avec leurs homologues adjacentes. La topologie du contour étant naturellement exprimée, le post-traitement de fusion précédemment nécessaire, devient inutile.

5.6.7 Création de la tranche

Dans le cas du tranchage, on appelle «tranche» un ensemble de régions pouvant être trouées et non connexes. Ces régions sont délimitées par des contours d'arêtes connectées. La création de la tranche consiste à identifier et à regrouper les arêtes connectées afin de construire l'ensemble des contours. Dans notre approche les arêtes sont naturellement connectées dès leur création. Ensuite, nous établissons leurs niveaux d'inclusion pour identifier les contours intérieurs des contours extérieurs. Ceux-ci sont enfin réorientés et regroupés pour construire les faces (sous-ensembles connexes pouvant être troués). Finalement la tranche (ensemble non connexe de faces) est créée.

5.7 Implémentation

Nous avons présenté notre approche ascendante consistant à construire les sommets et arêtes de la tranche dans l'ordre croissant de leur dimension et de la dimension des entités dont ils sont les intersections. Cette approche supprime les redondances de traitement et le post-traitement de fusion.

De plus, la classification est également ascendante puisque la classification des arêtes et des faces est déduite de la classification des sommets. A noter que ceci n'est valable que pour les géométries polyédriques. En effet, les extrémités d'une courbe, coupant le plan de tranchage, ne sont pas nécessairement situées de part et d'autre de ce dernier. Nous présentons une astuce d'implémentation permettant de calculer efficacement la classification.

5.7.1 Arithmétisation de la classification

L'arithmétisation A de la classification consiste à associer à chaque position π une valeur numérique entière $A(\pi)$. Ainsi, les relations existant entre les positions des arêtes, des faces et celles de leurs sommets peuvent être projetées en relations arithmétiques sur les entiers. Plusieurs formes d'arithmétisation étant possibles, les relations dépendent de la forme choisie.

$$\begin{array}{rcl} \{On,\ Top,\ Bottom,\ Cut\} & \rightarrow & \mathbb{N} \\ \pi & \rightarrow & A(\pi) \end{array}$$

π	$A(\pi)$	$A(\pi)$
On	0	00
Top	1	01
$Bottom$	2	10
Cut	3	11

TABLE 5.8 – *Arithmétisation de la classification*

L'arithmétisation choisie peut être interprétée d'après la représentation binaire de l'entier associé. Les bits de poids faible et fort représentent respectivement l'appartenance au demi-espace supérieur et inférieur d'au moins un point du sous-ensemble en question.

En substituant, dans la classification des arêtes (cf. tableau 5.2) les différentes positions par leurs équivalents entiers nous obtenons la relation suivante.

$$A(\pi(e(v_1, v_2))) = A(\pi(v_1)) \vee A(\pi(v_2))$$

La classification des arêtes est obtenue d'après la classification de ses extrémités.

$$\pi(e(v_1, v_2)) = A^{-1}(A(\pi(v_1)) \vee A(\pi(v_2)))$$

De la même manière, en substituant, dans la classification des faces (cf. tableau 5.5), les différentes positions par leurs équivalents entiers nous obtenons la relation suivante.

$$A(\pi(f(v_1, v_2, v_3))) = A(\pi(v_1)) \vee A(\pi(v_2) \vee A(\pi(v_3))$$

La classification des faces est obtenue d'après la classification de ses sommets.

$$\pi(f(v_1, v_2, v_3)) = A^{-1}(A(\pi(v_1)) \vee A(\pi(v_2) \vee A(\pi(v_3)))$$

5.7.2 Optimisation

Il est naturel de concevoir l'algorithme de tranchage comme une répétition de calculs de tranches. Cependant nous montrons qu'il est plus judicieux de considérer le problème dans sa globalité car il est possible et avantageux de réutiliser le calcul d'une tranche pour le calcul de la suivante. En effet, la classification nous permet d'isoler les informations juste nécessaires au calcul d'une tranche. Dans l'approche itérative, nous devrions, pour chaque nouvelle tranche, répéter les phases de classification et de création. Or il s'avère que la classification précédente ne soit pas tout à fait obsolète et puisse être réutilisée.

Evolution de la classification

En prenant la convention de trancher de bas en haut, il existe des relations entre les positions antérieures, présentes et futures des différentes entités.

sommets			arêtes et faces		
On	$\rightarrow$	$Bottom$	On	$\rightarrow$	$Bottom$
Top	$\rightarrow$	$On\|Top\|Bottom$	Top	$\rightarrow$	$On\|Top\|Bottom\|Cut$
$Bottom$	$\rightarrow$	$Bottom$	$Bottom$	$\rightarrow$	$Bottom$
			$Bottom$	$\rightarrow$	$Bottom\|Cut$

FIGURE 5.24 – *Evolution des positions des sommets, arêtes et faces*

Informations nécessaires à une approche globale

Le calcul d'une tranche nécessite de connaître l'ensemble des sommets On, des arêtes On et Cut et des faces Cut. Dans l'approche globale il faut également conserver les entités susceptibles de devenir intéressantes dans les itérations à venir. Soit les sommets susceptibles de devenir On, les arêtes de devenir On ou Cut et les faces de devenir Cut. Il faut donc conserver les sommets Top, arêtes et faces Top et Cut.

Initialement toutes les entités sont classées Top puis la classification est mise à jour pour le calcul des tranches suivantes. Les entités classées $Bottom$ étant inutiles au calcul des tranches suivantes, celles-ci sont supprimées. Ainsi la quantité d'informations à traiter diminue durant l'algorithme, le rendant de plus en plus rapide.

	On	*Top*	*Bottom*	*Cut*
v	$\surd$	$\surd$	$\times$	
e	$\surd$	$\surd$	$\times$	$\surd$
f	$\times$	$\surd$	$\times$	$\surd$

TABLE 5.9 – *Informations nécessaires à une approche globale*

5.8 Conclusion

Nous avons proposé une approche unifiée des algorithmes de tranchage appliqués aux solides polyédriques 2-manifold 2D et 3D. En 3D, il s'agit de solides à faces triangulaires. Dans ce cadre, nous avons développé une approche ascendante éliminant les redondances de traitement et le post-traitement de fusion associés à l'approche descendante. De plus, la classification des différentes entités est déduite de la classification des sommets uniquement. Ces algorithmes ont été implémentés en Java au sein d'un composant logiciel.

5.9 Illustrations

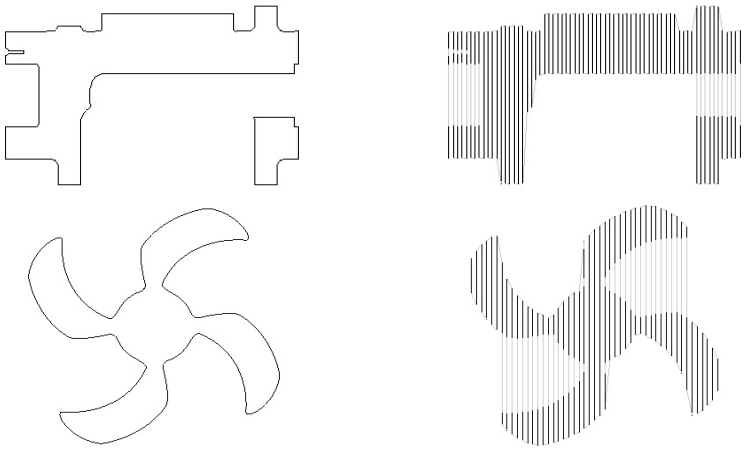


FIGURE 5.25 – *Illustrations du tranchage 2D*

Le résultat du tranchage est utilisé pour générer la trajectoire de balayage du laser. Les parties noires et les parties grisées de la trajectoire représentent respectivement, les déplacements du laser allumé et éteint.

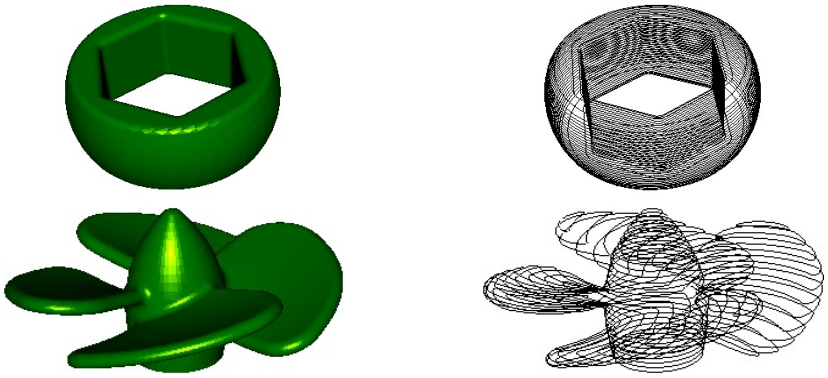


FIGURE 5.26 – *Illustrations du tranchage 3D*

5.10 Maquette logicielle

Dans le but de la valider, l'architecture présentée fut implémentée au sein d'une maquette logicielle. Nous illustrons à présent, l'implémentation de services de plus haut niveau. Comme nous l'avons évoqué au chapitre IV, la direction de construction conditionne l'état de surface et le temps de fabrication de la pièce prototype. Nous présentons dans ce paragraphe l'implémentation des services de simulation de l'effet d'escalier et de calcul du coût de fabrication. Le service de simulation de l'effet d'escalier sera illustré au sein d'un module de positionnement permettant de choisir une direction de construction minimisant la rugosité de la pièce prototype. Puis, une fois la direction de construction choisie, le calcul des tranches et des trajectoires de remplissage permettra de déterminer, en fonction des caractéristiques du procédé, le temps de fabrication. Les paramètres, tel que le matériau utilisé, la consommation énergétique ou encore les coûts de main d' oeuvre, permettront d'établir le coût de fabrication. Enfin, nous présentons une application client serveur, accessible via Internet, proposant les services de préparation des données et de devis automatique.

5.10.1 Modélisation de l'effet d'escalier

L'état de surface du prototype dépend de la direction de stratification. Nous proposons de modéliser l'effet d'escalier résultant du processus de stratification. Représenté graphiquement celui-ci permet à l'utilisateur de visualiser l'influence de la direction de construction sur l'état de surface du prototype.

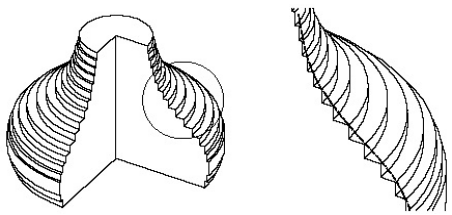
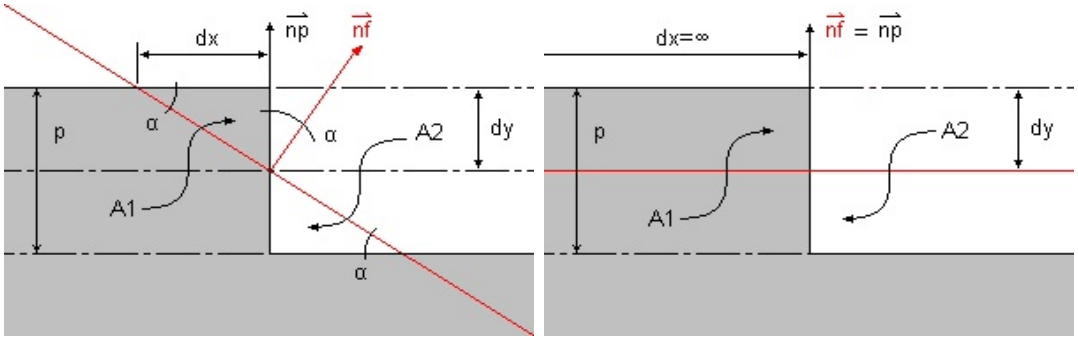


FIGURE 5.27 – *L'effet d'escalier*

Quantifions l'écart entre la surface idéale et la surface stratifiée de l'objet.

Ecart moyen

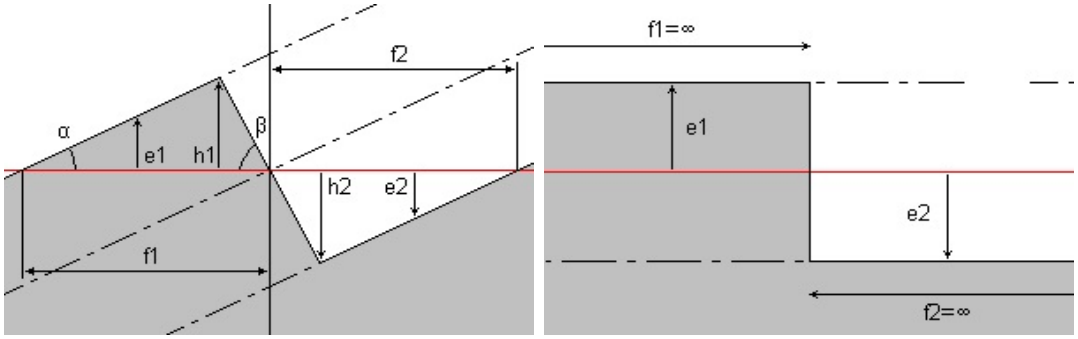
Nous proposons de déterminer l'écart dimensionnel moyen entre la géométrie idéale et la géométrie fabriquée. Nous verrons que celui-ci dépend de deux paramètres uniquement, le pas de stratification p et l'orientation de la surface idéale par rapport à la direction de construction. Nous verrons donc qu'il faut distinguer deux cas, le cas pour lequel la distribution de l'écart est triangulaire ($\alpha \in]0, \pi/2]$) et celui pour lequel la distribution est rectangulaire ($\alpha = 0$).



(a) Triangulaire (b) Rectangulaire

FIGURE 5.28 – *Répartition de l'écart entre la surface idéale et la surface stratifiée*

Sur la figure 5.28, la ligne diagonale représente une face plane de la surface idéale, sa normale $\overrightarrow{n_f}$ et la normale au plan de tranchage ou direction de construction $\overrightarrow{n_p}$. La zone grise représente la géométrie fabriquée, A_1 et A_2 dénote respectivement les aires des parties fabriquées supérieures et inférieures à la surface idéale. Pour la répartition triangulaire d_x et d_y paramètrent les cotés, adjacents et opposés à l'angle α. Pour la répartition rectangulaire d_y est la hauteur du rectangle. Donnons à présent le paramétrage complet du problème.



(a) Triangulaire (b) Rectangulaire

FIGURE 5.29 – *Expression des écarts locaux pour les deux répartitions*

Notons e_1 et e_2 respectivement les écarts locaux entre surface idéale et surfaces fabriquées supérieure et inférieure. Nous définissons l'écart moyen $\overline{e}$ comme la somme des valeurs absolues des écarts moyens supérieurs $\overline{e_1}$ et inférieurs $\overline{e_2}$.

$$\overline{e} = |\overline{e_1}| + |\overline{e_2}| \;\; avec \begin{cases} \overline{e_1} = \frac{1}{f_1}\int_0^{f_1} e_1(f)df = \frac{A_1}{f_1} \\ \overline{e_2} = \frac{1}{f_2}\int_0^{f_2} e_2(f)df = \frac{A_2}{f_2} \end{cases} \quad et \quad \begin{cases} f_1 = f_2 \\ e_1(f) = -e_2(f) \end{cases}$$

$$donc \;\; \overline{e} = \frac{|A_1|}{f_1} + \frac{|A_2|}{f_2}$$

Dans le cas de la répartition rectangulaire $|\overline{e_1}| = |\overline{e_2}| = \frac{p}{2}$ $donc$ $\overline{e} = p$.

Dans le cas de la répartition triangulaire

$$\begin{cases} \sin\alpha = \frac{d_y}{f_1} = \frac{d_y}{f_2} \\ \tan\alpha = \frac{d_y}{d_x} \end{cases} \quad donc \quad \begin{cases} \frac{1}{f_1} = \frac{1}{f_2} = \frac{\sin\alpha}{d_y} & (1) \\ d_x = \frac{d_y}{\tan\alpha} & (2) \end{cases} \quad et \;\; |A_1| = |A_2| = \frac{d_x d_y}{2} \; (3)$$

En substituant dans (3) d_x par (2) nous obtenons $|A_1| = |A_2| = \frac{d_y^2 \cos\alpha}{2\sin\alpha}$ (4)

D'après (1) et (4) $\frac{|A_1|}{f_1} = \frac{|A_2|}{f_2} = \frac{d_y}{2}\cos\alpha$ $donc$ $\overline{e} = d_y \cos\alpha = \frac{p}{2}\cos\alpha$

$$Finalement \; \overline{e} = \begin{cases} p & pour \quad \alpha = 0 \\ \frac{p}{2}\cos\alpha & pour \quad \alpha \in]0, \pi/2] \end{cases}$$

Le principal défaut de l'écart moyen pour représenter l'effet d'escalier est sa discontinuité en $\alpha = 0$. En effet quand α tend vers 0, $\overline{e}$ tend vers $p/2$ et non vers p.

Ecart différentiel

L'écart différentiel est la différence entre l'écart maximum et l'écart minimum. Dans le cas général, il n'est pas très représentatif car il ne prend en compte que l'amplitude et pas la forme. En effet, deux surfaces de même écart différentiel n'ont pas forcément le même écart moyen. En général, l'écart moyen est plus représentatif, cependant dans notre contexte notre surface fabriquée possède toujours la même allure, celle d'un escalier, celle-ci étant pleinement définie par son amplitude, l'écart différentiel s'avère suffisant. Si, toutefois, nous voulons connaître l'écart moyen, nous pourrons le faire à partir de l'écart différentiel.

$$e_{diff} = e_{max} - e_{min} \;\; donc \;\; e_{diff} = e_1 - e_2$$

Dans le cas de la répartition rectangulaire $e_1 = -e_2 = p/2$ $donc$ $e_{diff} = p$.

Pour la répartition triangulaire $e_1 = -e_2 = h_1 = -h_2 = \frac{p}{2}\cos\alpha$ $donc$ $e_{diff} = p\cos\alpha$.

Finalement on s'aperçoit que l'écart différentiel est continu puisque pour la forme triangulaire quand α tend vers 0, e_{diff} tend vers p, sa valeur pour la forme rectangulaire. Nous pouvons donc étendre la formule de la forme triangulaire au domaine entier.

$$e_{diff} = p\cos\alpha \;\; pour \;\; p \in [0, \pi/2]$$

De l'écart différentiel à l'écart moyen

Comme nous l'avons évoqué l'écart différentiel est suffisant pour caractériser l'effet d'escalier. Aussi, sa continuité en facilite l'implémentation informatique. Il peut, néanmoins, être intéressant de connaître l'écart moyen, celui-ci pouvant être déduit de l'écart différentiel par la relation suivante.

$$\overline{e} = \begin{cases} e_{diff} & pour \quad \alpha = 0 \\ 2e_{diff} & pour \quad \alpha \in]0, \pi/2] \end{cases}$$

Représentation graphique

L'écart différentiel est au minimum nul et au maximum égal au pas de stratification. Afin que celui-ci puisse être représenté graphiquement, de manière qualitative, c'est-à-dire sans que l'utilisateur ait à spécifier le pas de stratification, nous procédons à sa normalisation.

$$e_{diffnormal} = e_{diff}/p \;\; comme \;\; 0 \leq e_{diff} \leq p \;\; donc \;\; 0 \leq e_{diffnormal} \leq 1$$

Pour la représentation graphique nous définissons la fonction de coloration faisant correspondre à l'intervalle $[0, 1]$, une couleur allant du bleu vers le rouge en passant par le vert. Pour cela, nous définissons pour chacune des composantes RGB, une fonction d'intensité associant à une valeur de l'intervalle $[0, 1]$ une intensité comprise entre l'intensité minimum et maximum du contexte graphique, par exemple $[0, 1]$. Les fonctions d'intensité RGB sont discontinues et définies par intervalles.

$$\begin{aligned} c : [0,1] &\rightarrow [0,1]^3 \\ \varepsilon &\rightarrow c(\varepsilon) = (r(\varepsilon),\ g(\varepsilon),\ b(\varepsilon)) \end{aligned}$$

	$[0, 1/4]$	$[1/4, 1/2]$	$[1/2, 3/4]$	$[3/4, 1]$
$r(\varepsilon)$	0	0	$4\varepsilon - 2$	1
$g(\varepsilon)$	ε	1	1	$-4\varepsilon + 4$
$b(\varepsilon)$	1	$-4\varepsilon + 2$	0	0

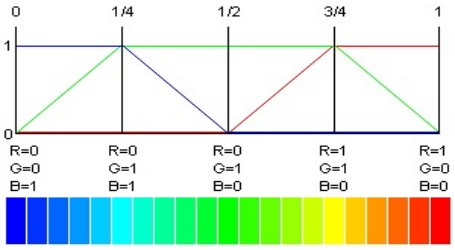


FIGURE 5.30 – *Fonctions d'intensité RGB*

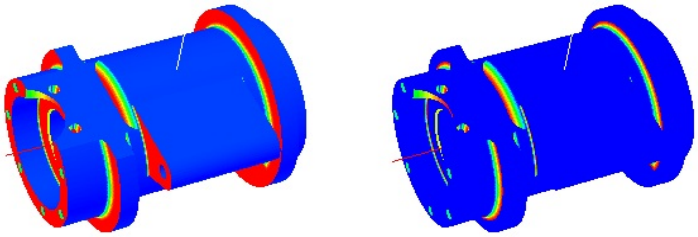
Nous obtenons une transition continue du bleue vers le rouge en passant par le vert.

FIGURE 5.31 – *Représentation graphique de l'effet d'escalier*

Rugosité

La fonction de rugosité est une variante du défaut de forme, elle est nulle pour $\alpha = 0$ (surfaces perpendiculaires à la direction de construction), ce qui en fait une fonction discontinue.

Nous avons défini et quantifié le défaut de forme et la rugosité de l'effet d'escalier. L'étude quantitative fait intervenir la direction de construction et le pas de stratification alors que l'étude qualitative ne fait intervenir que la direction de construction.

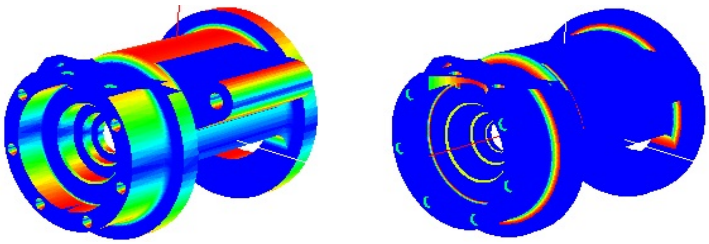


(a) Défaut de forme (b) Rugosité

FIGURE 5.32 – *Défaut de forme et rugosité d'une forme de révolution construite axialement*

L'effet d'escalier est en général mineur pour des procédés autorisant un faible pas de stratification (quelques centièmes de millimètres). Il est par contre non négligeable pour les procédés dont le pas vaut entre quelques centièmes et quelques millimètres. C'est le cas des procédés de strato conception.

Ainsi, une géométrie donnée, construite suivant une direction donnée, possède une rugosité plus ou moins importante. Nous pouvons donc démontrer des règles empiriques couramment utilisées pour positionner certaines formes. Ainsi un cylindre possède un défaut de forme et une rugosité nuls quand il est construit suivant son axe de révolution. Plus généralement, les formes de révolution construites suivant leur axe possèdent un défaut de forme uniforme.



(a) Radiale (b) Axiale

FIGURE 5.33 – *Rugosité d'une forme de révolution construite suivant une direction radiale et axiale*

Construite suivant son axe de révolution, une pièce de révolution possédera une rugosité uniformément répartie. Si celle-ci est cylindrique, l'effet d'escalier est nul.

5.10.2 Estimation du coût de fabrication

Hormis les facteurs financiers tel que l'amortissement du procédé, le coût de fabrication dépend essentiellement des quantités et des coûts des consommables utilisés durant la fabrication. Le temps de fabrication est donc un paramètre déterminant.

Le processus de fabrication est composé des phases de construction, de pré- et post-traitement. Nous nous attachons à évaluer le temps, et donc le coût, de la phase de construction d'une géométrie donnée.

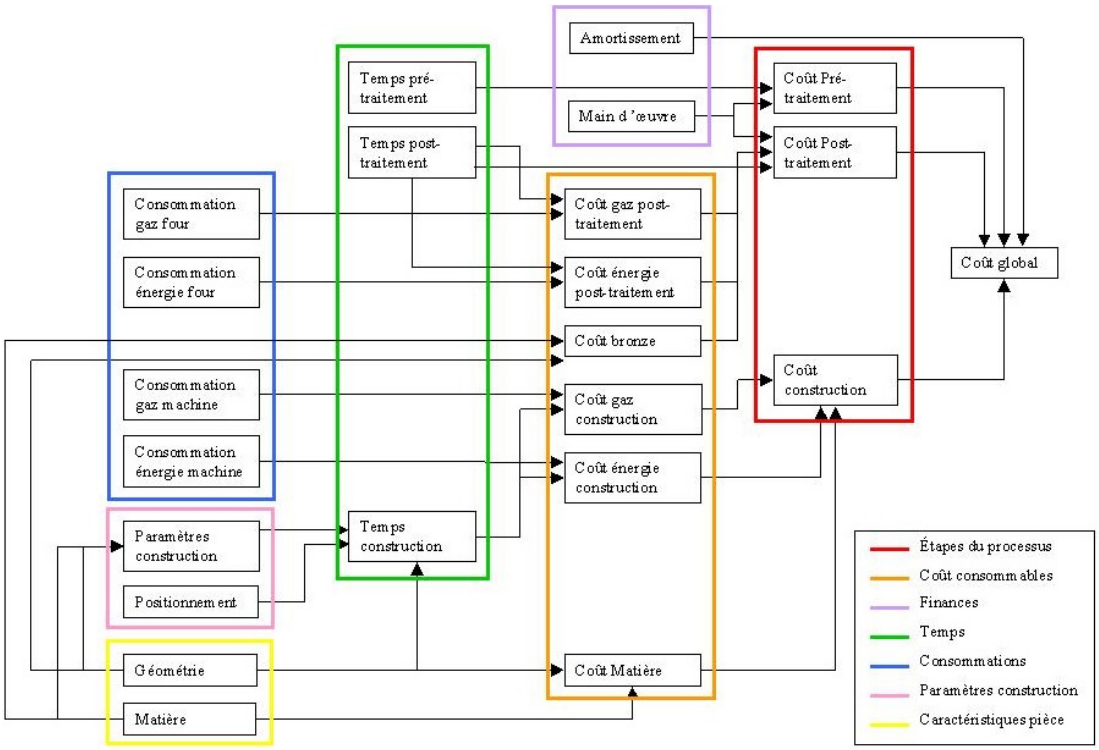


FIGURE 5.34 – *Paramètres influants sur le coût de fabrication pour le procédé de frittage de poudre*

Le temps de construction d'un objet donné peut être estimé, en connaissance des trajectoires et de la vitesse de l'outil. Celui-ci dépend donc de la direction de construction et des pas de stratification et de remplissage. En intégrant le coût du matériau et les coûts de fonctionnement, nous obtenons une estimation du coût de construction.

5.11 Conclusion

Nous avons présenté une application de simulation du prototypage rapide, validant l'architecture de composants logiciels spécifiée au chapitre IV. Accessible via Internet, celle-ci permet à un client, de préparer à distance, la fabrication d'une pièce prototype. L'utilisation des services CAO assurerait l'interopérabilité avec différents systèmes de CAO. Ainsi, un système de prototypage pourrait disposer d'informations et de services du système de CAO et inversement, la publication des services de prototypage permettrait de préparer la fabrication depuis un système de CAO. L'intégration avec un système de gestion de données technique améliorerait l'archivage et la gestion des dossiers clients. Du coté du procédé, le développement de composants de pilotage spécifiques permettrait d'interfacer divers types de procédés. De manière générale, le développement d'application d'IAO sous forme de composants logiciels assurerait une meilleure collaboration des services tout au long du processus de développement.

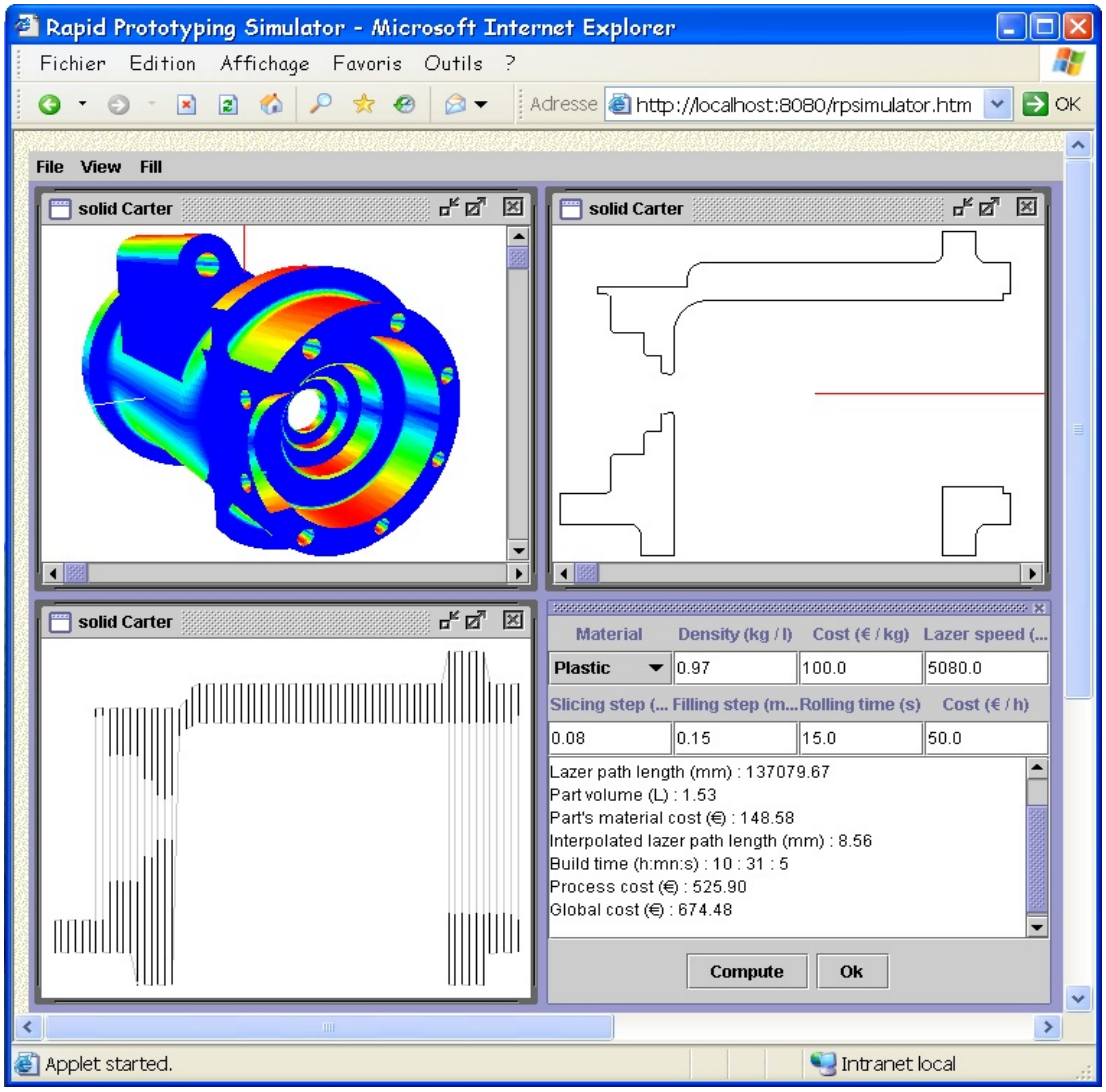


FIGURE 5.35 – *Simulation de la rugosité, du tranchage, du remplissage et estimation de coût d'un carter construit suivant une direction radiale*

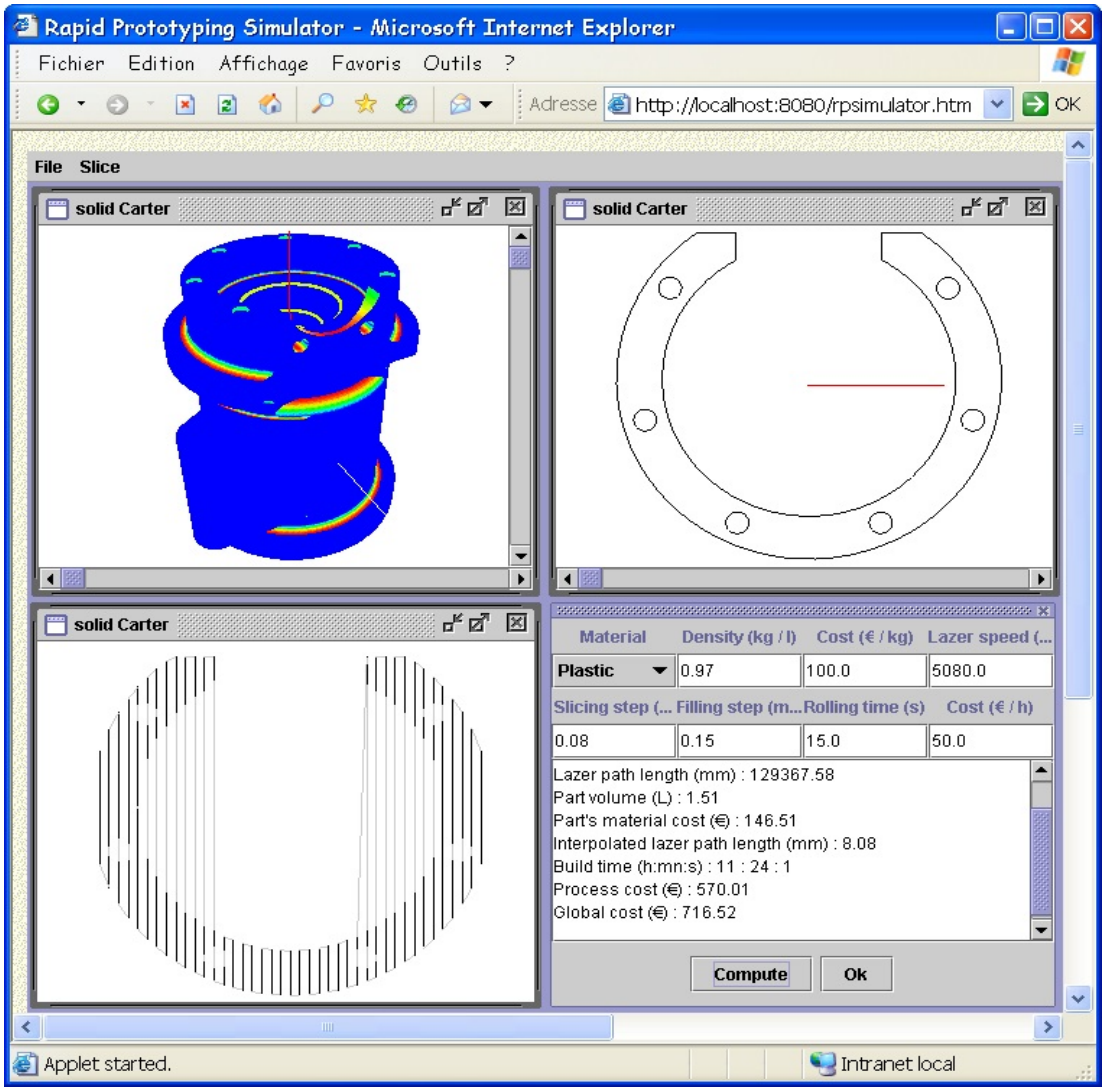


FIGURE 5.36 – *Simulation de la rugosité, du tranchage, du remplissage et estimation de coût d'un carter construit suivant une direction axiale*

Conclusion et perspectives

Face aux problèmes de communication des systèmes d'ingénierie, cette thèse montre la nécessité des consensus dans l'établissement des normes. Malheureusement ces derniers nécessitent la mobilisation d'un grand nombre d'acteurs rendant leur développement long et coûteux. De plus, l'évolution rapide des technologies de l'information ne facilite pas le processus de normalisation. Il n'est en effet pas rare qu'une norme se révèle obsolète dès sa publication. Ce constat met en péril la norme STEP, prise de vitesse par les standards de médiateur, ce qui est à notre avis un mal pour un bien car ces derniers semblent plus probants.

En effet, l'approche STEP est par essence focalisée sur la normalisation des représentations dans les domaines de l'ingénierie. L'approche médiateur fournit le cadre permettant aux applications et aux objets, de quelque nature qu'ils soient, d'outrepasser la barrière des processus, plateformes et réseaux en communicant au travers d'interfaces bien définies. Chaque corps de métier devrait, dans ce cadre, se concentrer à la spécification de leurs composants et interfaces.

Les chapitres I et II ont pour objectif de sensibiliser le lecteur aux aspects liés à la normalisation, aux évolutions technologiques mais surtout de présenter précisément les différences fondamentales des deux approches. L'approche STEP de normalisation des représentations des domaines de l'ingénierie et l'approche médiateur qui fournit les bases générales de l'interopérabilité par l'abstraction des représentations et la normalisation des services.

Après que le prototypage rapide ait été présenté au chapitre III, le chapitre IV propose d'appliquer cette approche à ce domaine. Nous montrons tout d'abord les faiblesses du standard de fait qu'est STL, en développant, à titre d'exercice, un format plus compact et supportant la topologie. Nous proposons ensuite de réviser les échanges en amont et en aval des systèmes de prototypage (avec les systèmes de CAO et les procédés de prototypage) non pas comme un transfert intégral mais comme des échanges de services. Nous proposons de spécifier un ensemble de composants logiciels permettant d'assurer une chaîne numérique offrant des services à plusieurs niveaux (3D, 2D polyédriques et exactes).

Le chapitre V présente, sur les bases théoriques des algorithmes d'opérations booléennes, une formulation unifiée (2D, 3D) des algorithmes de tranchage et de remplissage. Ces derniers étant analogues à des intersections booléennes, les cas d'ambiguïté On/On nécessitent l'étude des voisinages des opérandes en vue de leur intersection. Cependant, contrairement à l'intersection booléenne, le tranchage et le remplissage font intervenir des opérandes de dimensions différentes. Cette propriété permet de formuler le problème d'intersection des voisinages comme un cas particulier du problème général difficile à résoudre dans les cas d'ambiguïtés aux sommets.

Nous nous sommes également attaché à éliminer les redondances des traitements inhérentes à l'approche descendante ainsi que le post-traitement qui en découle. Nous présentons également une optimisation basée sur l'approche globale du tranchage. Finalement nous proposons des algorithmes de tranchage et de remplissage fiables et minimaux puisque traitant les cas

de singularités sans redondances. Nous pensons cependant que de nombreuses optimisations peuvent encore être élaborées, particulièrement par l'utilisation de tables de hachage pour le filtrage des informations pertinentes au tranchage d'une région particulière de l'objet.

Finalement nous présentons une maquette applicative accessible via Internet permettant la préparation des données de prototypage (placement, tranchage, remplissage) ainsi que la simulation de la fabrication (simulation de l'effet d'escalier, estimation du temps et du coût de fabrication).

Applications et perspectives de l'interopérabilité en IAO

Après avoir présenté l'approche informatique de la médiation, donnons les perspectives de celle-ci dans le domaine de l'IAO. Tout d'abord rappelons que l'IAO est un domaine «à cheval» entre l'informatique et l'ingénierie. Même si l'informatique en tant qu'outil est au service de l'ingénierie, les problématiques en IAO doivent concilier les contraintes de ces deux disciplines. Malheureusement les différences culturelles ne facilitent pas les approches pluridisciplinaires et il est souvent difficile de sensibiliser une communauté sur des aspects ne relevant pas de son domaine de compétence. C'est pourquoi après avoir exposé de manière détaillée les approches informatiques de l'interopérabilité nous en donnons les applications concrètes.

Exploitation de modèles hétérogènes

Dans un contexte de collaboration industrielle, l'approche de la médiation permettrait aux donneurs d'ordre et sous traitants, utilisant des systèmes hétérogènes, d'exploiter les mêmes modèles du produit outrepassant ainsi les différences de représentation. Il serait possible depuis un système X de visualiser, consulter voir modifier un modèle issu d'un système Y. Le modèle du produit pourrait être un assemblage de composants issus de divers systèmes. La médiation est dans ce cas indispensable à la gestion de relations inter modèles et inter applications.

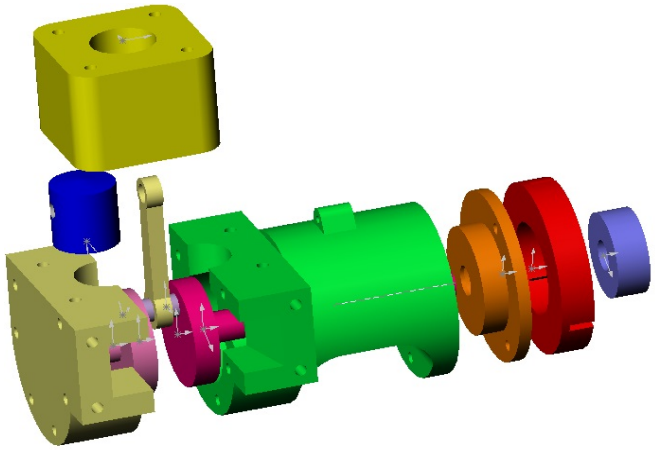


FIGURE 5.37 – *Conception par assemblage de modèles hétérogènes*

Sur la figure 5.37 chaque composant peut avoir été conçu sur des systèmes différents, leur assemblage nécessite donc des relations inter modèles voire inter applications.

Un logiciel de simulation pourrait effectuer un calcul de structure sur des modèles de produit indépendamment de leur représentation. Les systèmes de GDT pourraient gérer des informations, provenant de systèmes hétérogènes, et d'une granularité plus fine qu'actuellement (celle

du fichier). Ceci permettrait d'établir et de maintenir des relations complexes entre composants du produit tout en permettant leur hétérogénéité. Ces caractéristiques sont à notre avis indispensables à la gestion de données en environnement collaboratif et résultent en ce que nous appelons le modèle composite du produit.

Dans le domaine du prototypage rapide, les perspectives sont la délocalisation de la préparation des données qui s'effectue aujourd'hui sur le procédé de prototypage et la facilitation de l'usage de ce procédé. Les procédés de prototypage qui, par ailleurs très utiles en développement de produits, se révèlent relativement onéreux et malheureusement peu répandus. Offrir leur accès à des utilisateurs distants permettrait d'en augmenter l'utilisation et la rentabilité. La préparation des données pourrait être distribuée chez les clients, plutôt que centralisée sur le procédé tel que cela a lieu actuellement.

Modèle composite du produit

Dans l'optique de la collaboration de systèmes hétérogènes, le produit est représenté par un ensemble de documents électroniques métiers issus de divers systèmes. Les entités qu'ils contiennent doivent pouvoir être contraintes ou mises en relation. Nous désignons comme composite un tel modèle.

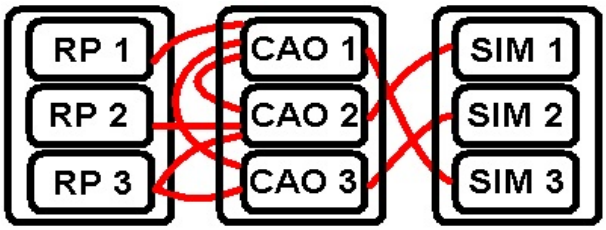


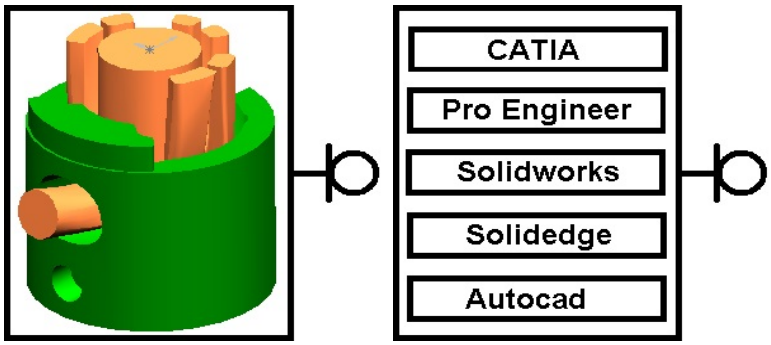
FIGURE 5.38 – *Modèle composite du produit*

La figure 5.38 représente un modèle composite du produit comprenant les documents relatifs aux domaines de la CAO, prototypage (RP) et à la simulation (SIM). Ces derniers peuvent avoir été créés par différents systèmes (notés 1, 2, 3). Ce modèle permettrait de naviguer entre les différentes caractéristiques métiers du produit mais aussi de faciliter les mises à jour et de tendre vers une meilleure cohérence. Ceci dit un tel modèle n'est possible que si ces composants possèdent des interfaces claires et normalisées. Dans ce sens cette thèse apporte notre contribution, tout du moins méthodologique, à la spécification des composants relatifs au domaine du prototypage rapide.

Interaction applicative et automatisation

Si le modèle composite fournit les vues relatives à divers métiers, les processus de l'entreprise régissent les dépendances des documents associés (création, actualisation, destruction, etc.). Dans ce cadre d'ailleurs le SGDT aide à mettre en place la politique de gestion. Il serait intéressant de permettre d'automatiser, autant que possible, certaines tâches ne nécessitant pas l'intervention humaine. Par exemple, suite à une évolution des documents de conception, les documents de fabrication pourraient être régénérés automatiquement. Pour les tâches ne pouvant être automatisées (soumission, validation, etc.), un processus consistant à avertir les responsables concernés pourrait être lancé.

Cette automatisation faisant intervenir les fonctionnalités des systèmes nous voyons là, en plus de la normalisation des services des objets métiers (modèle produit), la nécessité de la normalisation des fonctionnalités mêmes de ces systèmes.



(a) Modèle solide (b) Systèmes de CAO

FIGURE 5.39 – *Exemple d'interfaces du modèle solide et des systèmes de CAO*

Les interfaces des modèles métiers doivent être normalisées ainsi que celles des systèmes qui les manipulent. La figure 5.39(a) représente un solide et ses interfaces indépendantes du modeleur. La figure 5.39(b) représente les services de CAO indépendants des modeleurs. Les composants «STLTopModel» et «STLTopSlicer» sont leurs équivalents dans le domaine du prototypage rapide.

La migration

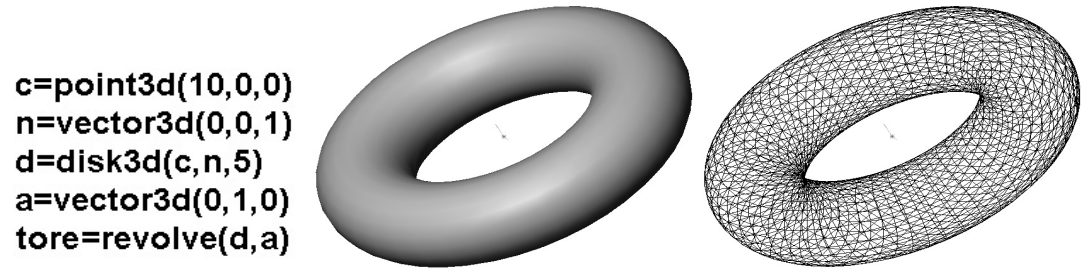
De manière générale, la médiation fournirait l'abstraction matérielle et logicielle nécessaire à une meilleur collaboration logicielle et humaine. Dans ce contexte, l'utilisateur ne devrait pas se soucier de savoir quel système héberge le modèle sur lequel il travaille. Néanmoins pour des questions évidentes de performances et de politique de gestion des privilèges, l'identification du système hôte est non seulement nécessaire mais la capacité de migration d'un modèle d'un système à un autre serait également fort utile. Même si la migration n'est pas une caractéristique fondamentale de la médiation, son approche fonctionnelle nous laisse entrevoir des solutions dans ce domaine.

Vision fonctionnelle de la conception

De manière général, la vision médiateur de l'interopérabilité tend à rendre les systèmes plus communicants, plus évolutifs et dépassant la barrière des distances et des architectures matérielles. Elle rationalise également le développement des applicatifs en déterminant les briques élémentaires, leurs responsabilités et les rapports qu'elles entretiennent au travers des contrats bien définis que sont les interfaces. Elles laissent également libre court quant à leur implémentation tant au niveau des modèles de données qu'au niveau des traitements (du code). Ce type d'architecture va, à notre avis, permettre d'élever le niveau d'abstraction des applications et particulièrement celles qui manipulent une masse importante et complexe de données tel que les systèmes d'IAO.

En CAO par exemple, l'abstraction des représentations géométriques et la normalisation des services de conception permettent de voir la conception, non pas comme un graphe d'entités

géométriques et topologiques, mais comme le résultat d'un ensemble d'opérations de conception normalisées et donc reproductibles sur n'importe quel système cible. De ce point de vue la migration du modèle produit devrait être envisagée comme un transfert de la procédure universelle de conception plutôt que par le transfert de sa représentation géométrique. Ainsi la migration vers un système Y d'un modèle M du produit conçu suivant la procédure P sur un système X consiste en l'exécution de P par Y.



(a) Procédure de conception (b) Résultat exact (c) Résultat polyédrique

FIGURE 5.40 – *Résultat exact et polyédrique de la procédure de construction d'un tore*

Sur la figure 5.40 sont illustrés, la procédure de conception d'un tore par révolution et les résultats de son exécution sur des modeleurs respectivement exacts et polyédriques. Le tore est construit par révolution suivant un axe $a(\overrightarrow{y})$ d'un disque de centre $c(10, 0, 0)$, de normale $n(\overrightarrow{z})$ et de diamètre 5. A noter que si la procédure de conception est universelle, son résultat d'exécution par un modeleur donné dépend lui en revanche de la représentation utilisée. Enfin cette approche permettrait de diminuer l'information échangée entre systèmes soit moins de cent caractères dans l'exemple du tore.

Plutôt que de transférer les représentations du produit d'un système à l'autre, les utilisateurs et les applications utiliseraient les services de conception, simulation et fabrication de haut niveau implémentés sur divers systèmes hétérogènes. Dans le domaine de la conception, le modèle produit ne devrait plus être décrit en termes de données géométriques (du moins pas seulement) mais comme une description procédurale de conception plutôt qu'en termes de graphes de sommets, arètes et faces. A noter que cette description n'est pas sans rapport avec l'historique de conception à base de caractéristiques «feature based». Dans cette perspective, l'ensemble des services de conception permettrait de définir un vocabulaire normalisé de conception et ce indépendamment des systèmes cibles. Ainsi la conception pourrait être exprimée dans un langage universel et sauvegarder efficacement sur des supports de masse et éventuellement échanger, par ce biais, vers d'autres systèmes.

Annexe A

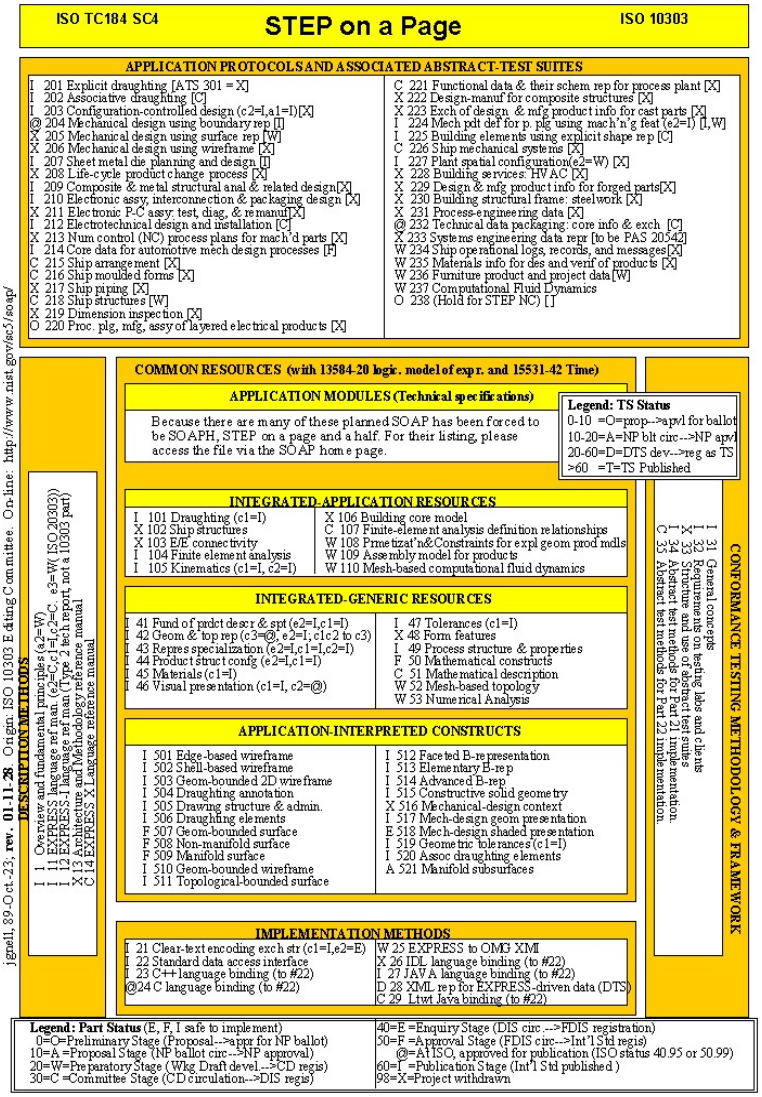


FIGURE A.1 – *Vue d'ensemble*

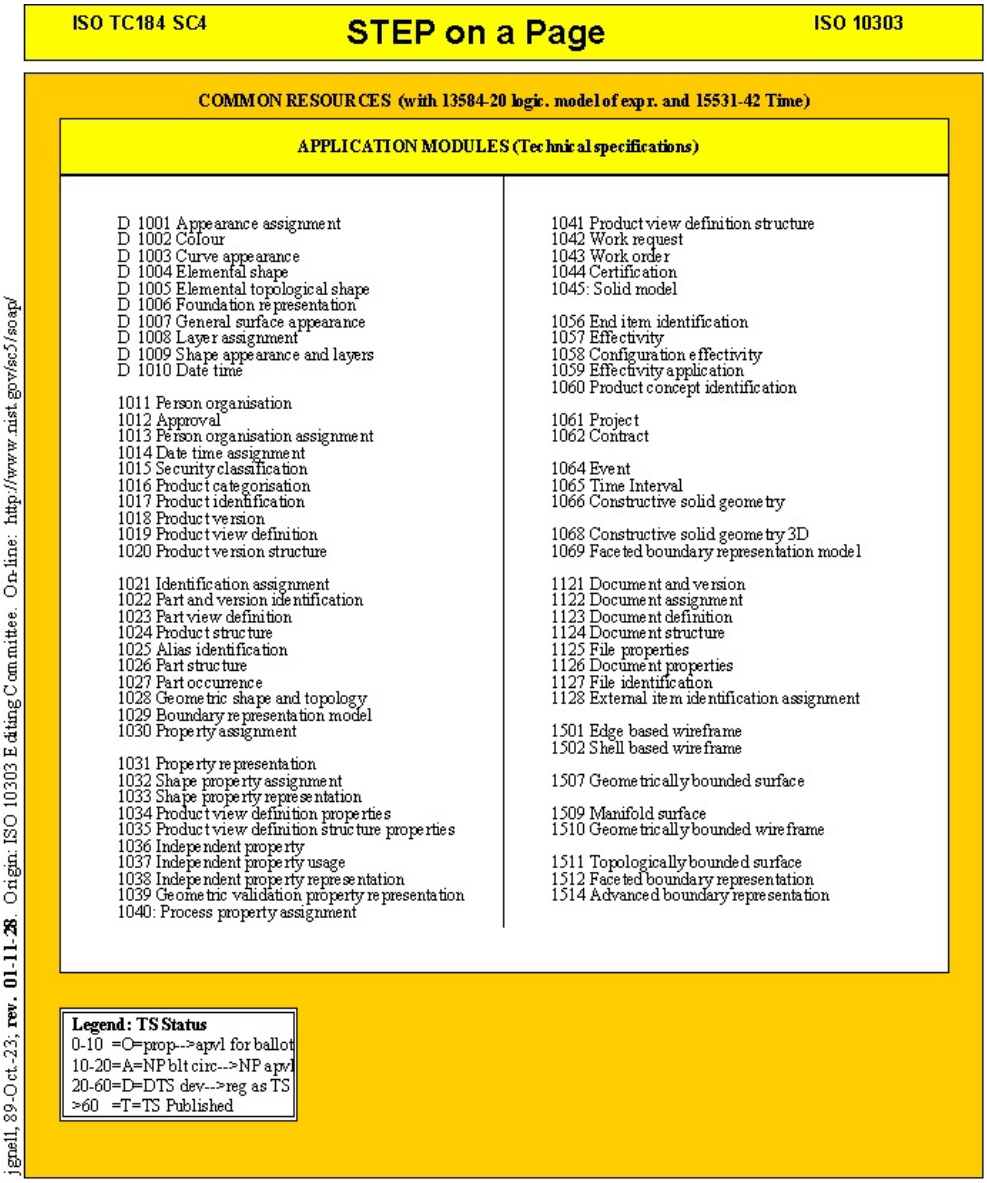
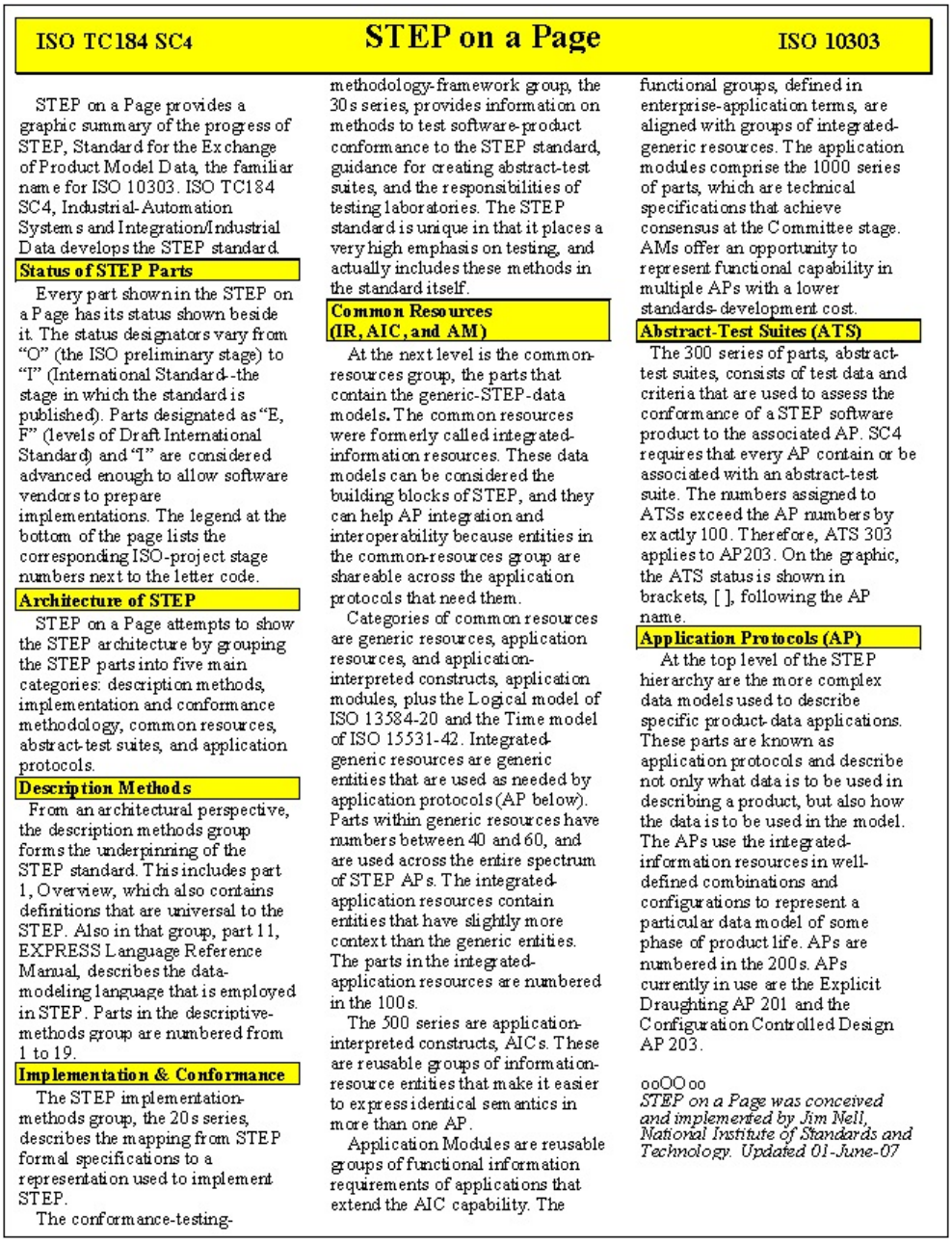


FIGURE A.2 – *Modules d'application*



ISO TC184 SC4

STEP on a Page

ISO 10303

STEP on a Page provides a graphic summary of the progress of STEP, Standard for the Exchange of Product Model Data, the familiar name for ISO 10303. ISO TC184 SC4, Industrial-Automation Systems and Integration/Industrial Data develops the STEP standard.

Status of STEP Parts

Every part shown in the STEP on a Page has its status shown beside it. The status designators vary from "O" (the ISO preliminary stage) to "I" (International Standard--the stage in which the standard is published). Parts designated as "E, F" (levels of Draft International Standard) and "I" are considered advanced enough to allow software vendors to prepare implementations. The legend at the bottom of the page lists the corresponding ISO-project stage numbers next to the letter code.

Architecture of STEP

STEP on a Page attempts to show the STEP architecture by grouping the STEP parts into five main categories: description methods, implementation and conformance methodology, common resources, abstract-test suites, and application protocols.

Description Methods

From an architectural perspective, the description methods group forms the underpinning of the STEP standard. This includes part 1, Overview, which also contains definitions that are universal to the STEP. Also in that group, part 11, EXPRESS Language Reference Manual, describes the data-modeling language that is employed in STEP. Parts in the descriptive-methods group are numbered from 1 to 19.

Implementation & Conformance

The STEP implementation-methods group, the 20s series, describes the mapping from STEP formal specifications to a representation used to implement STEP.

The conformance-testing-methodology-framework group, the 30s series, provides information on methods to test software-product conformance to the STEP standard, guidance for creating abstract-test suites, and the responsibilities of testing laboratories. The STEP standard is unique in that it places a very high emphasis on testing, and actually includes these methods in the standard itself.

Common Resources (IR, AIC, and AM)

At the next level is the common-resources group, the parts that contain the generic-STEP-data models. The common resources were formerly called integrated-information resources. These data models can be considered the building blocks of STEP, and they can help AP integration and interoperability because entities in the common-resources group are shareable across the application protocols that need them.

Categories of common resources are generic resources, application resources, and application-interpreted constructs, application modules, plus the Logical model of ISO 13584-20 and the Time model of ISO 15531-42. Integrated-generic resources are generic entities that are used as needed by application protocols (AP below). Parts within generic resources have numbers between 40 and 60, and are used across the entire spectrum of STEP APs. The integrated-application resources contain entities that have slightly more context than the generic entities. The parts in the integrated-application resources are numbered in the 100s.

The 500 series are application-interpreted constructs, AICs. These are reusable groups of information-resource entities that make it easier to express identical semantics in more than one AP.

Application Modules are reusable groups of functional information requirements of applications that extend the AIC capability. The functional groups, defined in enterprise-application terms, are aligned with groups of integrated-generic resources. The application modules comprise the 1000 series of parts, which are technical specifications that achieve consensus at the Committee stage. AMs offer an opportunity to represent functional capability in multiple APs with a lower standards-development cost.

Abstract-Test Suites (ATS)

The 300 series of parts, abstract-test suites, consists of test data and criteria that are used to assess the conformance of a STEP software product to the associated AP. SC4 requires that every AP contain or be associated with an abstract-test suite. The numbers assigned to ATSs exceed the AP numbers by exactly 100. Therefore, ATS 303 applies to AP203. On the graphic, the ATS status is shown in brackets, [ ], following the AP name.

Application Protocols (AP)

At the top level of the STEP hierarchy are the more complex data models used to describe specific product-data applications. These parts are known as application protocols and describe not only what data is to be used in describing a product, but also how the data is to be used in the model. The APs use the integrated-information resources in well-defined combinations and configurations to represent a particular data model of some phase of product life. APs are numbered in the 200s. APs currently in use are the Explicit Draughting AP 201 and the Configuration Controlled Design AP 203.

ooOOoo

STEP on a Page was conceived and implemented by Jim Nell, National Institute of Standards and Technology. Updated 01-June-07

FIGURE A.3 – *Synthèse*

Annexe B

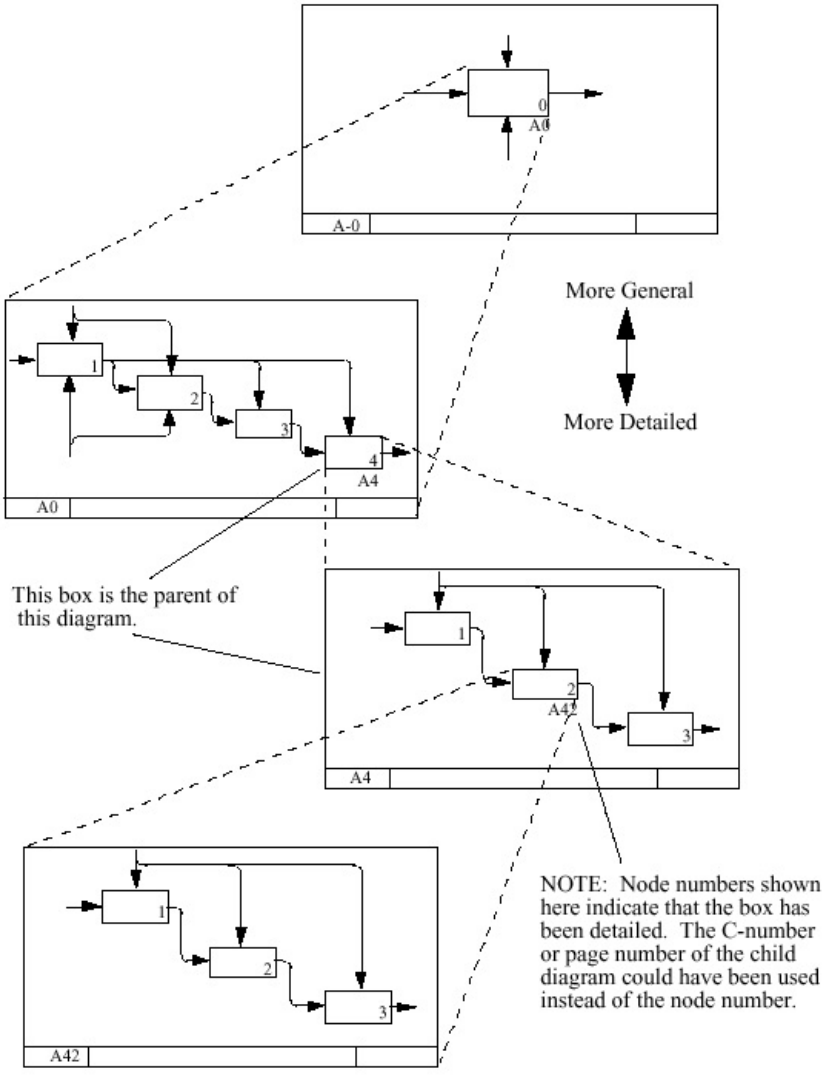


FIGURE B.1 – *Décomposition fonctionnelle*

Bibliographie

[1] Arbouy S. *STEP Concepts fondamentaux*. Afnor edition, 1994. Paris.

[2] Med Bouazza. *La norme STEP*. Hermès edition, 1995.

[3] Bernard Alain et Taillandier Georges. *Le prototypage rapide*. Hermès edition, 1998. Paris.

[4] G.M. Nijssen. Current issues in conceptual schema concepts. *Architecture and models in data base management systems*, pages 31–66, Juin 1977. Amsterdam, North-Holland.

[5] H. Habrias. *Le modèle relationnel binaire. Méthode Niam*. Eyrolles, eyrolles edition, 1988. préface de P. Jeulin et P. Sauge.

[6] National Institue of Standards and Technology. IdefØ integration definition for function modeling. Technical report, Federal Information Processing Standards Publication, 1993.

[7] Knowledge Based Systems Incorporated. Idef1 information modeling - a reconstruction of the original air force wright aeronautical laboratory technical report afwal-tr-81-4023. Technical report.

[8] National Institue of Standards and Technology. Idef1x integration definition for information modeling. Technical report, Federal Information Processing Standards Publication, 1993.

[9] Knowledge Based Systems Incorporated. Idef3 process descrition capture method report. Technical report, Information Integration for Concurrent Engineering (IICE), 1995.

[10] Knowledge Based Systems Incorporated. Idef4 object oriented design method report. Technical report, Information Integration for Concurrent Engineering (IICE), 1995.

[11] Knowledge Based Systems Incorporated. Idef5 ontology description capture method report. Technical report, Information Integration for Concurrent Engineering (IICE), 1995.

[12] Knowledge Based Systems Incorporated. Idef5 toward a method for business constraint discovery method. Technical report, Information Integration for Concurrent Engineering (IICE), 1995.

[13] E.F. Codd. A relational model of data for large shared data banks. *Communications of ACM*, 13(6), 1970.

[14] E.F. Codd. Further normalization of the database relation model. *Database system*, 1972. Prentice Hall Englewood Cliffs, NJ.

[15] P.P.S Chen. The entity relationship model - towards a unified view of data. *ACM TODS*, 1(1) :9–36, 1976.

[16] Ovidiu S. Noran. Business modelling : Uml vs idef. Technical report, Griffith University, School of Computing and Information Technology, Juin 2000. available at http ://www.cit.gu.edu.au/∼noran/cit_6114.

[17] Sara Willliams et Charlie Kindel. The component object model specification. Technical report, Developer Relations Group Microsoft Corporation, Décembre 1995. available at http ://www.microsoft.com/oledev/olecom/title.htm.

[18] N. Brown et C. Kindel. Distributed component object model protocol - dcom/1.0. Technical report, 1996. available at http ://www.microsoft.com/oledev/olecom/draft-brown-dcom-v1-spec-01.txt.

[19] OSF. Aes/distributed computing - remote procedure call, revision b. Technical report, Open Software Foundation, 1995. available at http ://www.osf.org/mall/dce/free_dce.htm.

[20] Y. Gardan. *La CFAO introduction, techniques et mise en oeuvre*. Hermès edition, 1991.

[21] Grady Booch et James Rumbaugh et Ivar Jacobson. *The Unified Modeling Langage user guide*. The Addison-Wesley object technology series. Addison-wesley edition, 1998.

[22] Rémy Fannader et Hervé Leroux. *UML principes de modélisation*. Dunod edition, 1999. Paris.

[23] OMG Manufacturing Domain Task Force. Revised submission cad services v 1.0, joint proposal to the omg in response to omg manufacturing domain task force cad services rfp. Technical report, Object Management Group, Juillet 2001.

[24] OMG Manufacturing Domain Task Force. Revised submission of pdm enablers, joint proposal to the omg in response to omg manufacturing domain task force. Technical report, Object Management Group, Février 1998.

[25] OMG. Distributed simulation system final adopted specification. Technical report, Object Management Group, Février 2002.

[26] OMG. Model driven architecture in manufacturing discussion. Technical report, Object Management Group, Juin 2002.

[27] Unknown. Step and omg product data management specifications a guide for decision makers. Cambridge Meeting, Novembre 1999.

[28] Dave Price. Iso express to uml mapping - describes the mapping of the iso express modeling language to uml. Technical report, Object Management Group, Avril 2002.

[29] Dave Price. Express/uml harmonization - a meta-model of express in uml for mof and uml to express. Technical report, Object Management Group, Avril 2002.

[30] Dave Price. Uml metamodel of iso express - uml metamodel of iso express modeling language. Technical report, Object Management Group, Avril 2002.

[31] Anne L. Marsan et Vinod Kumar et Debasish Dutta. An assessment of data requirements and data transfer formats for layered manufacturing. Technical report, National Institute of Standards and Technology.

[32] Famieson R. J. Direct slicing of cad models for rapid prototyping. *ISATA 94*, 1994. Aachen, Allemagne.

[33] Mani K. et Kulkarni P. et Dutta D. Region-based adaptative slicing. *in Computer-Aided Design*, 31 :317–333, 1999.

[34] Cha-Soo Jun et Dong-Soo Kim et JiSeon Hwang. Surface slicing algorithm for rapid prototyping and machining. *Geometric Modeling and Processing, IEEE Computer Society*, pages 373–382, 2000. Hong Kong, Chine.

[35] Cha-Soo Jun et Dong-Soo Kim et Deok-Soo Kim. Surface slicing algorithm based on topolgy transition. *Computer-Aided Design*, 33(11) :825–838, Septembre 2001.

[36] Zhiwen Zhao et Luc Laperrière. Adaptative direct slicing of the solid model for rapid prototyping. Trois-rivières, Québec, Canada.

[37] P. Vuyyuru et C. F. Kirschman et G. Fadel. A nurbs based approach for rapid product realization. Technical report, Intelligent Design and Rapid Prototyping Laboratory, Department of Mechanical Engineering, Clemson University. Clemson, SC 29634–0921.

[38] John F. Miller. Cad requirements for rapid prototyping tutorial. *Rapid Prototyping and Manufacturing 94, Society of Manufacturing Engineers*, 1994.

[39] Hardwick M. *The role of STEP to integrate design and solid freeform fabrication.* Actes du 1er congrès international de génie industriel, Montréal, 1995.

[40] Field B. et Thompson E. Objectivity first pdes/step shared database and express prototypes. *Proceedings of Autofact 91 conferences*, 1991. Chicago.

[41] Gilman C. R. et Rock S. J. The use of step to integrate design and solid freeform fabrication. *Symposium*, 1995. Austin.

[42] Mony C. Integration of rapid product development technologies information models using step. *Rapid product development technologies, Proceedings of SPIE international conference*, 2910 :156–165, 1996. Boston.

[43] D. Dutta et V. Kumar et M.J. Pratt. Towards step-based data transfer in layered manufacturing. *Proceedings of the Tenth International IFIP WG5.2/5.3 Conference PROLAMAT 98*, 1998.

[44] Unknown. Common layer interface version 1.31. Rapid prototyping techniques, Brite-Euram project BE5278, 1994.

[45] 3D Systems Inc. Stereolithography interface specification. Technical report, 3D Systems Inc., Juin 1988.

[46] Stephen J. Rock et Michael J. Wozny. A flexible file format for solid freeform fabrication. *Proceedings Solid Freeform Fabrication Symposium*, pages 1–12, 1991.

[47] Stephen J. Rock et Michael J. Wozny. Generating topological information from a "bucket of facets". *Proceedings Solid Freeform Fabrication Symposium*, pages 251–259, 1992.

[48] Sara Anne McMains. *Geometric Algorithms and Data Representation for Solid Freeform Fabrication.* PhD thesis, University of California, Berkeley, 2000.

[49] Sara A. McMains et Carlo H. Sequin et Jordan P. Smith. Sif : A solid interchange format for rapid prototyping. EECS Computer Science Division, University of California, Berkeley, CA 947201776.

[50] Jordan P. Smith et Sara A. McMains et Carlo H. Séquin. Sif : A solid interchange format for web-based prototyping. EECS Computer Science Division, University of California, Berkeley, CA 947201776.

[51] Vaclav Skala et Martin Kuchar. Hash function for geometry reconstruction in rapid prototyping. *Conference on scientific computing, Proceedings of algoritmy*, pages 379–387, 2000.

[52] Morvan Stéphane et Fadel George. Ivecs : An interactive virtual environment for the correction of .stl files. *in proceedings of The Design Engineering Technical Conferences and Computers in Engineering Conference*, pages 18–22, Août 1996.

[53] A. A. G. Requicha. Representation for rigid solids : Theory, methods and systems. *ACM Computer Survey*, 12(4) :437–464, Décembre 1980. Production Automation Project, University of Rochester, Rochester, NY.

[54] A. A. G. Requicha et H. B. Voelcker. Solid modeling : A historical summary and contemporary assessment. *IEEE Computer Graphics and Applications*, 2(2) :52–37, Octobre 1983.

[55] Unknown. Solid modeling : Current status and research directions. *IEEE Computer Graphics and Applications*, 3(7) :25–37, Octobre 1983.

[56] H. B. Voelcker. An introduction to padl : Characteristics, status and rationale. Technical Report 22, University of Rochester, Rochester, NY, Décembre 1974. Production Automation Project.

[57] H. B. Voelcker. The padl-1.0/2 system for defining and displaying solid objects. *ACM Computer Graphic*, 12(3) :257–263, Août 1978.

[58] C. M. Brown. Padl-2 : A technical summary. *IEEE Computer Graphics and Applications*, 2(2) :69–84, Mars 1982.

[59] A. A. G. Requicha et H. B. Voelcker. Constructive solid geometry. Technical Report 25, University of Rochester, Rochester, NY, Novembre 1977. Production Automation Project.

[60] A. A. G. Requicha. Mathematical models of rigid solid objects. Technical Report 28, University of Rochester, Rochester, NY, Novembre 1977. Production Automation Project.

[61] A. A. G. Requicha et R. B. Tilove. Mathematical foundations of constructive solid geometry : General topology of closed regular sets. Technical Report 27a, University of Rochester, Rochester, NY, Juin 1978. Production Automation Project.

[62] A. A. G. Requicha et H. B. Voelcker. Boolean operations in solid modeling : Boundary evaluation and merging algorithms. *Proceedings of the IEEE*, 73(1), Janvier 1985.

[63] R. B. Tilove et A. A. G. Requicha. Closure of boolean operations on geometric entities. *Computer Aided Design*, 12(5) :219–220, Septembre 1980.

[64] R. B. Tilove. Set membership classification : A unified approch to general intersection problems. *IEEE trans. Comp.*, C-29(10) :874–883, Octobre 1980.

[65] Martti Mäntylä. Boolean operations of 2-manifolds through vertex neighborhood classification. *ACM Transactions on Graphics*, 5(1) :1–29, 1986.

[66] A. A. G. Requicha et J. R. Rossignac. Constructive non-regularized geometry. *Computer-Aided Design*, 23(1), Janvier 1991.

[67] A. A. G. Requicha et J. R. Rossignac. Solid modeling and beyond. *Proceedings of the IEEE*, Septembre 1992.

[68] Ian C. Braid. Boundary modeling. *In Fundamental Developments of Computer-Aided Geometric Modeling*, 1993. L. Piegl editor, Academic Press, London.

[69] A. E. Middleditch. The 'bug' and beyond. *CSG 94 Set-theoretic Solid Modelling : Techniques and Applications*, pages 1–16, Avril 1994. Information Geometers Ltd, Winchester, UK.

[70] Kevin Weiler. The radial edge structure : A topological representation for non-manifold geometric boundary modeling. *Geometric Modeling for CAD Applications*, pages 3–36, 1988. North-Holland, Amsterdam.

[71] E. L. Gursoz et Y. Choi et F. B. Prinz. Vertex-based representation of non-manifold boundaries. *In Geometric Modeling for Product Engineering*, pages 107–130, 1990. Elsevier Science Publishers B. V., North Holland.

[72] E. Gursoz et Y. Choi et F. B. Prinz. Boolean set operations on non-manifold boundary representation objects. *Computer Aided Design*, 23(1) :33–39, Février 1991.

[73] Kevin Weiler. Vertex neighborhood topological information and data structures in a non-manifold environment. Technical report, Autodesk, Avril 1996.

[74] Y. E. Kalay. Determining the spatial containment of a point in general polyhedra. *Computer Graphics Image Processing*, 19(4) :303–334, Août 1982.

[75] Sashidhar Guduri et Richard H. Crawford et Joseph J. Beaman. A method to generate exact contour files for solid freeform fabrication. *Proceedings Solid Freeform Fabrication Symposium*, pages 95–101, Août 1992. University of Texas at Austin.

[76] Jarek Rossignac et David Cardoze. Matchmaker : Manifold breps for non-manifold r-sets. *Fifth Symposium on ACM Solid Modeling and Applications*, pages 31–41, Juin 1999. Ann Arbor, MI.

Table des figures

Liste des tableaux

Résumé

Dans le contexte de l'ingénierie collaborative et de l'entreprise étendue, la collaboration des différents acteurs passe par l'interopérabilité des systèmes d'IAO (Ingénierie Assistée par Ordinateur), celle-ci devant être fondée sur des solutions technologiques normalisées.

Cantonnée aux échanges de fichiers suivant différents formats (IGES, SET, VDA etc.), l'interopérabilité des systèmes de CAO (Conception Assistée par Ordinateur) s'est longtemps heurtée et se heurte encore aujourd'hui à l'hétérogénéité des représentations utilisées. Au milieu des années quatre vingts, le projet STEP (STandard for Exchange of Product data model) propose de normaliser les représentations du produit utilisées dans les divers secteurs de l'industrie pour en permettre l'échange et le partage. Vingt ans plus tard, le processus de normalisation n'est toujours pas terminé et ses retombées toujours attendues...

Parallèlement dans le secteur des télécommunications, le problème de l'interopérabilité logicielle se pose de manière générale : comment faire collaborer des ressources hétérogènes et distribuées ? Basée sur les concepts orientés objet d'abstraction et d'encapsulation, les architectures de médiation (CORBA, DCOM) permettent à des objets hétérogènes et distribués de collaborer au travers d'interfaces bien définies. Récemment l'OMG (Object Management Group) décide d'appliquer cette approche à l'IAO. Les services de CAO, de simulation distribuée et de GDT (Gestion de Données Techniques) sont alors normalisés.

A travers la présentation détaillée des normes STEP, DCOM et CORBA, cette thèse présente les différences fondamentales entre ces deux approches. Nous proposons ensuite, dans la continuité des travaux de l'OMG, de revoir l'interopérabilité entre systèmes de CAO et systèmes de prototypage au moyen de la médiation. Nous proposons pour cela de spécifier les interfaces d'un certain nombre de composants usuels des systèmes de prototypage : les modèles STL et STL topologique, les services de tranchage et de remplissage. Nous donnons également une formalisation unifiée des algorithmes de tranchage et de remplissage, traitant des cas de singularités au regard des voisinages. Enfin, nous présentons un système de prototypage rapide utilisable via Internet, développé à des fins de validation.