Interopérabilité sémantique des connaissances des modèles de produits à base de features

Samer Abdul Ghafour

N° d'ordre 113-2009 Année 2009

THESE DE L'UNIVERSITE DE LYON

Délivrée par

L'UNIVERSITE CLAUDE BERNARD LYON 1

ECOLE DOCTORALE INFORMATIQUE INFORMATION ET SOCIÉTÉ

pour l'obtention du diplôme de

« DOCTORAT EN INFORMATIQUE »

(arrêté du 7 août 2006)

présentée et soutenue publiquement le **09 juillet 2009**

par

M. ABDUL GHAFOUR Samer

INTEROPÉRABILITÉ SÉMANTIQUE DES CONNAISSANCES DES MODÈLES DE PRODUITS À BASE DE FEATURES

Directeur de thèse : Pr. GHODOUS Parisa

Composition du Jury :

Rapporteurs : Mme CAUVET Corine
Mme FAUDOT Dominique

Examinateurs : M. GARDAN Yvon
M. GIRAUDIN Jean-Pierre
Mme GHODOUS Parisa, Directrice de thèse
M. SHARIAT Behzad, Co-encadreur
Mme PERNA Eliane, Co-encadreur
Mme MADDALENA Julie, Responsable Industriel

À mes parents,

À mon épouse, Rim

À mon frère jumeau, Ayman

Remerciements

Je tiens tout d'abord à remercier mes directeurs de thèse, Madame Parisa GHODOUS, Madame Eliane PERNA, et Monsieur Behzad SHARIAT pour l'aide compétente qu'ils m'ont apportée, pour leur patience, pour leur encouragement, et pour leur disponibilité tout au long de mon travail de thèse. Ils ont su me laisser la liberté nécessaire à l'accomplissement de mes travaux, tout en y gardant un œil critique et avisé. Leurs conseils m'ont été très précieux pour structurer le travail et pour améliorer la qualité des différentes sections.

Cette thèse a bénéficié d'une convention CIFRE. Je tiens à exprimer ma profonde gratitude à l'ANRT (Association Nationale de la Recherche et de la Technologie) et à la société DATAKIT, partenaire industriel, pour le financement de cette thèse. Je remercie chaleureusement Monsieur Francis CADIN, le PDG de la société DATAKIT, de m'avoir accepté dans la société et d'avoir assuré les conditions nécessaires pour le bon accomplissement de mon travail. De plus, il m'a fait l'honneur de participer au Jury de cette thèse.

J'adresse également ma reconnaissance au directeur du Laboratoire d'InfoRmatique en Image et Systèmes d'information (LIRIS), pour la disposition des moyens nécessaires au bon déroulement du travail au sein du laboratoire. Je tiens aussi à mentionner le plaisir que j'ai eu à travailler au sein du laboratoire, et j'en remercie ici tous les membres.

Je remercie Madame Corine CAUVET, Professeur à l'Université Aix-Marseille, et Madame Dominique FAUDOT, Professeur à l'Université de Bourgogne, d'avoir bien voulu rapporter sur mon travail et d'avoir porté un regard si approfondi sur le manuscrit. Je suis très reconnaissant pour le temps consacré à ce travail ainsi que pour leurs remarques et suggestions.

Monsieur Yvon GARDAN, Professeur à l'Université de Reims-Champagne-Ardenne, et Monsieur Jean-Pierre GIRAUDIN, Professeur à l'Université Université Pierre-Mendès France, m'ont fait l'honneur de participer au Jury de soutenance; Je les remercie profondément d'avoir accepté de juger mon travail. Je remercie également Mademoiselle Julie MADDALENA, ingénieur CAO à la société DATAKIT, de bien vouloir faire partie du Jury et de ses précieuses remarques durant toute la thèse.

J’adresse mes remerciements aussi aux personnes avec lesquelles j’ai collaboré au cours de ma thèse. Tout d’abord, à Monsieur Roland MARANZANA, Professeur à l’ETS, Ecole de Technologie Supérieure, Université de Québec, avec qui j’ai échangé des points de vue sur certains concepts du domaine de la CFAO. Je remercie Monsieur Christophe PALANDRE, ingénieur à la société Volvo IT, avec qui j’ai eu des discussions très fructueuses sur l’intention de conception. Je remercie Monsieur Shaw FENG de NIST, (National Institute of Standards and Technologies), pour nos discussions sur la classification des features.

Je remercie vivement tous ceux sans qui cette thèse ne serait pas ce qu'elle est : aussi bien par les discussions, leurs suggestions et leurs contributions. Je pense ici en particulier à tous mes collègues à la société Datakit. Je remercie également Catarina Ferreira Da Silva pour nos discussions sur de nombreux sujets, mais aussi pour son implication dans mes travaux. Je remercie aussi Patrick Hoffmann, Moisés Dutra et tous les membres de l’équipe, SOC. Enfin, ces remerciements ne seraient pas complets sans mentionner les personnes qui ont consacré du temps pour la lecture et la correction du manuscrit : Julie, Olivier, Aurore, Michel, et Rim.

Le côté personnel avait une forte implication dans mon travail de thèse. J’adresse un grand merci à tous les amis qui m’ont soutenu, de près ou à distance, pendant toute la période de la thèse. Je pense tout particulièrement à Imane et Omar, pour leurs encouragements et leur assistance morale qui m'ont permis de faire cette thèse dans de bonnes conditions. Je remercie également Assia et Rachid, qui, malgré la distance, ont su m’entourer par leur aide et leur attention. Je crois avoir trouvé en eux tous une famille qui m'a aidé dans la vie lorsque j'en avais besoin. Qu’ils trouvent en ces lignes mon amour et ma profonde reconnaissance pour tout. Je suis reconnaissant également pour le soutien de tous mes amis qui m’ont accompagné durant cette période, que je ne peux pas tous citer, mais particulièrement Rami, Dima, Brigitte, Louis-Noël, Ségolène, Osman, Nabil, Ahmad, Maher, Pauline, Nihad, Ouissal, Mahmoud et encore tous ceux qui méritent d’être remerciés, ils se reconnaitront…

Cela va de soi, je remercie évidemment ma famille pour son irremplaçable et inconditionnel soutien. Un merci plein d’amour et d’affection d’être restée si près de moi, malgré les distances. Cette thèse est un peu la leur, aussi. Elle représente l'aboutissement du soutien et des encouragements que mes parents m’ont prodigués tout au long de ma vie. Que ma famille en soit remerciée par cette modeste dédicace. Je n’oublie pas de remercier ma belle-famille, surtout Rola, Rami et Rana. Le dernier remerciement, mais pas le moindre, je l’adresse à mon épouse Rim, pour tout. Elle a su bien m’entourer par son amour et son tendre soutien. Merci d’être toujours à côté de moi.

Abréviations

API	Application Programming Interface
APs	Applications Protocols
BREP	Boundary Representation
CAO/CAD	Conception Assistée par Ordinateur
CAPP	Computer Aided Process Planning
CFAO	Conception et Fabrication Assistées par Ordinateur
CSG	Constructive Solid Geometry
DL	Description Logics
DXF	Drawing Exchange Format
FAO/CAM	Fabrication Assistée par Ordinateur
IGES	Initial Graphics Exchange Specification
IRs	Integrated Resources
ISO	International Organization for Standardization
JESS	Java Expert System Shell
KB	Knowledge Base
KIF	Knowledge Interchange Format
LDs	Logiques de Description
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	RDF Schema
SET	Standard d'Echange et de Transfert
STEP	STandard for the Exchange of Product model data
SWRL	Semantic Web Rule Language
URI	Uniform Resource Identifier
VDA-FS	Verband Der Automobilindustrie – Flächenschnittstelle, *allemand*
W3C	World Wide Web Consortium
WWW	World Wide Web
XAO	(X: conception, modification, exploration, fabrication, *etc.*)

Dans un environnement collaboratif de développement de produit, plusieurs acteurs, ayant différents points de vue et intervenant dans plusieurs phases du cycle de vie de produit, doivent communiquer et échanger des connaissances entre eux. Ces connaissances, existant sous différents formats hétérogènes, incluent potentiellement plusieurs concepts tels que l'historique de conception, la structure du produit, les features, les paramètres, les contraintes, et d'autres informations sur le produit. Les exigences industrielles de réduction du temps et du coût de production nécessitent l'amélioration de l'interopérabilité sémantique entre les différents processus de développement afin de surmonter ces problèmes d'hétérogénéité tant au niveau syntaxique, structurel, que sémantique.

Dans le domaine de la CAO, la plupart des méthodes existantes pour l'échange de données d'un modèle de produit sont, effectivement, basées sur le transfert des données géométriques. Cependant, ces données ne sont pas suffisantes pour saisir la sémantique des données, telle que l'intention de conception, ainsi que l'édition des modèles après leur échange. De ce fait, nous nous sommes intéressés à l'échange des modèles « intelligents », autrement dit, définis en termes d'historique de construction, de fonctions intelligentes de conception appelées features, y compris les paramètres et les contraintes.

L'objectif de notre thèse est de concevoir des méthodes permettant d'améliorer l'interopérabilité sémantique des systèmes CAO moyennant les technologies du Web Sémantique comme les ontologies OWL DL et le langage des règles SWRL. Nous avons donc élaboré une approche d'échange basée sur une ontologie commune de features de conception, que nous avons appelée CDFO « *Common Design Features Ontology* », servant d'intermédiaire entre les différents systèmes CAO. Cette approche s'appuie principalement sur deux grandes étapes. La première étape consiste en une homogénéisation des formats de représentation des modèles CAO vers un format pivot, en l'occurrence OWL DL. Cette homogénéisation sert à traiter les hétérogénéités syntaxiques entre les formats des modèles. La deuxième étape consiste à définir des règles permettant la mise en correspondance sémantique entre les ontologies d'application de CAO et notre ontologie commune. Cette méthode de mise en correspondance se base principalement, d'une part, sur la définition explicite des axiomes et des règles de correspondance permettant l'alignement des entités de différentes ontologies, et d'autre part sur la reconnaissance automatique des correspondances sémantiques supplémentaires à l'aide des capacités de raisonnement fournies par les moteurs d'inférence basés sur les logiques de description. Enfin, notre méthode de mise en correspondance est enrichie par le développement d'une méthode de calcul de similarité sémantique appropriée pour le langage OWL DL, qui repose principalement sur les composants des entités en question tels que leur description et leur contexte.

TITRE en anglais

Semantic Interoperability of Knowledge in Feature-based CAD Models

RESUME en anglais

A major issue in product development is the exchange and sharing of product knowledge among many actors. This knowledge includes many concepts such as design history, component structure, features, parameters, constraints, and more. Heterogeneous tools and multiple designers are frequently involved in collaborative product development, and designers often use their own terms and definitions to represent a product design. Thus, to efficiently share design information among multiple designers, the design intent should be persistently captured and the semantics of the modeling terms should be semantically processed both by design collaborators and intelligent systems.

Regarding CAD models, most of the current CAD systems provide feature-based design for the construction of solid models. Features are devised to carry, semantically, product information throughout its life cycle. Consequently, features should be maintained in a CAD model during its migration among different applications. However, existing solutions for exchanging product information are limited to the process of geometrical data, where semantics assigned to product model are completely lost during the translation process. Current standards, such as ISO 10303, known as STEP have attempted to solve this problem, but they define only syntactic data representation so that semantic data integration is not possible. Moreover, STEP does not provide a sound basis to reason with knowledge.

Our research investigates the use of Semantic Web technologies, such as ontologies and rule languages; *e.g.* SWRL, for the exchange of "*intelligent*" CAD models among different systems, while maintaining the original relations among entities of the model. Thus, we have proposed an ontological approach based on the construction of a common design features ontology, used as an *Interlingua* for the exchange of product data. This ontology is represented formally with OWL DL. Furthermore, axioms and mapping rules are defined to achieve the semantic integration between the applications ontologies and the common ontology. The integration process relies basically on reasoning capabilities provided by description logics in order to recognize automatically additional mappings among ontologies entities. Furthermore, the mapping process is enhanced with a semantic similarity measure in order to detect similar design features. However, this will enable data analysis, as well as manage and discover implicit relationships among product data based on semantic modeling and reasoning.

MOTS-CLES
CAO – Feature – Ontologie – Web Sémantique – SWRL – Interopérabilité – Format neutre – STEP

Laboratoire LIRIS – Laboratoire d'InfoRmatique en Image et Systèmes d'information
Université Claude Bernard Lyon1 - Bâtiment Nautibus Campus de la Doua - 8, Bd Niels Bohr 69622 Villeurbanne Cedex

Table des matières

Liste des Figures

Liste des Tables

Introduction

Les changements économiques dans la société impliquent une adaptation significative de l'industrie dans l'utilisation des technologies de développement de produit. La compétitivité des produits industriels se fonde essentiellement sur des critères de coût, de qualité, d'innovation et de disponibilité.

Dans le monde industriel actuel, l'optimisation du processus de conception et de réalisation des produits constitue un nouveau challenge pour les entreprises. Des systèmes ont été développés pour faire circuler, partager, et exploiter toutes les informations relatives aux produits, et des travaux de recherches ont été élaborés pour intégrer tous les paramètres du cycle de vie d'un produit dans le processus de conception et de réalisation. L'enjeu majeur est celui de la réduction des délais de mise sur le marché des nouveaux produits. Ceci ne peut être atteint que grâce aux méthodes d'ingénierie coopérative et simultanée (Ghodous, 2003).

L'ingénierie simultanée a d'abord concerné la réduction des délais par la réalisation de l'ensemble de processus de manière simultanée et répartie. Le concept s'est ensuite étendu à la coopération entre les participants au développement du produit, à l'amélioration de la qualité et des coûts globaux, puis d'une manière générale à l'amélioration de toutes les caractéristiques du produit. Ce nouveau mode de travail implique une évolution profonde à la fois des modes de pensée, des méthodes, des organisations, des techniques et des outils à tous les stades du développement de produit. Dans ce cadre, il devient donc primordial de disposer de moyens informatiques de modélisation, d'échange et de partage des connaissances descriptives du produit en phase de développement et de ses processus associés.

Le développement de produit consiste en différentes phases. Les phases principales sont « la conception » et « la fabrication ». Les techniques de CAO, *Conception Assistée par Ordinateur*, sont utilisées pour concevoir un produit, et les techniques de FAO, *Fabrication Assistée par Ordinateur*, pour la fabrication de ce produit. Plus précisément, les systèmes CAO permettent de représenter un produit en termes d'entités géométriques et de features de conception, alors que la FAO concerne les tâches de transformation d'un produit de matières premières en produit fini. Ainsi, ces techniques doivent échanger des données entre elles (Han, 1996). Cette intégration entre les différentes applications exige que l'information soit

suffisamment précise et complète. Par conséquent, une représentation unique est recherchée pour constituer un moyen efficace unifiant les informations nécessaires pour l'échange de données durant le cycle de développement d'un produit.

Problématique

Le problème de communication des données d'un produit durant son développement est assez vaste à résoudre, mais nous restreindrons notre étude sur une partie de cette problématique, la phase de conception. La conception est une phase fondamentale dans le cycle du développement d'un produit, et a également une influence importante sur le temps de mise sur le marché et les coûts de production.

D'abord, les modeleurs géométriques ont été considérés par les chercheurs comme étant la représentation appropriée pour un produit en développement. Ensuite, une approche à base de « features » a été proposée pour augmenter les capacités des modeleurs géométriques (Case, et al., 1994). Les méthodes à base de « features » ont émergé pour répondre aux besoins cruciaux de l'industrie dans la conception et dans la fabrication, particulièrement le besoin de réduire le temps de développement d'un produit (Shah, et al., 1995). De nos jours, différents systèmes CAO commerciaux existent sur le marché. La plupart reposent sur une modélisation paramétrique basée sur l'historique de construction. Ainsi, un modèle conçu est le résultat d'un ensemble de fonctions de modélisation, appelées aussi features, créés par le concepteur durant le processus de modélisation.

Etant donné la diversité de la gamme de systèmes CAO existant sur le marché industriel, l'échange de données entre ces différents systèmes devient de plus en plus indispensable. Le nombre croissant de ces systèmes et leur diversité nécessitent de rendre ces systèmes interopérables afin de répondre aux exigences de l'industrie en termes de diminution du coût et de temps de mise sur le marché des nouveaux produits. L'objectif principal de l'interopérabilité est de permettre le partage et l'échange des données d'un produit entre différentes applications durant le cycle de vie d'un produit. La réduction du coût de production et de temps de développement d'un produit ne peut pas être acquise à cause de certaines restrictions liées aux méthodes actuelles d'échange de données. Effectivement, Les techniques classiques utilisées pour l'échange de données ne permettent pas une communication optimale entre les agents (humains et logiciels) puisqu'elles sont souvent limitées à l'échange de données géométriques, où les paramètres et les informations des

features ne sont plus maintenus (Fu, et al., 2003). Ainsi, une fois un modèle CAO à base de features transféré par format neutre, tel que STEP ou IGES, l'intention de conception comprenant l'historique de construction, le features, les paramètres et les contraintes ne peut plus être retrouvée. Cette perte d'information engendre dans les systèmes destinataires des modèles purement géométriques, appelés aussi modèles « *inertes* », puisque leur édition exige un travail laborieux. Cependant, le modèle d'origine est un modèle « *intelligent* », dans le sens où son édition peut être aisément effectuée grâce à l'accès aux paramètres des features inclus. Ce problème d'édition des modèles CAO transférés dans le système destinataire est dû principalement à la perte de la sémantique des données, notamment le contexte du processus de conversion (Patil, et al., 2005). Un exemple sur la perte des informations sémantiques est illustré dans la figure 1.

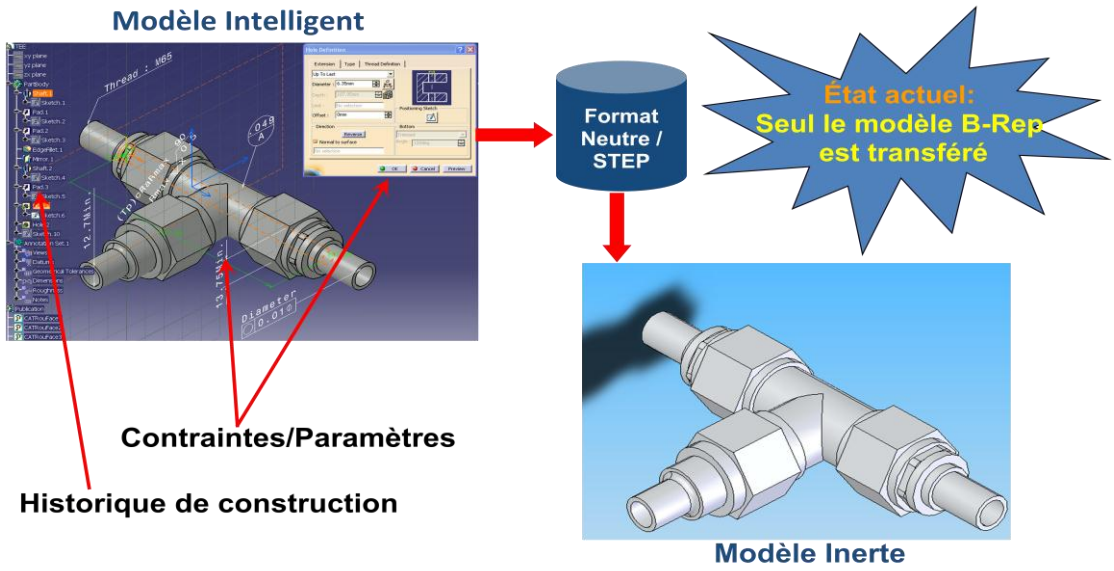


Figure 1. Conversion d'un modèle à base de features avec les standards actuels

Par conséquent, les concepteurs n'ont pas seulement besoin d'échanger les données géométriques des modèles, mais aussi de saisir les connaissances d'ingénierie sur la conception et le processus de développement de produit, y compris des spécifications, des règles de conception, des contraintes, et la logique de conception. Ainsi, les besoins de saisie de la connaissance de conception et de collaboration, de plus en plus fréquents, nécessitent le développement d'un environnement collaboratif permettant la représentation formelle, la saisie, l'extraction et la réutilisation de la connaissance d'un produit (Lutters, et al., 1997). L'amélioration du processus de développement d'un produit exige la création des modèles de produits ne contenant pas seulement des données géométriques, mais également de la

sémantique de conception qui permet le stockage systématique, le transfert, et la réutilisation des connaissances (Manley, 2006).

Objectif

La collaboration effective reste une tâche difficile à accomplir, vu l'hétérogénéité des systèmes utilisés et les différents points de vue des ingénieurs. Ces derniers utilisent souvent leurs propres termes et définitions pour représenter un modèle de produit. Ainsi, afin de partager efficacement l'intention de conception entre différents concepteurs, celle-ci doit être saisie avec pertinence et la sémantique des termes utilisés doit être interprétée de manière cohérente (Manley, 2006). La sémantique désigne la signification associée à une terminologie dans un contexte particulier.

Notre travail de thèse se situe à la fois dans les domaines de la représentation des connaissances et de la communication des connaissances techniques appliquées aux environnements d'ingénierie coopérative. L'objectif principal de la représentation des connaissances est d'exprimer formellement les connaissances d'un domaine, c'est-à-dire de façon interprétable par la machine. Il est donc nécessaire de concevoir des méthodes et d'outils intelligents capables d'échanger les données supplémentaires, notamment les features, d'un système CAO à un autre, tout en maintenant la sémantique associée aux données de produit. De nos jours, les standards de définition des features n'existent pas en mode unique, mais les développeurs des systèmes CAO ont conçu leur propre ensemble de définitions de features (Fu, et al., 2003). Par conséquent, le travail de standardisation de ces définitions vise à faciliter l'échange de données entre les différents systèmes CAO.

Dans ce contexte, une étape essentielle de notre travail consiste à analyser les techniques existantes, basées sur un format neutre, permettant l'échange des modèles procéduraux de CAO. Ces derniers sont définis en fonction de l'historique de construction et des fonctions intelligentes de construction, appelées features (Pratt, et al., 2005). Les informations comprenant le paramétrage et les contraintes associées au modèle doivent être également échangées. Ce mécanisme d'échange aboutira à la possibilité de modification ou de re-paramétrage des modèles échangés, au contraire des modèles géométriques qui sont difficiles ou parfois impossibles à éditer après leur transfert.

Les premiers travaux théoriques sur la représentation des connaissances montrent le caractère classificatoire des structures sémantiques. Ces travaux ont été suivis par de nombreuses applications qui ont prouvé l'utilité de la modélisation des connaissances pour le traitement de l'information. Toutefois, la collaboration effective entre les différents systèmes de développement d'un produit doit tenir compte de l'hétérogénéité de la sémantique associée aux données d'un produit depuis sa conception, et à travers les différentes activités de son développement. Ceci implique un niveau d'échange de données plus élevé qu'une simple conversion de la terminologie des différentes applications.

Méthodologie de notre approche

Pour améliorer le processus de développement d'un produit dans un environnement collaboratif, nous visons à élaborer une méthodologie d'échange des connaissances des modèles CAO à base de features. Cette méthodologie consiste à optimiser l'échange des données par la prise en compte de la sémantique associée aux données d'un produit. Ainsi, nous avons besoin de représenter formellement les features des modèles CAO par l'utilisation des technologies du Web Sémantique, telles que les ontologies OWL et le langage des règles SWRL. Ces technologies permettent de traiter les données intégrées dans un modèle CAO, et également de prendre en considération le contexte dans lequel ces données sont utilisées. Par l'utilisation des ontologies, les relations sémantiques entre les composants d'un modèle CAO sont explicitement représentées afin d'améliorer l'interopérabilité entre les systèmes CAO à base de features.

Afin d'atteindre notre objectif, nous avons proposé une méthodologie d'échange des données des modèles CAO en se basant sur le développement d'une ontologie commune de features de conception, servant d'intermédiaire entre les différentes applications de CAO. Cette méthodologie se base principalement sur les quatre étapes suivantes :

1) La première étape est l'homogénéisation des formats de données. En effet, les différents modèles CAO sont hétérogènes à plusieurs niveaux, y compris au niveau syntaxique. Cette étape consiste à traiter l'hétérogénéité syntaxique par la conversion de différents formats d'origine en un format pivot, en l'occurrence le langage d'ontologie OWL DL. Ceci permet de faciliter l'intégration sémantique des différentes ontologies, réalisée dans une phase ultérieure. Les ontologies générées lors de cette étape sont appelées « *ontologies d'application* » correspondant chacune au domaine et à la terminologie spécifiés dans chaque application. La syntaxe des ontologies

générées est décrite en OWL DL, et ainsi peut être transportée par le web ce qui facilite son partage.

2) La deuxième étape consiste à développer une ontologie commune de features de conception, que nous appelons CDFO « *Common Design Features Ontology* ». Elle sert de format neutre pour l'échange des données avec les ontologies d'application résultant de la première étape. Cette ontologie est décrite en OW DL et enrichie par la création d'axiomes et de règles permettant de représenter formellement la sémantique associée aux données de produit.
3) La troisième étape consiste à définir une méthode d'intégration sémantique des ontologies générées lors des deux étapes précédentes. Il s'agit de mettre en place un processus de mise en correspondance entre notre ontologie commune d'une part, et les ontologies d'application d'autre part. Durant cette étape, nous mettons en œuvre, dans un premier temps, un processus d'axiomatisation permettant la création d'axiomes et de règles de correspondance entre différentes entités. Ce processus d'axiomatisation repose principalement sur les constructeurs OWL DL afin d'établir des relations de correspondance, telles que la subsomption ou l'équivalence. Ensuite, nous faisons appel aux services d'inférence permettant la déduction automatique des correspondances supplémentaires entre les entités des différentes ontologies. Cette étape s'appuie essentiellement sur le mécanisme de raisonnement basé sur les logiques de description (LD) et effectué par des services fournis par des moteurs d'inférence tels que Pellet et Jess.
4) Enfin, nous enrichissons notre approche par le développement d'une méthode de recherche de similarité en cas où les équivalences sémantiques n'aient pas été détectées. Notre méthode de calcul tire parti de l'expressivité riche du langage OWL DL pour le calcul de similarité entre les entités, tout en considérant les informations incluses dans la définition de ces entités.

Ce travail de thèse a été mené dans le cadre d'une Convention Industrielle de Formation par la Recherche (CIFRE) au sein de l'équipe SOC[1], *Service Oriented Computing*, du LIRIS[2] (Laboratoire d'InfoRmatique en Image et Systèmes d'information) de Lyon. Le partenaire industriel de la convention CIFRE est la société DATAKIT SARL[3], éditeur de logiciels de conversion de modèles CFAO.

[1] http://liris.cnrs.fr/~soc/doku.php
[2] http://liris.cnrs.fr/
[3] http://www.datakit.com

Organisation de la thèse

Dans le premier chapitre, nous allons effectuer une étude de l'état de l'art sur les modèles de conception d'un produit, notamment les modèles à base de features. Nous abordons ce chapitre par une révision générique sur le domaine de la conception et sur l'évolution des modeleurs de CAO jusqu'à la modélisation par features. Nous effectuons ensuite une étude détaillée sur le concept feature largement utilisé dans la CAO. Ce concept constitue l'élément de base des données à échanger dans notre travail. C'est ainsi que le besoin d'introduction des features, notamment les features de conception, est expliqué par rapport aux entités géométriques. Nous étudions également les différentes définitions des features et leur classification dans certaines taxonomies proposées dans la littérature. Nous étudions ensuite les différentes techniques de création des features, notamment la reconnaissance des features à partir d'un modèle géométrique et la construction d'un modèle de produit à base de feature par l'utilisation d'une librairie de features associée au système de modélisation.

Dans le deuxième chapitre, nous nous intéressons aux méthodes d'échange des modèles de produit, avec un intérêt particulier à l'échange des modèles procéduraux, basés sur l'historique de construction. Les mécanismes d'échange sont basés sur une approche directe ou indirecte par l'utilisation d'un format neutre. Avec l'échange indirect, différents standards ont été développés pour l'échange des données d'un produit. Le plus important et le plus complet est le projet international pour l'échange de données d'un produit, ISO 10303, connu sous le nom STEP. Nous détaillons, dans ce chapitre les différentes implémentations de STEP liées à l'échange des features, notamment les parties 48 et 224. De plus, nous présentons quelques ressources de STEP qui ont été proposées pour traiter l'intention de conception, notamment l'historique de construction (partie 55), les paramètres et les contraintes (partie 108), et les features (partie 111). Nous présentons enfin d'autres projets et efforts qui ont été effectués pour l'interopérabilité des systèmes CAO.

Dans le troisième chapitre, nous proposons une approche d'échange, basée sur la méthode indirecte, tout en considérant la sémantique associée aux données d'un produit. Notre méthode se base sur l'utilisation des technologies du Web Sémantique, telles que les ontologies. Ainsi, nous présentons une étude sur ces technologies dans la première partie de ce chapitre. Ensuite nous décrivons notre approche d'échange qui repose principalement sur le développement d'une ontologie commune, CDFO, servant d'intermédiaire entre les

différentes applications de CAO. Le développement de cette ontologie sera détaillé dans le chapitre 4. Dans ce chapitre, nous présentons la méthodologie suivie pour la construction de notre ontologie. Ensuite, nous décrivons les représentations graphique et formelle de notre ontologie commune.

Dans le chapitre 5, Nous procédons à la définition de notre méthode d'intégration sémantique des ontologies. Cette méthode est basée sur l'alignement binaire des ontologies par la mise en correspondance de différentes entités. Cette méthode d'intégration se base, d'une part, sur la définition des axiomes et des règles de correspondance et d'autre part sur la reconnaissance automatique des correspondances supplémentaires à l'aide des services d'inférence. Ensuite, nous étendons cette méthode d'intégration par une méthode de recherche de similarité sémantique des entités de différentes ontologies. Nous décrivons le développement de cette méthode de recherche de similarité dans le chapitre 6. Cette méthode est adaptée au langage OWL DL afin d'exploiter l'expressivité du langage pour le calcul de similarité. Enfin, nous développons, dans le 7ème chapitre, une application visant à faciliter l'intégration des deux ontologies de CAO en se basant sur les inférences des correspondances effectuées par les raisonneurs. Le prototype que nous avons élaboré est implémenté en langage Java. Nous allons enfin conclure ce mémoire par un bilan des travaux réalisés, les contributions effectuées et les perspectives de travail à réaliser.

Chapitre 1 Modélisation CAO à base de « Features »

1.1 Introduction

Dans ce premier chapitre, nous nous intéressons à l'étude de différents modèles de conception, et plus spécifiquement les modèles de CAO à base de features, contenant des informations sémantiques associées aux données géométriques. Dans ce contexte, nous allons débuter ce chapitre par une présentation du domaine de conception ainsi que l'évolution des modeleurs CAO jusqu'à la modélisation par features. Cette présentation concerne les motifs qui ont abouti à la création des features dans la modélisation et leurs avantages par rapport aux entités de base telles que la géométrie et la topologie d'une pièce. Ensuite, nous nous focalisons sur le concept « feature » utilisé dans les différentes applications de CFAO. Le terme feature est largement utilisé dans le domaine de modélisation avec différentes significations selon leur domaine d'utilisation. Ainsi, nous présentons dans la section 1.3 les différentes définitions qui ont été proposées pour ce terme et les différents types définis dans la littérature. Les travaux de recherche sur les features ont abouti à différentes classifications que nous présentons dans la section 1.4. D'autre part, les features peuvent exister dans un modèle selon différentes techniques : la modélisation à base de features et la reconnaissance de features à partir d'un modèle géométrique. Ces méthodes de création sont détaillées dans la section 1.6.

1.2 Le domaine de la conception

La conception est une activité innovante qui implique de nouvelles techniques et de nouveaux besoins pour des produits (Smith, et al., 1995). Elle peut être décrite comme le processus qui transforme un ensemble de spécifications fonctionnelles et de besoins en une description physique d'un produit (ou d'un système) qui rencontre ces spécifications et ces besoins (Salomons, 2000). La conception assistée par ordinateur (CAO) désignait initialement les outils logiciels qui permettaient à l'ordinateur de remplacer la planche à dessin.

Effectivement, dans les années 70, les logiciels de CAO n'étaient que des logiciels de DAO (Dessin Assisté par Ordinateur). Ils ont évolué petit à petit grâce, d'une part à l'augmentation des performances du matériel informatique et d'autre part à la recherche dans le domaine du logiciel (Maculet, et al., 2003). De nos jours, la CAO comprend l'ensemble des logiciels et des techniques de modélisation géométrique permettant de concevoir, de tester virtuellement - à l'aide d'un ordinateur et des techniques de simulation numérique - et de réaliser des produits manufacturés et les outils pour les fabriquer. Les logiciels d'aujourd'hui peuvent, en incorporant des règles-métiers, capturer efficacement l'intention de l'utilisateur, et allègent ainsi sa tâche bien au-delà de ce que pouvait faire la planche à dessin.

1.2.1 Processus de Conception

Un produit fabriqué peut appartenir à différents domaines tels que la mécanique, l'électronique, le bâtiment *etc*. La complexité du produit à fabriquer implique le besoin d'une méthodologie où les objets intermédiaires de conception, en particulier les plans, sont indispensables. On assiste ainsi à une rationalisation de la production qui permet de faire baisser le temps et le coût de la construction du produit. Depuis les années 1960, plusieurs modèles de processus de conception ont été développés comprenant des étapes qui permettent le développement du produit final. Les différences entre les modèles du processus de conception d'un produit sont identifiées à partir des activités intégrées dans chaque phase de conception. Cependant, les contenus des modèles sont plus au moins semblables. *Pahl et Beitz* proposent un ensemble exhaustif de tâches génériques de conception (Pahl, et al., 1996). Cet ensemble, illustré dans la figure 2, est décomposé en quatre grandes phases, chacune correspondant à un niveau de concrétisation du produit (Daniel, et al., 1997), (Dartigues, 2003), (Ghodous, 1996), (Pahl, et al., 1996):

- Cahier des charges (*specifications*) : C'est l'étape initiale du processus de développement d'un produit appelée aussi phase fonctionnelle. Elle consiste à élaborer le cahier des charges fonctionnel (*CDCF*) pour un produit mécanique. Le rôle du concepteur est de récolter et analyser les besoins des clients, puis à identifier les différentes contraintes à satisfaire durant le processus du développement de la conception. Le résultat est une spécification initiale du produit exprimée sous forme d'une liste de fonctions et de caractéristiques du produit, des contraintes, etc.

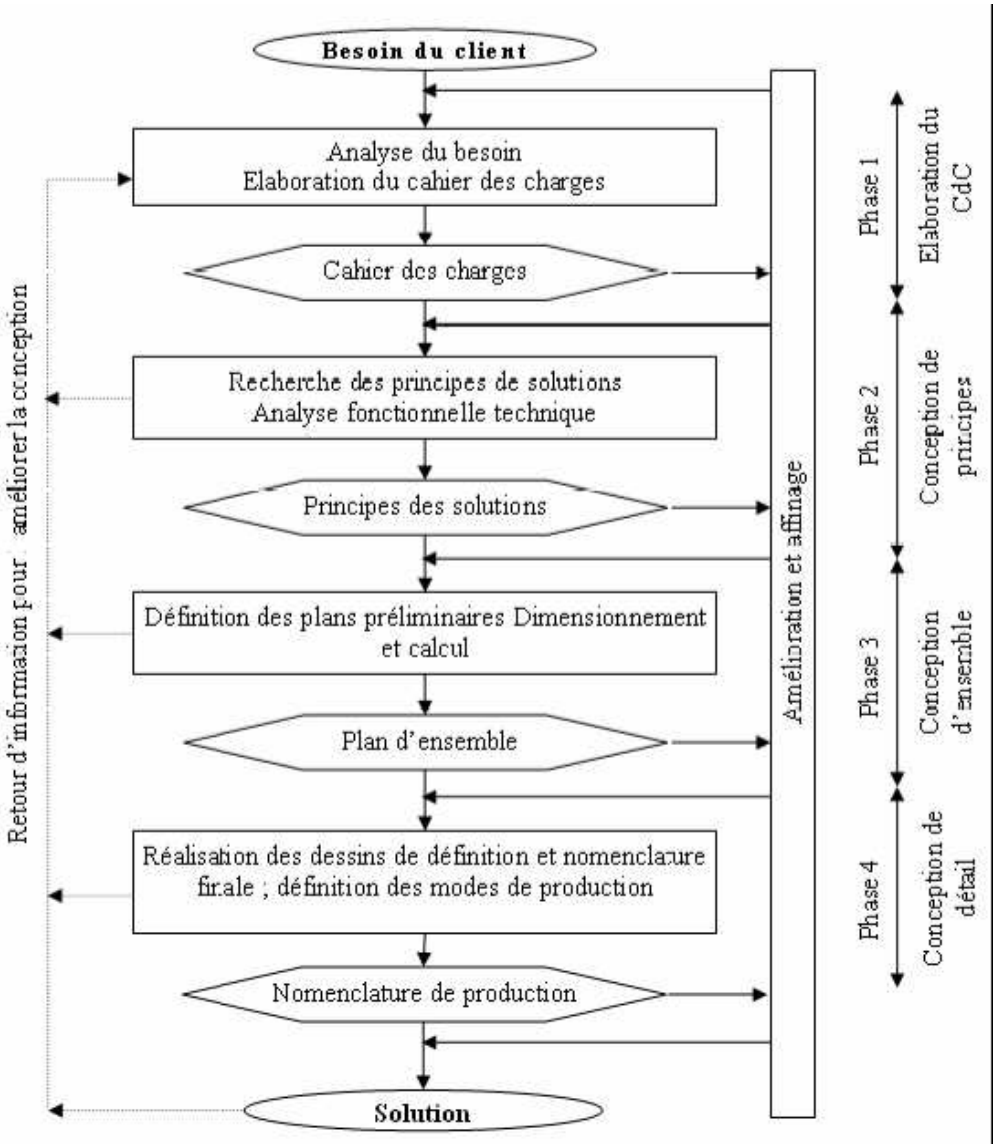


Figure 2. Les phases du processus de conception (Pahl, et al., 1996)

- Conception conceptuelle ou de principe (*conceptual design*) : C'est l'étape de formalisation et de spécification des principes. Elle permet d'affiner le contenu de la première phase c'est-à-dire à structurer, hiérarchiser, et caractériser les différents besoins fonctionnels exprimés. Ainsi, elle correspond à la génération d'idées qui sont confrontées aux besoins définis par le client dans le cahier des charges. À ce stade, les informations manipulées ne permettent pas de concevoir la forme des produits. Durant cette phase, le concepteur cherche et évalue les principes physiques qui correspondent aux besoins fonctionnels identifiés auparavant. D'ailleurs, Le développement de cette phase peut permettre une approche plus globale du processus de conception.

- Conception préliminaire ou conception d'ensemble (*preliminary design*) : C'est l'étape de conception d'ensemble qui débouche sur un Avant-Projet Sommaire (APS) avec des plans à petite échelle. C'est l'activité qui conduit, à partir de la spécification des besoins liés à un produit, à la rédaction d'un ensemble de documents qui décrivent la fonction, le comportement, la forme, la complexité structurelle ainsi que les matériaux de l'objet à concevoir (Dartigues, 2003). Durant cette phase, les concepts associés au produit sont développés par la description technique complète de la structure finale du produit. La solution produite sera raffinée concrètement par des représentations et des vues géométriques d'ensemble de plus en plus évoluées, par une nomenclature de composants et par une évaluation des coûts. Lors de cette étape, les principaux choix technologiques sont effectués, et à l'issue de cette phase un prototype partiel ou total du produit peut être réalisé.
- Conception détaillée (*detailed design*) : Cette phase finale du processus de conception est bien couverte par les logiciels de CAO actuels et se termine par des plans à grande échelle. Elle décrit en détail chacun des points énumérés lors de la phase de conception préliminaire. Le concepteur définit complètement et en détail chaque composant sélectionné et validé en spécifiant ses dimensions, ses caractéristiques physiques (matériaux), ses schémas et plans détaillés, son coût et une description de son processus d'industrialisation (fabrication, assemblage, distribution). Plus précisément, la structure et la forme du produit sont complètement définies. La forme est notamment exprimée à l'aide d'entités géométriques de haut niveau appelées caractéristiques, qui permettent d'associer à un ensemble d'entités géométriques une ou plusieurs fonctions. Cette étape permet également de déterminer les tolérances associées au produit, les processus utilisés pour concevoir le produit ainsi que les coûts de fabrication des produits (Feng, et al., 1996). Dans notre étude, nous nous focaliserons sur cette dernière phase : la conception détaillée.

Au terme de tout processus de conception, il est souvent question d'une phase de vérification et de validation des résultats. Elle permet en effet de vérifier si le produit répond bien aux spécifications du cahier des charges. Comme illustré dans la figure 2, l'aspect itératif et dynamique du processus de conception montre que le déroulement des différentes phases n'est pas séquentiel. La conception est une activité fortement itérative. Par exemple, même à

un stade avancé du processus de conception, une simple modification du besoin client peut remettre en cause certains choix et calculs déjà validés pendant la phase de spécification des principes (Pahl, et al., 1996).

1.2.2 Evolution des modeleurs CAO

Le composant essentiel d'un logiciel de CAO est le modeleur géométrique. Son vocabulaire est celui des entités géométriques (points, droites, courbes, plans, surfaces,…) et des entités topologiques (sommets, arêtes, faces). En CAO, le modèle est la représentation informatique de l'objet en cours de conception. C'est la maquette numérique, virtuelle de l'objet. Aujourd'hui, le modèle géométrique est le cœur d'un système de CAO.

Les techniques de modélisation géométrique, largement évoluées, sont capables de représenter des objets très complexes. Pourtant, les informations géométriques restent insuffisantes pour saisir la sémantique d'un produit, où l'information fonctionnelle d'un produit est parfois nécessaire[4]. Ainsi, la tendance d'évolution des systèmes CAO est d'intégrer au modèle purement géométrique d'autres informations non géométriques et à des niveaux différents (par exemple les contraintes d'ingénierie, les tolérances, matériau, *etc.*). Des entités technologiques de haut niveau (features) ont été donc introduites dans la majorité des systèmes actuels de CAO.

Les besoins d'utilisation des features et leurs avantages par rapport aux modèles purement géométriques sont présentés dans (Shah, et al., 1995). D'abord, la construction de modèles géométriques peut être fastidieuse. Les primitives de modélisation sont souvent de bas niveau (points, lignes et plans) et ne correspondent pas à la façon de penser du concepteur. De plus, les modèles géométriques représentent seulement une instance idéale du produit, et n'intègrent pas facilement les tolérances, qui sont essentielles pour la fabrication afin de s'assurer qu'une pièce soit fonctionnelle, fabricable et assemblable à un coût raisonnable. Les tolérances peuvent être appliquées sur la taille, la dimension, l'orientation, la position, ou la forme. Enfin, les modèles géométriques ne stockent pas l'intention de conception[5].

[4] Information fonctionnelle : peut être la fonction de l'utilisateur d'une partie d'un produit, ou l'information à propos de la manière dont elle est fabriquée ou assemblée.

[5] Ils ne représentent pas pourquoi la géométrie est telle qu'elle est. Cette information est nécessaire, par exemple, pour rendre la ré-conception effective.

1.2.3 Modélisation par features

Avec l'augmentation des capacités de calcul et de stockage des systèmes informatiques, les systèmes actuels de modélisation par features permettent d'incorporer autour des modèles géométriques de plus en plus de connaissances non géométriques, liées au produit lui-même et à sa fabrication. L'avantage de la modélisation par features par rapport à la modélisation géométrique conventionnelle est sa capacité à associer des informations fonctionnelles et d'ingénierie aux informations concernant la forme dans un modèle de produit (Bidarra, et al., 2000). L'utilisation des features facilite également l'intégration entre différentes phases du développement de produit par exemple la conception et la fabrication (Ghodous, 1996).

Dans les systèmes CAO à base de features, en plus de la description B-REP de l'objet, le modeleur maintient au cours de la réalisation de l'objet un arbre de spécifications qui retrace la suite des features employés. L'un des avantages est la possibilité de revenir en arrière (*undo*), au prix d'une gestion délicate de l'édition du modèle et de ses mises à jour. Ces systèmes facilitent donc la mise à jour du modèle après une modification d'une partie du modèle ainsi que l'intégration de l'intention du concepteur. La conception d'un produit est un raffinement continuel, c'est une boucle de conception/re-conception (Maculet, et al., 2003). Ainsi, l'utilisation des features comme des primitives de conception améliore l'efficacité de création d'un modèle et réduit le temps nécessaire pour les mises à jour appliquées à ce modèle (Fontana, et al., 1999).

La modélisation de produit par features pourra surmonter les problèmes provenant de la modélisation géométrique. Les features sont des objets paramétrés qui représentent l'intention du concepteur (Shah, et al., 1995) et dont l'échange permet de maintenir cette intention durant les différentes phases de développement de produit. L'intention de conception est un terme très répandu dans le domaine de conception par features. Elle implique la prise en compte et l'explicitation de la sémantique intégrée dans le modèle du produit durant son cycle de vie. (Anderson, et al., 1998) ont défini l'intention de conception par « des exigences fonctionnelles fournies par les clients, consistant un ensemble de règles géométriques et techniques que le produit final doit satisfaire ». Elle désigne la représentation de l'historique de construction du modèle, ainsi que les features, les paramètres, et les contraintes intégrés dans le modèle (Pratt, et al., 2005). Le paramétrage fournit au concepteur

plus de liberté lors de la construction de son modèle. Par contre, des contraintes peuvent être imposées, restreignant cette liberté afin d'assurer la continuité de sa fonctionnalité suite à une modification éventuelle du modèle (Pratt, et al., 2001).

1.2.4 Exemple de vue méthodologique de conception

Considérons l'exemple du modèle d'un produit illustré dans la figure 3. Pour un modeleur géométrique, l'objet visualisé possède la description géométrique suivante :

- 1 Corps solide
- 48 faces
- 86 arêtes
- 40 sommets
- 47 surfaces (dont 13 plans, 20 cylindres, 8 sphères, 6 torus)
- 86 courbes (dont 40 lignes, 46 cercles)

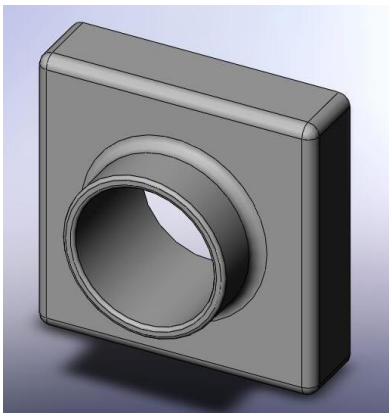


Figure 3. Exemple d'un modèle 3D construit avec le logiciel SolidWorks

Or, le modeleur géométrique complète cette description géométrique en fournissant les paramètres des entités géométriques et topologiques figurant dans le modèle 3D. Ceci comprend la définition des coordonnées de chaque sommet, la spécification des sommets pour chaque arête, et l'indication des arêtes qui délimitent chaque face ainsi que sa normale.

Avec l'approche par features, les objets géométriques sont plus complexes : ils sont formés par un assemblage d'éléments simples, par exemple une rainure dans une pièce mécanique. Chaque feature, un congé ou un chanfrein par exemple, résume une longue suite de commandes et d'opérations à effectuer par le modeleur et la visualisation, et contient des

règles en usage dans un métier. Par conséquent, dans un modeleur à base de features, le concepteur définit l'exemple du modèle précédent comme suit :

- Un pavé rectangulaire
- Un pavé circulaire
- Une poche
- Deux congés
- Une coque

L'historique de construction du modèle est représenté dans la figure 4.

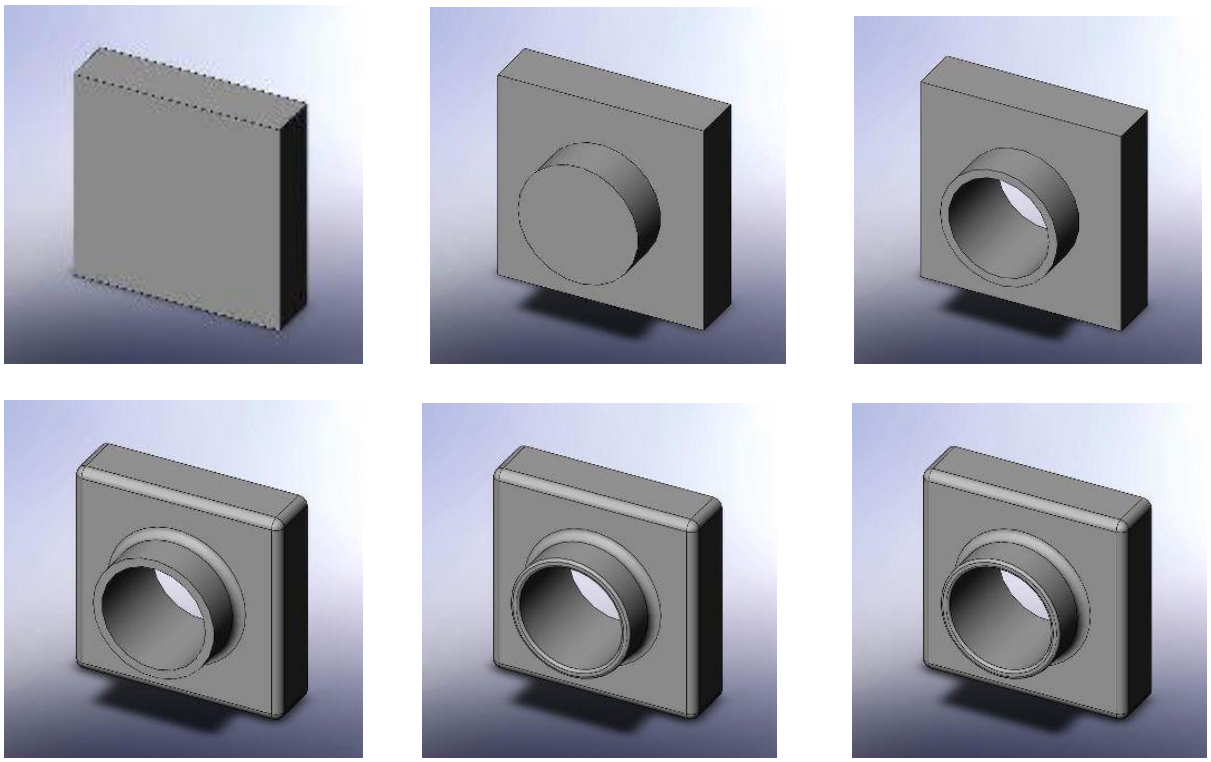


Figure 4. Evolution du modèle 3D durant la conception par features

Chaque étape représente l'ajout d'un feature de conception au modèle 3D. Pour compléter la description, il suffit de définir les paramètres de chaque feature, par exemple pour une extrusion ou une poche il faut définir son esquisse de base, sa direction, sa profondeur, *etc*. L'extrusion ajoute de la matière tandis que la poche en enlève. Pour chaque congé, il faut définir son rayon, les objets à tailler, *etc*. La forme géométrique finale de la pièce résulte de l'application d'une séquence de features paramétrés et intégrés dans le modèle 3D.

Cette description implicite est plus économique et accessible à un non mathématicien puisqu'elle comporte de l'information sémantique. Les features intégrés (tels que extrusion, chanfrein, trou débouchant, *etc*.) suffisent pour décrire une série de fonctions de modélisation de haut niveau. Des règles en usage dans les métiers peuvent être intégrés, par exemple ne pas percer un trou près du bord.

D'ailleurs, le terme feature est largement utilisé dans les systèmes actuels de développement de produit, où différentes interprétations ont été fournies selon le domaine d'utilisation. Dans la section suivante, nous allons présenter les différentes définitions qui ont été proposées pour les features ainsi que les différents types de features qui ont été élaborés, en particulier les features de forme pour la géométrie.

1.3 Définitions des features

Le concept feature a été interprété différemment par les chercheurs, et aucun consensus n'a été élaboré pour une définition conventionnelle de ce terme. Ainsi, un grand nombre de définitions a été donné pour les features. Certaines de ces définitions sont spécifiques aux domaines d'application (conception, gamme d'usinage, fabrication, *etc.*), et d'autres sont, en revanche, plus générales. D'un point de vue général, les features sont définis dans (Smith, et al., 1995) comme des primitives de conception de haut niveau avec leurs attributs, leurs qualificatifs, et les restrictions qui affectent la fonctionnalité et/ou la capacité pour le produit d'être fabriqué. Ils peuvent décrire la forme (taille et dimensions), la précision (tolérances et finition), ou le matériau (type, propriétés et traitement) et varient avec le produit et le processus de fabrication. Une définition plus générale des features est donnée dans (Wilson, et al., 1988): c'est une région d'intérêt dans un modèle de produit.

Selon leur contexte d'application, les features ne désignent pas le même concept. Dans le domaine de la conception, les features sont des éléments utilisés dans la génération, analyse, et évaluation du modèle d'un produit (Shah, et al., 1991). D'ailleurs, ils ont été introduits dans l'ingénierie mécanique comme étant des éléments essentiels pour associer une signification fonctionnelle spécifique à des groupes d'éléments géométriques (faces, arêtes, sommets), offrant l'avantage de traiter des ensembles d'éléments comme des entités uniques (Shah, et al., 1995). Ces entités sont plus significatives, pour les applications, que des éléments primitifs de bas niveau, et sont manipulées par des paramètres. Elles sont destinées à

réaliser une fonction donnée, ou à modifier la forme d'une pièce (Wang, et al., 1990). Certaines autres définitions d'un feature mettent l'accent sur l'information fonctionnelle liée à l'ingénierie, ce qui rend les features d'un niveau sémantique plus élevé que les entités géométriques.

Dans le domaine de la fabrication, (Salomons, 2000) définit les features comme un regroupement sémantique utilisé pour décrire une pièce et ses assemblages qui rassemblent d'une manière significative l'information fonctionnelle de conception et de fabrication. Une autre définition similaire est donnée dans (Shah, 1990) : un feature est un porteur de l'information de produit qui peut aider la communication entre la conception et la fabrication ou entre les autres activités de l'ingénierie. D'une manière plus générale, un feature concerne des données de produit, reliées à des spécifications fonctionnelles, aux processus de fabrication, ou à des propriétés physiques de conception.

Par ailleurs, la majorité des définitions considèrent le feature comme une association d'une forme géométrique avec des informations annexes nécessaires pour le développement de produit. Il représente une partie physique d'un objet correspondant à une forme générique et ayant une signification fonctionnelle. C'est un modèle récurrent de l'information liée à la description d'une pièce[6] (Salomons, 2000). Une définition plus précise considère un feature comme étant une représentation des aspects de forme d'un produit qui sont correspondants à des formes génériques, et qui ont une signification fonctionnelle pour une activité particulière du cycle de vie d'un produit (Bidarra, et al., 2000).

Dans le domaine de conception et de fabrication, parler de features avait la même signification que parler de features de forme. Dans ce contexte, un feature est défini comme une configuration géométrique spécifique formée sur la face, l'arête ou le sommet d'une pièce, par exemple poche, trou, *etc.* (Wingard, 1991). Ils constituent des éléments conformes à des règles permettant leur reconnaissance et leur classification (Pratt, et al., 1991). C'est cette définition des features que nous retiendrons pour la suite de notre travail. Nous considérons plus spécifiquement les features de forme, c'est à dire des entités de haut niveau qui définissent une forme précise sur une pièce. Ils sont définis comme des instances d'une forme

[6] Cela signifie qu'un feature permet notamment de définir la forme générale et les attributs d'un trou (diamètre, profondeur, etc.) et que la création d'une instance d'un trou se fait en fonction de ce modèle ;

générique, incorporant des attributs géométriques et non géométriques et des contraintes. De plus, ces entités portent une signification sémantique aux utilisateurs.

Les features sont donc des entités plus complexes que des simples éléments géométriques. Ainsi, on peut conclure que le nombre de différents types de features n'est pas limité. Dans la CFAO, différents types de features ont été créés afin de répondre à des besoins bien spécifiques. Cette diversité de features nécessite la classification des features selon leur type, et leur domaine d'application. Cette classification sera prise en considération dans la sous section suivante.

1.3.1 Types de feature

Le concept feature est utilisé dans la modélisation pour désigner une variété de caractéristiques d'un produit. Chaque groupe de recherche a défini sa propre classification de features. Un feature de forme réfère à une forme construite liée à une fonction spécifique où à l'usinage. Il peut être de nature additive (bossage) ou soustractive (dépression) de la matière. Un feature d'usinage est donc un feature de forme soustractive associé à un processus d'usinage distinct (Han, 1996).

Les features sont classifiés dans (De Kraker, 1998) selon deux critères orthogonaux : l'application, dans laquelle les features sont utilisés, et le niveau d'abstraction des features. Les features sont reliés à leur domaine d'application. Par exemple, les features de conception représentent la fonctionnalité intentionnelle de produit, les features de fabrication représentent l'information du processus de fabrication, et les features d'assemblage représentent l'information de manipulation et de connexion. Le niveau d'abstraction d'un feature est défini comme l'extension à laquelle le feature représente la géométrie concrète. Par exemple, un feature de conception conceptuelle (*conceptual design*) qui représente les concepts ou les fonctions, est d'un niveau d'abstraction plus élevé que celui, par exemple, d'un feature rond sur une arête qui représente la géométrie actuelle.

Par ailleurs, plusieurs catégories de features ont été identifiées dans la littérature (Shah, et al., 1995): features de forme, features de contraintes, features de tolérance ou de précision, features d'assemblage, features fonctionnels, et features de matériau. Les trois types : forme, tolérance, et assemblage, sont liés à la géométrie de la pièce, ainsi ils sont appelés « les features géométriques ». La plupart des systèmes CAO actuels se rapportent

principalement aux features de géométrie, particulièrement les features de forme et certains types de features d'assemblage (Shah, et al., 1995). Nous nous intéressons dans notre étude aux features de forme.

- **Features de forme** : Les features de forme sont des entités dont l'aspect physique est mis en évidence. La notion de feature de forme permettra de saisir l'intention de conception par l'association d'une signification d'ingénierie à la forme. Néanmoins, l'extraction de l'intention pourra être fastidieuse parce que cette signification d'ingénierie est souvent implicite dans un feature (Salomons, et al., 1993). Selon les deux critères de classification, application et abstraction, discutés dans (De Kraker, 1998), et présentés ci-dessus, les features de forme ont un niveau d'abstraction moins spécifique à une application, et définis dans la plupart des applications. Le niveau d'abstraction modéré des features de forme provient du fait qu'ils ont une signification fonctionnelle et une forme paramétrique prédéfinie.
- **Features de contraintes** : Ils représentent des contraintes spécifiant certaines propriétés des features de forme. Les contraintes les plus courantes sont les contraintes géométriques et algébriques. Les premières spécifient les relations géométriques comme le parallélisme et la distance. Les dernières définissent les relations entre les dimensions sous forme d'équations. Un type de contraintes moins courant, spécifiant des propriétés importantes des features, correspond aux contraintes sémantiques. Ces contraintes précisent l'extension à laquelle une face de feature doit appartenir. Elles sont exprimées comme des prédicats logiques qui peuvent être vérifiés. Dans un modèle défini en termes de séquence d'opérations de construction, les contraintes peuvent être de deux types: implicite et explicite (Pratt, et al., 2005). Les contraintes implicites proviennent de l'exécution de certaines opérations. Les contraintes explicites sont représentées dans le modèle, où elles sont appliquées aux éléments explicites de construction inclus dans l'historique de construction.
- **Features de précision** (ou de tolérance) : La modélisation géométrique des solides est une méthode significative pour définir une pièce ayant des dimensions précises. Ainsi, des informations supplémentaires sont parfois nécessaires pour définir les variations susceptibles pour une pièce. Ainsi, des tolérances de dimensions peuvent être définies sur la taille, la forme, l'orientation, et la position.

- **Features d'assemblage** : Ils regroupent des features nécessaires pour la définition des relations d'assemblage, par exemple les conditions de regroupement (*mating conditions*), les positions relatives des composants, ainsi que leur orientation.
- **Features fonctionnels** : Ce sont les ensembles de features liés à des fonctions spécifiques. Ils peuvent inclure des informations sur l'intention de conception, des paramètres non géométriques liés à une fonction ou à la performance, etc.
- **Features de matériau** : Ils sont spécifiques aux compositions matérielles de la pièce.
- **Features de primitive** : Un feature de primitive est une entité géométrique de base d'une pièce, comme les surfaces, les arêtes, et les sommets ou des attributs géométriques auxiliaires d'une partie comme des centres des lignes, ou des plans.

1.4 Taxonomies de features

Bien que le nombre des features soit indéfini, leur classification en groupes ayant des propriétés communes est un sujet important. Une taxonomie de features représente une structure hiérarchique de regroupement des features selon des caractéristiques communes. Le principal avantage d'une taxonomie de feature est la structuration utilisée pour classifier les features. Un autre avantage est la notion d'héritage : les propriétés des classes parents peuvent être transmises ou héritées par leurs sous-classes sans qu'elles soient explicitement répétées (Salomons, et al., 1993). Ainsi, une représentation plus compacte de la connaissance peut être effectuée. De plus, la représentation des features dans une taxonomie permettra de les identifier dans un modèle d'échange de données d'un produit (Fu, et al., 2003).

Aucun consensus n'a été élaboré pour une définition d'une taxonomie de feature. Par conséquent, différents critères ou standards peuvent mener à différentes taxonomies de features. Ces dernières peuvent être basées sur les catégories du produit (features de tôlerie, d'usinage,…), leurs applications (conception, gamme d'usinage,…) et leurs formes (prismatique, rotation, *etc.*). Ainsi, différents schémas de taxonomie ont été proposés en se basant sur la forme plutôt que sur l'application (Shah, et al., 1995).

Certaines de ces classifications ont été données pour des domaines particuliers. Dans le domaine d'usinage, une classification, proposée par (Gindy, 1989), définit trois catégories de features de forme : *protrusion*, *depression*, et *surfaces*. *Protrusion* et *depression* désignent

des features volumiques d'ajout et d'enlèvement de la matière respectivement. Dans le domaine de la conception, une classification générique a été présentée dans (Anderson, et al., 1990), d'une façon appropriée spécialement à la modélisation d'un produit, aux features d'assemblage et aux features de tolérance. Comme illustrée dans la figure 5, cette taxonomie définit deux grandes classes de features selon le domaine d'application: les features composites et les features atomiques. Les features composites désignent spécialement les contraintes d'assemblage, « *fixture* », et les features décrivant la cinématique des pièces « *Kinematic* ». Les features atomiques désignent notamment les features de conception d'une pièce.

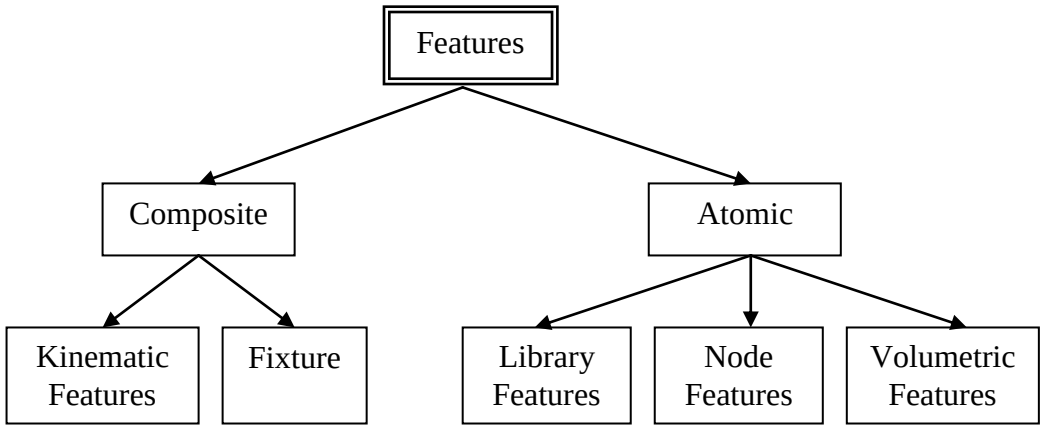


Figure 5. Hiérarchie de classe de features (Anderson, et al., 1990)

Dans PDES (*Product Data Exchange Specification*) (Dunn, 1988), une taxonomie définit les classes de features selon leur forme comme suit :

- *Passages* : ces features comprennent les volumes négatifs qui débouchent les deux extrémités du modèle de la pièce.
- *Depressions* : ce sont les volumes négatifs qui débouchent une des extrémités du modèle de la pièce.
- *Protrusions* : ce sont les volumes positifs qui débouchent une des extrémités du modèle de la pièce.
- *Transitions* : ce sont les régions résultantes du lissage de l'intersection de plusieurs régions.
- *Area* : signifie les éléments 2D définis sur les faces du modèle de la pièce.
- *Deformations* : concernent les opérations de changement de la forme comme la fusion et l'étirement.

(Wingard, 1991) a proposé de concevoir des taxonomies de features de forme en se basant sur leur domaine d'application. Dans cette taxonomie, représentée dans la figure 6, les features sont divisés en deux catégories : atomiques et composés. Les features de forme atomiques « *atomic* » sont eux-mêmes décomposés en trois sous catégories : « *Part form features* », « *Modifier form features* » et « *Grouping form features* ». Les features de la première sous-catégorie peuvent être définis indépendamment de toute forme existante, tandis que les features de la deuxième sous-catégorie dépendent d'une autre forme. La troisième sous-catégorie regroupe un ensemble d'entités contenant une signification d'ingénierie. D'autre part, les features de forme composés « *compound* » sont subdivisés en « *Pattern Form features* » pour les features de répétition, « *Complex Form Features*» pour les features basés sur des features simples, et « *Assembly Form Features* » pour les features d'assemblage.

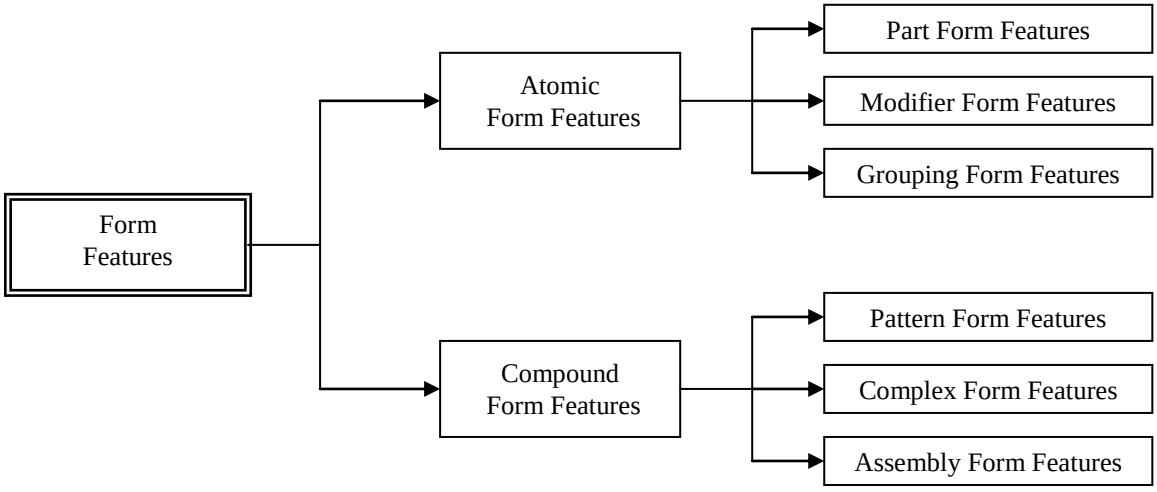


Figure 6. Taxonomie de features de forme proposée par Wingard

Cependant, une classification plus complète est celle qui a été réalisée dans la partie 48 de la norme STEP (ISO 10303-48, 1992), (*Standard Exchange for Product Data Exchange*), une norme pour l'échange de données qui sera étudiée dans le chapitre 2 de ce manuscrit. Ce travail était effectué par un groupe de chercheurs afin de standardiser une ressource pour la modélisation des features de forme. Pourtant, ce travail s'est heurté à des problèmes de généralisation du modèle, et par conséquent ces difficultés ont mené à l'abandon de la finalisation de cette partie de la norme. Dans cette taxonomie, les features de forme sont classifiés selon trois types de base de features : volumétrique, transition, et répétition (pattern). Cette classification est représentée dans la figure 7 :

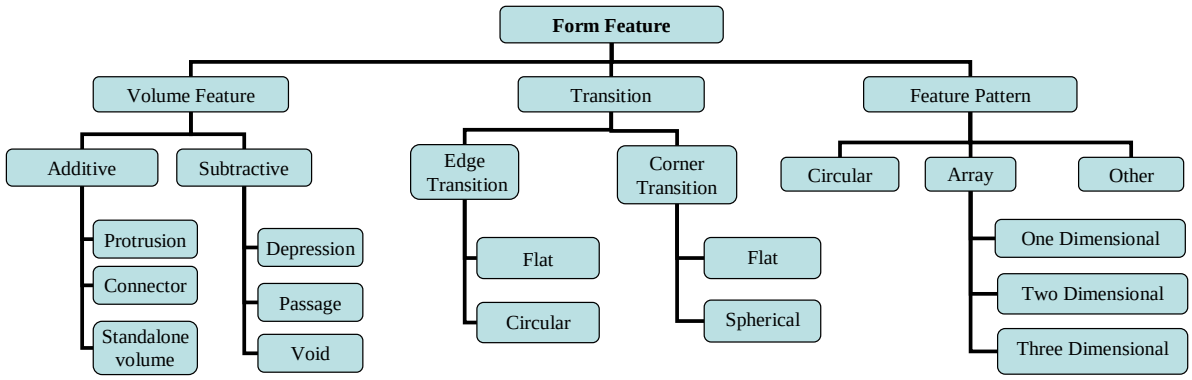


Figure 7. Classification des features de forme dans la partie 48 de STEP

Les features volumétriques (a) correspondent à une approche volumique de la modélisation de produits (trou, poche, extrusion). Ils sont classifiés selon différents critères, par exemple leur forme, leur nature (ajout ou un enlèvement de la matière), *etc*. Les features de transition (b) correspondent à une approche surfacique de la modélisation de produits. Ils spécifient une transition entre les éléments de la topologie (arête, face, *etc.*…), par exemple un congé sur deux arêtes. Les features de répétition (c) : consistent en des features de forme multiples, disjoints, et volumétriques dans une répétition régulière. Elles reproduisent certains features à l'identique selon un schéma bien défini. Des exemples sont des répétitions rectangulaires et circulaires[7].

Une fois les features sont définis et classifiés, leur représentation est un point d'intérêt qu'on doit prendre en considération. Etant des entités complexes, les features intègrent des informations sur la forme, ainsi que d'autres informations supplémentaires qui explicitent sa signification d'ingénierie. Cette complexité exige des techniques de représentation assez puissantes afin de maintenir la sémantique associée. C'est dans la partie suivante que nous récapitulons la représentation des features de forme.

[7] Un arrangement régulier de plusieurs caractéristiques individuelles d'un type commun qui a une certaine signification sémantique d'ingénierie, comme un patron circulaire de trous ;

1.5 Représentation des features

Etant des composants essentiels pour un modèle de conception, les features doivent être bien représentés. Nous nous intéressons, dans cette section, à la représentation des features de forme concernant leur forme et leur signification d'ingénierie.

1.5.1 Représentation de la forme

La plupart des travaux de recherche sur la représentation géométrique des features de forme consentent sur une représentation volumique (3D), en combinaison avec la représentation surfacique. Par exemple, on peut citer la représentation basée sur B-REP (*Boundary Representation*), sur CSG (*Constructive Solid Geometry*), aussi bien que la représentation hybride (Wang, et al., 1991). A part la représentation géométrique, les features peuvent être représentés de différentes manières, par exemple en utilisant des attributs et des règles (Anderson, et al., 1990), des différentes formes de graphes (Billo, et al., 1989), des attributs (Dixon, et al., 1989), et des grammaires de graphe (Fu, et al., 1994).

Deux approches ont été identifiées dans (Wilson, et al., 1988) pour la représentation de la forme des features : implicite et explicite. Dans la représentation explicite, tous les détails géométriques du feature sont entièrement définis. Dans la représentation implicite, certaines informations suffisantes sont fournies pour définir le feature, mais les détails géométriques complets pourront être calculés en cas de nécessité. Par exemple, un trou cylindrique peut être implicitement défini en fonction de son rayon et sa profondeur, ou explicitement en termes d'ensemble de faces qui le composent dans un modèle de représentation B-REP.

1.5.2 Représentation de la signification d'ingénierie

Les méthodes de représentation de la signification d'ingénierie ne sont pas assez développées (Salomons, et al., 1993). Dans l'approche orientée objet, la signification d'ingénierie peut être saisie par des règles et des méthodes attachées à chaque description de classe (Wingard, 1991). Une autre approche est décrite dans (Nielsen, et al., 1991), où les relations géométriques entre les features sont utilisées pour saisir l'intention de conception.

Dans un système de modélisation à base de feature, le feature constitue l'élément de base pour la création d'un modèle de produit. Ces systèmes fournissent des moyens et des facilités de création déterminant l'efficacité et la robustesse d'un tel système. Dans la section suivante, nous nous intéressons à l'étude des techniques de création d'un modèle de produit à base de features.

1.6 Techniques de création de « feature »

Un modèle complet à base de features ne comprend pas uniquement des features, mais également le modèle géométrique résultant. Ce modèle contient la représentation B-REP, CSG, ou d'autres représentations géométriques de l'objet. Le modèle feature regroupe l'ensemble des features intégrés dans le modèle du produit, ainsi que leurs propriétés, et d'autres données de haut niveau (Han, 1996).

Plusieurs techniques ont été conçues pour la création des modèles à base de features. Ces techniques sont divisées en deux approches principales (Allada, et al., 1995), (Shah, et al., 1995):

- La reconnaissance des features où les features sont reconnus à partir d'un modèle géométrique en appliquant des règles de reconnaissance. Par conséquent, plusieurs modèles de features, décrivant le même produit, peuvent être dérivés.
- La conception par features où un modèle de produit est construit en exploitant des features prédéfinis. Dans le cas de la conception par features, la géométrie explicite du produit est créée à partir du modèle de features.

1.6.1 Reconnaissance des features

La reconnaissance des features consiste à dériver un modèle de feature à partir d'un modèle géométrique. Cette reconnaissance peut être réalisée selon deux approches (Allada, et al., 1995) : « *reconnaissance automatique* » où la reconnaissance est effectuée automatiquement par des algorithmes et « *reconnaissance interactive* » qui s'effectue à l'aide de l'intervention humaine.

1.6.1.1 **Reconnaissance automatique**

Cette approche consiste à reconnaître automatiquement les features sur une pièce après sa modélisation dans un modeleur purement géométrique. Cette reconnaissance se réalise principalement par l'exploitation des données géométriques et topologiques du modèle. Typiquement, une configuration (géométrique/topologique) spécifique est recherchée pour détecter la présence d'un type particulier de feature (Allada, et al., 1995). La représentation de cette approche est illustrée dans la figure 8.

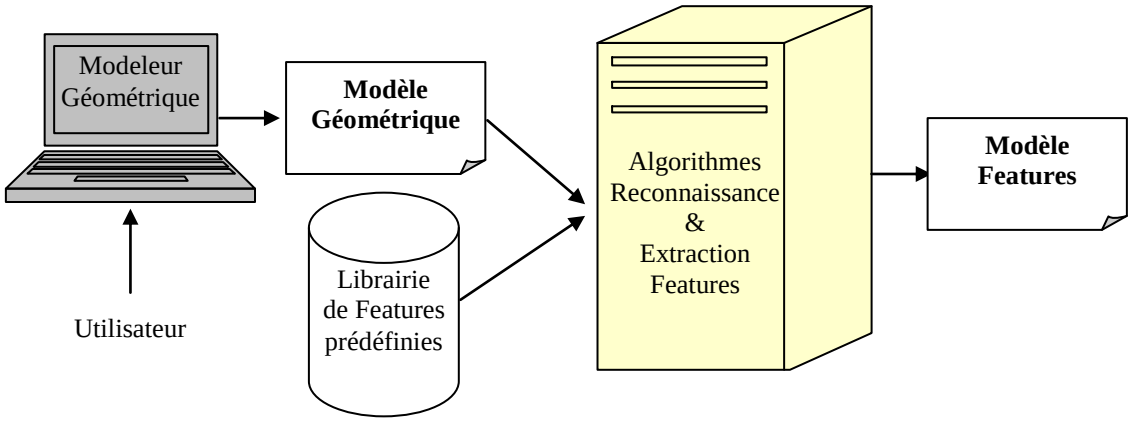


Figure 8. Méthode de reconnaissance automatique des features

1.6.1.2 **Reconnaissance interactive**

Cette approche, représentée dans la figure 9, consiste à réaliser la reconnaissance des features moyennant l'assistance humaine. Ainsi, l'ingénieur crée les features en choisissant des éléments topologiques du produit constituant un feature spécifique dans une image du modèle géométrique. Un avantage de cette approche est qu'elle est assez flexible, car toute combinaison d'entités peut être reconnue comme un feature. Un autre avantage réside dans la facilité d'implémentation de cette approche (Shah, et al., 1995) : seulement les features reliés à une application doivent être définis. Cependant, l'inconvénient principal de son utilisation est que c'est une tâche laborieuse lorsque le nombre de features à identifier est important, où l'ingénieur se charge de la validation des features (De Kraker, 1998). Par conséquent, cette approche est considérée inefficace et incommode (Allada, et al., 1995).

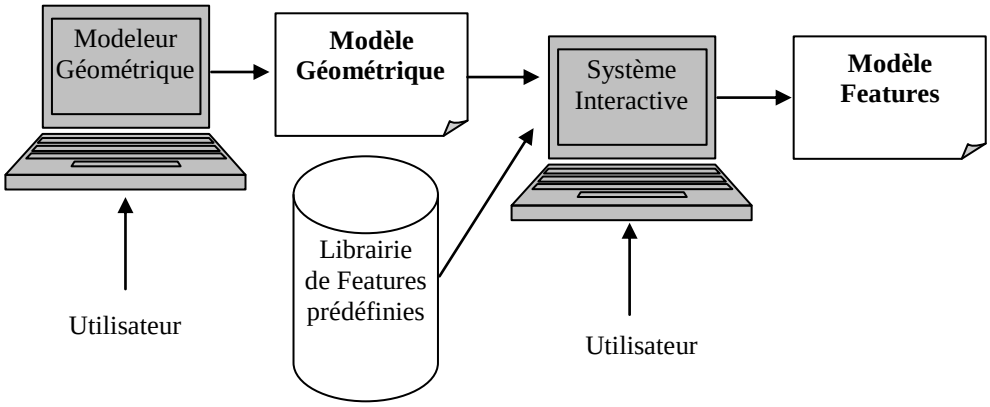


Figure 9. Méthode de reconnaissance interactive des features

La reconnaissance des features, définie dans les deux approches précédentes, peut être spécifique à une application, permettant à chaque application d'avoir sa propre méthode de reconnaissance. Néanmoins, la reconnaissance est considérée comme étant un « effort redondant ». En effet, le défaut de cette approche est que seules les informations géométriques du modèle sont considérées et que les informations techniques ne sont pas disponibles dans la base de données géométriques (Ghodous, 1996).

1.6.2 Conception par features

Cette approche permet à l'utilisateur d'exploiter un ensemble de features défini dans une librairie. Cette librairie permet de substituer des concepts de bas niveau, par exemple les primitives utilisées dans CSG (un bloc ou un cylindre), par des concepts de haut niveau ayant une sémantique déterminée, par exemple un trou ou une poche (Han, 1996). Par conséquent, le modèle résultant est plus facile à éditer grâce à l'utilisation des paramètres.

Dans cette approche, les propriétés des features sont enregistrées durant la phase de conception du modèle de la pièce. Un ingénieur conçoit un modèle en créant des instances des features génériques définis dans une librairie de features. Cette approche présentée dans la figure 10, dispense l'ingénieur de faire appel aux techniques de reconnaissance des features. Cependant, certains inconvénients de cette approche sont définis dans (Allada, et al., 1995):

- L'infinité du nombre de features existants prévient l'intégration de toutes les possibilités de features dans une librairie. Dès lors, la conception par features est restreinte à un domaine d'application bien spécifique.

- La validation de features doit être examinée lors de l'ajout d'un nouveau feature. Cette validation est nécessaire pour s'assurer que le nouveau feature créé est bien positionné, et qu'il n'altère pas la validité des features existants.
- Enfin, cette approche pourra entraver la créativité du concepteur de produit tant que l'ensemble de features définis dans la librairie ne soit pas exhaustive.

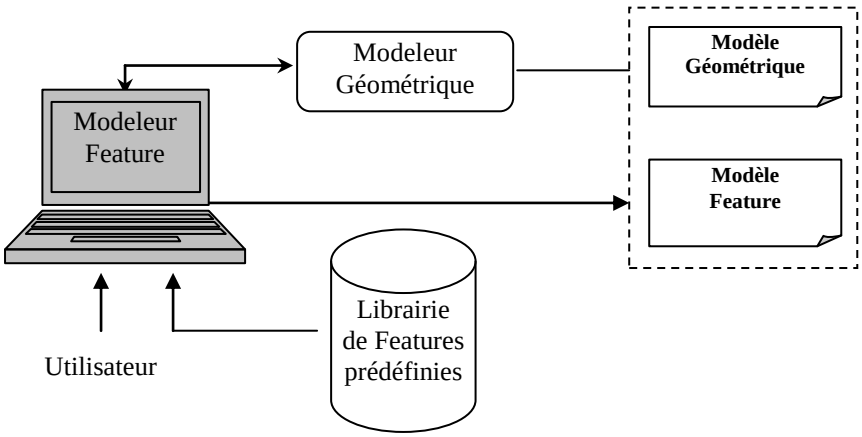


Figure 10. Conception par features

De nos jours, les systèmes basés sur la conception par features connaissent une évolution significative. Ils offrent l'avantage de permettre au concepteur de transférer au modèle les données liées aux features de la pièce, aussi bien que leurs paramètres de dimension, et leurs schémas de tolérances. Autrement dit, ils permettent de stocker l'intention de conception lors de la construction d'un modèle. Par conséquent, la conception devient une tâche plus rapide et plus commode aux besoins des utilisateurs. De plus, des modifications éventuelles sur un modèle de produit peuvent être effectuées plus facilement. Dans la section suivante, nous allons étudier les différentes approches utilisées lors de la conception à base de features. Ceci inclut les deux approches suivantes : procédurale et déclarative.

1.7 Approches de «Conception par features»

Généralement, deux méthodologies pour la conception par features sont utilisées (Shah, et al., 1995) : la « destruction par features d'usinage » et la « synthèse par features de conception ». La première méthodologie consiste à créer le modèle de la pièce à usiner par une suite d'opérations d'enlèvement de la matière appliquées sur le modèle du matériel de base. Cette méthodologie facilite le développement de la gamme d'usinage. Les systèmes basés sur cette méthodologie utilisent un ensemble prédéfini de features d'usinage qui

peuvent être soustraits du solide de base. La deuxième méthodologie permet l'utilisation des features d'addition ou d'enlèvement de la matière afin de concevoir le modèle de la pièce. Dans notre travail, nous consacrons notre étude sur la deuxième méthodologie.

Dans la conception par features, l'utilisateur construit son modèle en utilisant directement des features d'une librairie associée au système, où le modèle géométrique est généré à partir du modèle construit. Les features sont instanciés en spécifiant des paramètres de dimension et de position, des contraintes, des relations avec d'autres features, et certains autres attributs. La manipulation des contraintes varient d'un système à un autre. Deux approches génériques ont été identifiées pour classifier ces manipulations (Shah, et al., 1995): l'approche procédurale et l'approche déclarative. La première est basée sur la séquence d'opérations utilisées dans la construction du modèle de produit, appelée « *historique de construction* », tandis que la deuxième est basée sur les contraintes des paramètres et des features prédéfinis.

1.7.1 Approche Procédurale

Dans l'approche procédurale, le modèle d'un produit est représenté en termes d'une séquence d'opérations utilisées durant sa construction. Cette séquence d'opérations, appelée « historique de construction » ou « historique du modèle », inclut les données concernant les instances des features intégrés, les paramètres ou les dimensions, et les contraintes (Pratt, et al., 2005). Cependant, cette approche est implicite vu qu'elle ne contient pas explicitement la géométrie ou la topologie du modèle (Bidarra, et al., 2000). Par contre, la séquence d'opérations permet de générer les informations explicites du modèle, par exemple sa représentation B-REP décrite en termes de faces, d'arêtes et de sommets.

Cette concaténation de l'approche procédurale avec le modèle B-REP, permet de garder l'information sur chaque opération de modélisation effectuée, par exemple le type de feature créé, les paramètres, et les modèles de référence pour leur positionnement (Bidarra, et al., 2000). Les instances des features peuvent être modifiées en déterminant des nouvelles valeurs de leurs paramètres, ou peuvent être effacées du modèle. Ceci est faisable par la modification ou la suppression de l'opération respective de création du feature dans l'historique du modèle. Ensuite, un nouveau modèle B-REP est régénéré par une réexécution séquentielle des opérations dans l'historique du modèle.

Dans cette approche, plusieurs inconvénients peuvent être distingués. Dans la suite, on présente certains inconvénients qui ont été décrits dans (Bidarra, et al., 2000). Les trois premiers sont liés à la dépendance sur l'ordre chronologique de la création des features. Le quatrième est dû à la limitation de résolution des contraintes. Le cinquième est lié à l'évolution historique des entités dans le modèle B-REP évalué.

1) **Le coût de calcul** (*Computational Cost*) : la réexécution entière de l'historique du modèle après la modification et la suppression d'un feature conduit à un coût informatique assez élevé, proportionnel à la taille de l'historique du modèle. Plusieurs méthodes ont été conçues pour pallier ce problème, par exemple, l'enregistrement des représentations B-REP entre les étapes de l'historique. Un progrès intéressant consiste à enregistrer seulement les deltas entre les étapes de l'historique, et de revenir à l'état auquel le modèle a besoin d'être réévalué. Cette amélioration a besoin de moins d'espace mais exige plus de temps de calcul.
2) **L'ensemble des opérations non-associatives** (*Non-associative set operations*) (Union et différence) : l'évaluation du modèle ne garantit pas la correspondance du modèle avec les paramètres des features qui se chevauchent. Cette réévaluation est basée sur un ordre de précédence, qui est l'ordre chronologique de la création des features. Le processus d'évaluation utilise deux ensembles d'opérations non-associatives selon la nature du feature traité : Union pour les features additifs, et différence pour les features soustractifs.
3) **Les références sur les entités dans l'historique du modèle** (*Entity References in the model history*) : la réévaluation basée sur l'historique du modèle ne peut pas toujours traiter une opération de modification d'un feature, comme le positionnement de ce feature par rapport à d'autres entités dans le modèle (les entités créées après ce feature dans l'historique de construction). Par conséquent, l'évaluation du modèle par une réexécution progressive de la séquence d'opérations permet de se référer seulement aux entités B-REP écrites par des opérations précédentes.
4) **Les types de contraintes** : les contraintes peuvent être créées dans un modèle, et doivent être prises en compte durant l'exécution de l'historique de construction. Dans la plupart des systèmes, ces contraintes sont unidirectionnelles, par exemple la longueur d'un trou peut dépendre de la longueur d'une rainure, mais pas réciproquement.
5) **Problème de persistance des noms** (*Persistent naming problem*) : une opération de modélisation peut se référer à des entités topologiques dans la représentation B-REP du modèle courant, qui résulte des opérations précédentes. Par exemple, un nouveau feature peut se référer à une face ou une arête du modèle calculé. Chaque opération dans

l'historique de construction exige donc un ensemble spécifique d'entités topologiques du modèle. Ces entités topologiques peuvent être divisées, fusionnées ou effacées suite à des opérations de modélisation. La persistance des noms est le processus d'identification des entités topologiques suite à un changement dans le modèle géométrique (Kripac, 1995). Un exemple illustrant ce problème est présenté dans la figure 11. Supposons un congé appliqué à une arête a d'un modèle. Si une nouvelle opération est créée divisant cette arête en deux arêtes a_1 et a_2, alors une ambiguïté d'application de ce congé est produite. Le résultat est déterminé par un schéma de base de « *persistent naming* » spécifique à chaque application. Bien que le résultat soit déterministe, c'est à dire il faut obtenir le même résultat à chaque réexécution de la même séquence d'opérations de modélisation, il est ambigu, dans le sens où ce résultat n'est pas toujours prévu pas l'utilisateur du système de modélisation.

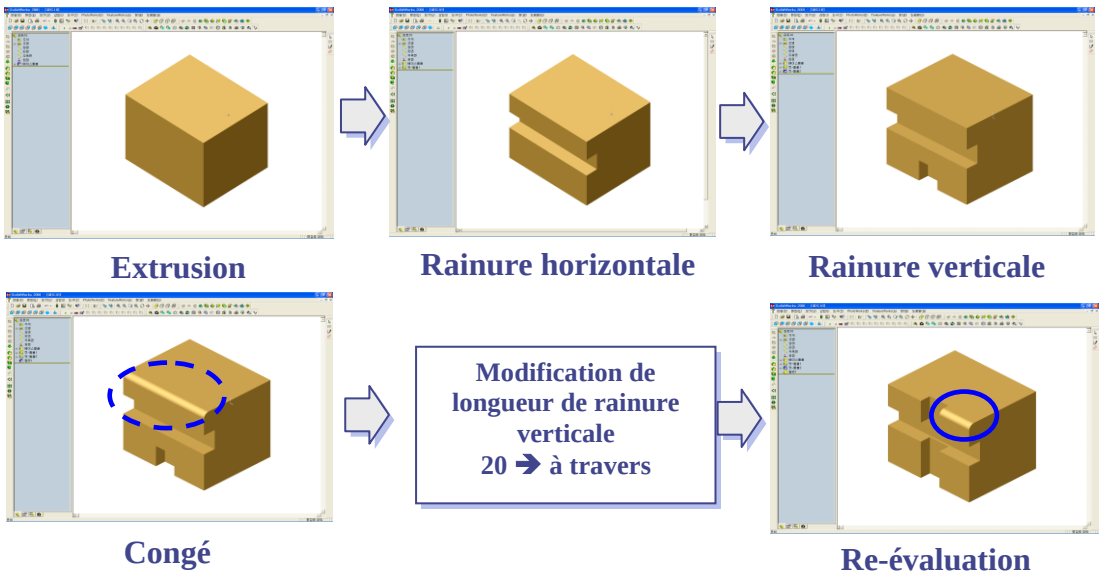


Figure 11. Problème de persistance de nom dans SolidWorks (Choi, et al., 2002)

Par ailleurs, tenant compte de la liberté que peuvent apporter les paramètres au concepteur CAO, une approche macro-paramétrique, basée sur l'approche procédurale, a été proposée dans (Choi, et al., 2002). Cette approche dérive de la méthode paramétrique basée sur l'historique de conception. Elle est destinée à faciliter l'échange des données paramétriques inclues dans l'intention de conception. Dans cette approche, les modèles CAO sont échangées en forme des fichiers « *macro* » contenant l'historique de commandes utilisées durant la phase de conception par l'utilisateur. Ces commandes sont classifiées, et un

ensemble de commandes de modélisation a été défini, et utilisé dans un format neutre (Mun, et al., 2003). Cette approche est illustrée dans la figure 12.

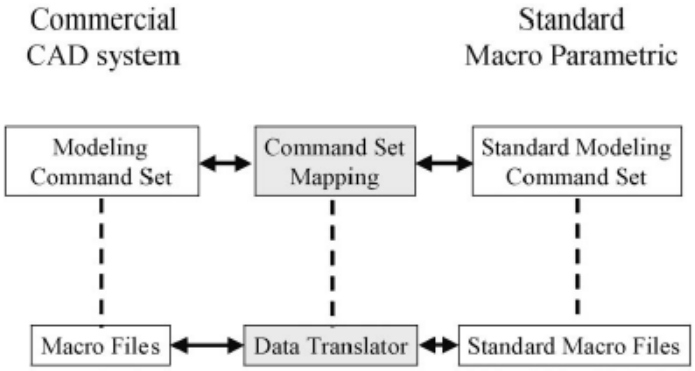


Figure 12. Correspondance dans l'approche macro-paramétrique (Choi, et al., 2002)

La majorité des systèmes de modélisation par feature sont paramétriques et basés sur l'historique de construction. Ils utilisent le modèle B-REP pour la représentation de la géométrie explicite résultante. Des exemples sont les systèmes commerciaux actuels tels que *Autodesk Mechanical Desktop* (Autodesk, 1999), *Pro/Engineer* (ProEngineer, 1999), *MicroStation Modeller*(Peters, 1997), et *I-DEAS Master Series*(IDEAS, 1999).

1.7.2 Approche basée sur un schéma déclaratif de features

Les problèmes provenant de la modélisation basée sur l'historique de construction, présentés ci-dessus, ont entraîné l'approche déclarative. En outre, deux aspects importants ne sont pas bien considérés par la plupart des systèmes actuels de modélisation par feature : d'une part, la sémantique des features est pauvrement définie, ce qui restreint la saisie de l'intention du concepteur. D'autre part, la faible maintenance de la sémantique des features aboutit éventuellement à une altération de l'intention du concepteur. Ces aspects engendrent des problèmes de la maintenance de validation des features (Bidarra, et al., 2000). Ainsi, certains travaux de recherche ont adopté un schéma déclaratif afin de combler certaines lacunes sémantiques résultant des approches de modélisation procédurale conventionnelle.

L'approche déclarative consiste à définir les classes de feature en précisant explicitement les relations entre les entités géométriques constituant le feature, soit en termes de relations entre les primitives de volume, ou entre les entités du modèle B-REP, notamment face, arête, et sommet (Shah, et al., 1995). Ainsi, la spécification de chaque classe de feature

comprend les critères de validité qui déterminent la sémantique de toutes ses instances (Bidarra, et al., 2000). Par ailleurs, les systèmes de modélisation par features s'engagent à vérifier la maintenance de tous les features du modèle conformément à ces critères. Certains systèmes se limitent à la détection d'un nombre de situations invalides prédéfinies (Vieira, 1995). Ensuite, la solution consiste à refuser l'opération de modélisation correspondante. L'avantage principal de la modélisation déclarative de feature est dans la liberté de spécification des contraintes, et par conséquent dans la manière dont un modèle peut être édité et maintenu. En revanche, à chaque fois qu'un changement est effectué sur la position ou l'orientation d'une entité géométrique, la validité des contraintes est toujours vérifiée. Ainsi, un changement est mis en place seulement si toutes les contraintes restent valides (Shah, et al., 1995). Par conséquent, ce système rigide entrave le processus de modélisation, et permet en outre certaines inconsistances non prévues dans le modèle.

Une autre approche de modélisation sémantique de feature a été proposée dans (Bidarra, 1999). Cette approche est une modélisation déclarative de features, où la spécification de feature et la maintenance du modèle sont clairement séparées. Toutes les propriétés des features, incluant les paramètres géométriques et les conditions de validité sont définies en termes de contraintes (Bidarra, et al., 2000). Dans cette approche, chaque feature a une signification ou une sémantique bien définie. Ceci est spécifié par des classes de feature qui sont des descriptions structurées sur toutes les propriétés d'un type de feature, définissant un modèle pour toutes leurs instances. Ces propriétés incluent les conditions de validité que les instances doivent satisfaire. Les conditions de validité sont indispensables pour maintenir la sémantique des features durant le processus de modélisation. Sans ces conditions, les features ne peuvent plus être considérés d'un niveau supérieur par rapport aux primitives de modélisation géométrique. Ces conditions, ainsi que la forme des features et leurs paramètres sont spécifiés en utilisant différents types de contraintes.

Dans cette approche, les utilisateurs peuvent définir leurs propres classes de feature, par exemple en héritant des classes existantes et en ajoutant certaines contraintes à leur définition. Ces classes sont enregistrées dans des librairies, à partir desquelles de nouvelles instances peuvent être créées. Une autre caractéristique de cette approche est que le processus de modélisation est accompli en termes de features et de leurs entités (ex : faces et paramètres), et des contraintes entre elles. Un des avantages est qu'un feature, et particulièrement ses faces et leurs noms sont persistants.

1.8 Synthèse

Nous avons introduit dans ce chapitre le domaine d'étude de notre travail, notamment la modélisation à base de features. Les features sont largement utilisés dans les systèmes actuels de CAO pour faire face aux limites de représentation de données purement géométriques. Dans ce cadre, nous avons présenté les différentes définitions et classifications des features proposées dans la littérature. Parmi les différents types de features, nous nous intéressons au features de forme, alors des entités de modélisation de haut niveau qui définissent une forme précise sur une pièce. Elles sont définies comme des instances d'une forme générique, ayant des attributs géométriques et non géométriques et des contraintes. De plus, ces entités portent une signification sémantique aux utilisateurs. Ainsi, l'utilisation du terme feature dans le reste de ce manuscrit désigne implicitement les features de forme.

Nous avons étudié les différentes techniques de création des features. Dans ce contexte, nous avons étudié deux approches principales, la reconnaissance des features et la modélisation par features. La première approche consiste à reconnaitre l'ensemble de features d'un modèle à partir de sa représentation géométrique. Malgré les différents algorithmes développés et l'utilité de cette approche dans des domaines tels que la fabrication, cette approche présente un inconvénient considérable dans la perte de la sémantique définie par le concepteur. Pour un même modèle géométrique, plusieurs interprétations et historiques de modélisation pourront être dérivés. Ainsi, nous nous intéressons à la modélisation par feature, où la sémantique d'un modèle de produit représente celle définie initialement par le concepteur.

Ensuite, nous avons distingué deux approches de modélisation par features : l'approche procédurale et l'approche déclarative. Nous nous intéressons à la première approche puisqu'elle est la plus répandue dans les systèmes actuels de CAO, et permet de garder l'historique de construction indispensable pour échanger l'intention de conception. Les modèles conçus avec l'approche procédurale sont appelés des modèles procéduraux. Ils sont basés sur l'historique de construction tout en utilisant la représentation B-REP pour la géométrie résultante. D'ailleurs, l'échange des features devient de plus en plus primordial pour maintenir la sémantique des données des modèles de produits. Ainsi, nous procédons dans le chapitre suivant à l'étude des différents travaux effectués sur l'échange des données des modèles CAO, notamment l'échange de l'intention de conception.

Chapitre 2 Interopérabilité des systèmes CAO

2.1 Introduction

Dans le chapitre précédent, nous avons étudié les modèles CAO, avec un intérêt spécifique aux modèles à base de features. Dans ce chapitre, nous nous intéressons au sujet de l'échange des données des modèles CAO. Dans ce cadre, nous examinons des standards actuels et des projets développés pour répondre aux besoins d'échange de données. (Bianconi, et al., 2006) distinguent deux domaines principaux d'échange de données : l'échange horizontal et l'échange vertical. Le premier échange concerne les données des différents systèmes situés au même niveau de développement de produit, par exemple la CAO. Dans le deuxième échange, il s'agit particulièrement d'échanger des données entre des systèmes de niveaux différents, tel que l'échange effectué entre un système de CAO et une application de phase ultérieure dans le cycle de vie du produit, par exemple la fabrication, les outils de prototypage, *etc*. Dans l'échange horizontal, un des majeurs problèmes porte essentiellement sur les difficultés d'édition du modèle dans le système cible après son transfert. En revanche, dans l'échange vertical, il est beaucoup moins fréquent que les modèles aient besoin d'être modifiés. Notre travail porte principalement sur la première catégorie d'échange puisqu'il s'agit d'interopérabilité des données des modèles CAO à base de features, notamment les modèles procéduraux durant la phase de conception.

Dans la suite de ce chapitre, nous allons présenter différentes approches qui ont été proposées pour l'interopérabilité des systèmes CAO. Dans ce contexte, nous présentons brièvement quelques standards développés, et nous détaillerons dans la section 2.3 le standard ISO 10303 pour l'échange des données de produit durant sa phase de développement. Dans la section 2.4, nous étudions quelques projets et solutions proposés pour l'échange de données pour répondre aux besoins d'échange de l'intention de conception.

2.2 Méthodes actuelles d'échange de données

Tout au long du cycle de vie d'un produit, des informations sont utilisées et échangées par différentes applications (Ghodous, 2003). Un objectif primordial dans le développement d'un produit consiste donc à faciliter la communication des données du produit entre les différents systèmes et applications intervenant dans le cycle de son développement (Shah, et al., 1995). Ces systèmes s'avèrent dissimilaires au niveau de leur application et hétérogènes dans leurs formats (Allada, et al., 1995).

Les features sont utilisés dans une grande gamme d'applications de développement de produit: conception de pièce et d'assemblage, gamme d'usinage, fabrication, analyse et inspection, *etc.* Leur échange peut être effectué entre des systèmes du même domaine d'activité, par exemple la conception, ou entre des applications de deux phases différentes du cycle de vie de produit. D'une part, l'avantage de l'exploitation des features, dans l'interopérabilité des systèmes du même domaine, réside dans la possibilité de maintenir dans le système destinataire les données complètes fournies par le système source, ce qui permettra, en conséquence, l'édition du modèle transmis dans le système destinataire. D'autre part, le maintien des features dans les modèles de produit a un impact direct sur la gestion du cycle de vie d'un produit. En effet, les features sont destinés à jouer un rôle important dans l'intégration des activités de CAO et de FAO (Allada, 1994). Par conséquent, leur échange a un potentiel important dans l'ingénierie simultanée, notamment sur la réduction significative du temps et du coût de production (Allada, et al., 1995). Tenant compte de la croissance de complexité des produits à modéliser et du nombre des systèmes CAO intervenant à sa modélisation, nous nous intéressons plus particulièrement à l'étude de l'échange de features entre les divers systèmes de conception à base de features.

Les premières approches qui ont été développées pour résoudre le problème de l'interopérabilité et la communication des données de produit sont basées sur un échange direct entre des applications (Dartigues, 2003). Ces méthodes offrent l'avantage de limiter les pertes sémantiques lors de l'échange de données mais elles impliquent un long et fastidieux travail de maintenance et de mise à jour. Suite à ces problèmes, et tenant compte du nombre croissant de systèmes CAO, des approches basées sur l'utilisation d'un format neutre ont vu le jour. Des formats neutres sont développés pour servir d'intermédiaire entre ces différents systèmes. Cette approche d'échange indirect rejette le besoin de développer des traducteurs

directs entre les applications. Ainsi, le nombre de traducteurs nécessaires pour l'interopérabilité de n applications se réduit de $O(n^2)$ en $O(n)$, comme illustré dans la figure 13. Des standards tels que *IGES* (Smith, 1988), *VDA-FS* ou *STEP* (Pratt, 2001 - b) ont ainsi été développés comme format neutre. Ils permettent une meilleure prise en compte des mises à jour mais augmentent le risque de pertes sémantiques (Ghodous, 1996). Dans notre travail, nous nous intéressons aux méthodes d'échange indirect tout en visant la réduction de cette perte de sémantique.

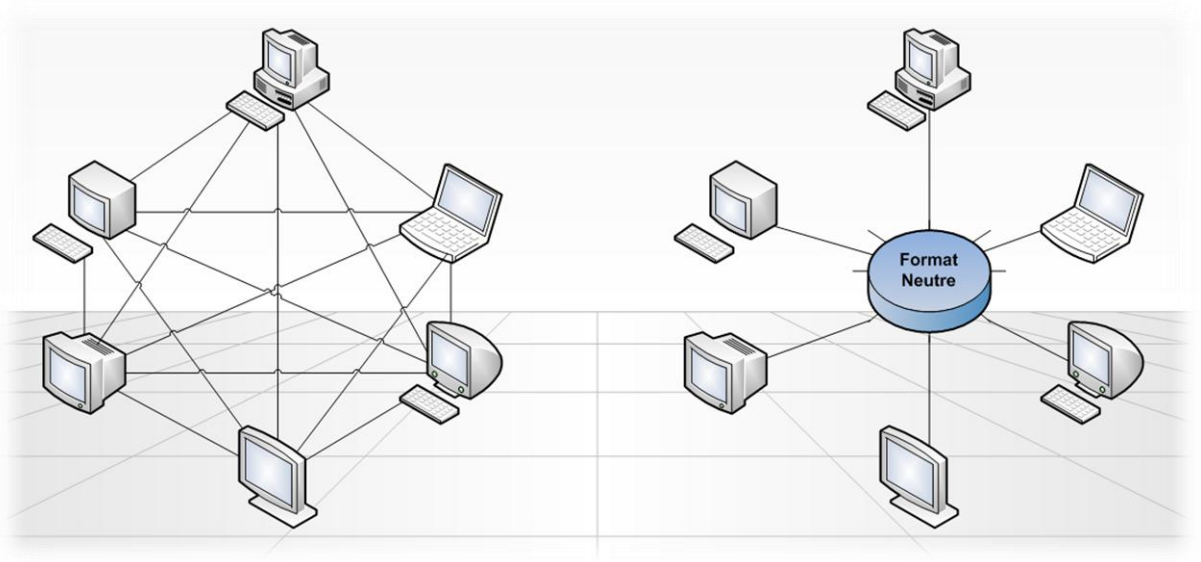


Figure 13. Approches d'échange: directe (gauche) et indirecte (droite)

Plusieurs formats neutres ont été développés, et notamment des standards géométriques ont vu le jour depuis les années 60 (Ghodous, 1996). Les standards offrent un grand intérêt pour le développement coopératif de produits. Tous ces standards ont été développés indépendamment les uns des autres. Certains d'entre eux sont le résultat d'efforts nationaux, comme SET (*Standard d'Echange et de Tansfert*) en France, IGES (*Initial Graphics Exchange Specifications*) aux États-Unis ou VDA-FS en Allemagne. Certains standards ont également été développés conjointement à des logiciels : formats DFT d'Autodesk, Neutral file de PTC, *etc*. Les échanges de données devenant de plus en plus mondiaux, la communauté internationale s'est regroupée autour de l'ISO (*International Organization for Standardization*), pour développer un standard d'envergure internationale. (Ghodous, 1996) montre que tous les standards qui ont été développés ont ensuite convergé vers un même standard : l'ISO 10303, connu sous le nom STEP (*STandard for the Exchange of Product model data*). Ce standard sera détaillé dans la section 2.3.

(Choi, et al., 2002) ont distingué deux approches d'échange de données. La première approche, basée sur une interface statique, consiste à échanger des données par l'intermédiaire d'un format neutre, par exemple STEP, IGES ou DXF (*Drawing Exchange Format*). L'autre approche, basée sur une interface dynamique, consiste à implémenter une interface d'échange procédural en utilisant l'interface de programmation, API (*Application Programming Interface*), des systèmes concernés. L'interface statique traite le contenu et la structure de données, tandis que l'interface dynamique fournit des opérations de modélisation géométrique des solides (Magleby, et al., 1991).

Pourtant, les standards actuels ne répondent qu'à une partie des besoins industriels en échange de données : DXF ne peut pas être utilisé pour les modèles solides 3D, et IGES ne peut pas traiter les données non géométriques. STEP permet de manipuler l'information géométrique, mais aussi l'information concernant le cycle de vie du produit. Dans la section suivante, nous étudions d'une façon plus détaillée le format STEP, les parties et les extensions de STEP visant à traiter les données plus complexes telles que celles définies dans l'intention de conception, des expérimentations développées basées sur STEP, ainsi que les limites de STEP du point de vue d'échange de la sémantique des données.

2.3 ISO 10303 - STEP

ISO 10303 est un ensemble de standards internationaux, désigné par STEP. Il fournit des représentations de données d'un produit, ainsi que les mécanismes nécessaires pour l'échange des données dans un format indépendant du système et interprétable par la machine (Pratt, et al., 2005). STEP est destiné à couvrir une vaste gamme de domaine d'applications (aérospatiale, architecture, automobile, électronique, électromécanique, *etc.*) durant les différentes phases du cycle de vie d'un produit (conception, analyse, gamme d'usinage, fabrication, inspection, *.etc.*). Sa syntaxe est représentée avec le langage formel EXPRESS, et sa représentation graphique avec son schéma EXPRESS-G (ISO 10303-11, 1992).

L'infrastructure de STEP est composée d'un ensemble de ressources intégrées (*Integrated Resources IRs*), qui définissent un modèle unique de données d'un produit. Les parties implantables du standard sont les protocoles d'applications (*Applications Protocols APs*). Chaque protocole *AP* est fondé sur des spécifications des concepts de base appropriés,

définies dans les *IRs*, afin de répondre aux besoins des classes spécifiques et au cycle de vie de produit (Pratt, et al., 2005).

STEP a été créé en 1984, et son premier ensemble de standards est réalisé en 1994, (ISO 10303-21, 1994). Dans la mesure où la forme du modèle était le point d'intérêt pour l'échange des données, les premières versions étaient destinées entièrement à l'échange des modèles explicites, définis en fonction de la géométrie et éventuellement de la topologie représentant les relations de connectivité entre les éléments géométriques (Pratt, et al., 2005). La partie 42 de STEP (ISO 10303-42, 1994) est appropriée pour cette représentation explicite du modèle. Elle inclut des entités de représentation du modèle CSG ou B-REP du produit, ainsi que d'autres entités utilisées dans la construction du modèle de produit, par exemple les opérations booléennes, le bossage, la rotation, mais elle n'inclut pas des fonctions (par exemple coque, dépouille) pour la modélisation par features (Pratt, 1998). STEP n'était pas donc approprié pour représenter l'intention de conception (Choi, et al., 2002). Une extension de STEP était donc nécessaire pour résoudre ces problèmes. Une extension, présentée dans (Pratt, et al., 2005), définit un schéma de STEP visant à échanger les modèles procéduraux.

En effet, STEP est en développement perpétuel pour augmenter ses capacités fonctionnelles[8] afin de répondre aux nouvelles exigences de conversion, surtout pour l'échange des modèles procéduraux largement utilisés dans les systèmes actuels de CAO, où les données intègrent l'historique de construction, les paramètres, et les contraintes. Des initiatives ont mené vers l'extension de STEP pour le traitement des informations paramétriques et les tolérances. Ainsi, de nouvelles ressources sont développées pour définir les paramètres et les contraintes des éléments géométriques, notamment dans la partie 108 (ISO 10303-108, 2005). De plus, d'autres nouvelles ressources ont été définies pour l'échange des modèles procéduraux, notamment les parties 55 et 111. Dans la section suivante, nous présentons quelques parties de STEP liées à l'échange des modèles procéduraux et des features.

[8] Les dernières mises à jour et les détails sont disponibles sur : http://www.tc184-sc4.org/

2.3.1 Parties de STEP liés aux modèles procéduraux

2.3.1.1 ISO 10303 – 48 (les features de forme)

La partie 48 définit un feature de forme comme étant un aspect de forme conforme à un stéréotype préconçu (ISO 10303-48, 1992). L'aspect de forme est défini par une forme ou une partie perceptible de forme. La spécification d'un feature comprend un nom de feature, des paramètres, ainsi qu'un label descriptif optionnel (Shah, et al., 1995). Cette partie de STEP traite l'information des features à deux niveaux : schéma de feature de forme et schéma de représentation des features de forme. La classification des features de forme est présentée ci-dessus (*cf.* figure 7 de la section 1.4). Cette partie a été abandonnée avant sa finalisation.

2.3.1.2 ISO 10303 – AP 224 : les features d'usinage dans la gamme d'usinage

La partie 224 de la norme STEP (ISO 10303-224, 1999) est un protocole de définition du produit mécanique pour la gamme d'usinage en utilisant les features d'usinage. Elle fournit un schéma de classification systématique des features du point de vue d'usinage. Cependant, cette partie n'identifie pas les relations fondamentales de la taxonomie des features avec les entités géométriques et topologiques de la pièce.

Plusieurs aspects différencient les deux parties 224 et 48 dus aux différences de fonctions attribuées à chaque partie (Shah, et al., 1995). La partie 224 est spécifique à une application (gamme d'usinage), tandis que la partie 48 est une ressource générique définissant des features indépendamment du domaine d'application. De plus, les features de la partie 48 n'ont pas de représentation unique. Cependant, les features communs de la partie 224, comme les poches, les trous, et les congés sont paramétrés d'une manière unique. Enfin, la partie 48 traite uniquement la forme exacte du produit, alors que la partie 224 inclut les tolérances.

2.3.1.3 ISO 10303 – 111 : les features de l'historique de construction

En dépit de l'existence des entités de modélisation de forme dans la partie 42 de STEP, utilisées comme opérations de construction dans les modèles procéduraux, ces entités sont actuellement limitées en nombre, par rapport aux opérations de modélisation par features

disponibles et fournies par les systèmes actuels de CAO [9] (Pratt, et al., 2005). L'extension de ces entités dispense l'utilisateur d'un travail fastidieux qui est dû à l'utilisation des entités de bas niveau pour la construction d'un modèle. La partie 111 (ISO 10303-111, 2003) est conçue pour représenter des opérations de construction de haut niveau (à base de features) en définissant des entités supplémentaires. Les features de forme définis dans cette partie incluent des types d'entités de haut niveau comme les poches, les trous, les nervures, les gorges, et les bossages. Des sous types spécifiques peuvent être définis. Par exemple, les sous-types du feature « trou » incluent les trous « borgnes » et les trous « à travers ». Pour les trous « borgnes », différents types de fonds sont spécifiés (plat, sphérique, conique). De plus, cette ressource offre la possibilité d'ajouter aux solides des congés, des chanfreins, et supporte les features de répétition (*pattern*).

L'interprétation des features dans les parties 111 et 224 est différente. Effectivement, un feature de l'AP 224 est considéré comme une région sur la surface d'un modèle existant, tandis que dans la partie 111, un feature est une fonction de construction de la forme du modèle, qui pourrait ainsi être modifié ou effacé par l'effet des features subséquents (Pratt, et al., 2005).

2.3.1.4 ISO 10303 – 55 : La représentation procédurale et hybride

Cette ressource fournit des mécanismes de base pour la modélisation procédurale, basée sur l'historique de construction (ISO 10303-55, 2005). L'ordre correct des opérations de modélisation est crucial dans l'approche procédurale, puisque l'exécution d'un même ensemble d'opérations de construction, dans différents ordres, mènera à différents résultats pour la forme du modèle. L'entité fondamentale de cette partie est *procedural_representation_sequence*, qui permet de représenter une séquence ordonnée d'opérations (Pratt, et al., 2005).

2.3.1.5 ISO 10303 – 108 : Les paramètres et les contraintes

Cette ressource permet la représentation des paramètres, des contraintes explicites, et des esquisses 2D (ISO 10303-108, 2005). Dans un modèle CAO, un paramètre est une valeur quantitative qui peut être changée par l'utilisateur afin de modifier le modèle, par exemple la

[9] A l'exception des configurations de forme générées par l'extrusion ou la révolution des esquisses 2D, qui sont fournies par la partie 42 de STEP, et qui sont considérés par certains systèmes CAO comme des features.

modification des dimensions (longueur et largeur) d'un rectangle. Dans un modèle procédural, les contraintes géométriques sont souvent inhérentes aux opérations de construction : ce sont les contraintes implicites imposées par le système. Par exemple, les contraintes de parallélisme et de perpendicularité sont automatiquement imposées pour un rectangle. Cependant, les contraintes explicites, représentées par des entités individuelles dans la structure du modèle de données, définissent des relations entre des éléments géométriques et topologiques du modèle. Les contraintes explicites sont utilisées, par exemple, pour définir les types de relations suivantes :

- Les relations entre les éléments géométriques des esquisses ou profiles 2D.
- Les positions des features par rapport à d'autres éléments dans le modèle.
- Les positions relatives et les orientations des pièces dans un modèle d'assemblage.

2.3.1.6 ISO 10303 – AP 203 (1ère édition)

Le protocole AP 203 (*Configuration Controlled 3D Designs of Mechanical Parts and Assemblies*), existant depuis les premières versions du standard, est fréquemment utilisé dans plusieurs secteurs de l'industrie (Pratt, et al., 2005). La première édition de l'AP 203 (1994) fournissait la représentation de la forme du produit, la structure d'assemblage, ainsi que certaines informations administratives (version de la pièce, les données d'autorisation, *etc.*). La représentation de la forme d'un produit peut être filaire, surfacique, ou volumique. Les assemblages sont représentés comme des collections de pièces orientées et positionnées. Cependant, cette édition du protocole connaissait certaines restrictions : elle ne permettait pas l'échange des modèles procéduraux, ce qui engendre la perte de certaines informations sémantiques présentes dans le système émetteur. Par conséquent, le modèle transféré dans un système cible ne sera pas facilement éditable. Une 2ème édition de l'AP 203 est réalisée afin de surmonter les problèmes causés par la première édition.

2.3.1.7 ISO 10303 – AP 203 (2ème édition)

L'avantage majeur de cette nouvelle édition réside dans la capacité de transfert des modèles procéduraux de CAO, c'est à dire l'historique de construction ainsi que les features incluant des informations paramétriques et des contraintes. Les ressources intégrées nécessaires pour cette édition ont été publiées par l'ISO, et par conséquent peuvent être référencées par ce protocole afin d'effectuer les extensions techniques nécessaires (Pratt, et al., 2005).

Dans le cadre de la norme ISO 10303, d'autres tentatives ont été effectuées pour supporter le transfert de l'intention de conception entre les systèmes CAO. Par exemple, l'AIC 533 (*Application Interpreted Construct*) concerne la spécification des features d'usinage. Le protocole d'application AP 224 se concentre sur l'entrée des systèmes de gamme d'usinage (*Process planning*).

2.3.2 Expérimentations

De multiples efforts ont été effectués par des organisations académiques et industrielles pour tester l'échange des modèles procéduraux entre les systèmes CAO (Pratt, et al., 2005). La partie 108 de STEP a été testée par l'institut américain NIST (*National Institute of Standards and Technology*). Le KAIST (*Korean Advanced Institute of Science and Technology*) a développé une méthode d'échange des modèles procéduraux selon la méthode macro paramétrique (Choi, et al., 2002). Les modèles échangés sont représentés en forme des fichiers « *macro* » qui définissent la séquence de commandes de modélisation utilisée par le concepteur durant le processus de modélisation. Cette approche n'est pas complètement compatible avec STEP, mais a été développée dans ce sens. L'université St. Gallen en Suisse, en collaboration avec l'organisation ProSTEP, a développé des projets de conversion basés sur les parties 108 et 203 pour le transfert des modèles paramétriques de CAO.

2.3.2.1 SMCH (*Solid Model Construction History*)

Le groupe ISO/TC184/SC410 a proposé le schéma SMCH qui inclut des modèles pour l'échange de l'information paramétrique (Anderson, 2001). Le schéma SMCH est composé d'une structure de données permettant l'échange des données paramétriques, des contraintes, des features, et l'historique de conception. Ce schéma est appliqué dans les domaines suivants (Anderson, 2001):

- Les modèles de solide contenant des features paramétriques et des contraintes géométriques.
- La représentation des entités implicites et des opérations permettant l'échange des modèles basés sur l'historique de conception.

[10] http://www.tc184-sc4.org/

- Les structures de la partie 42 (2ème édition) pour l'échange des modèles CSG en utilisant des opérations booléennes (*Union, Intersection, et Différence*) sur des primitives de solide, des solides B-REP manifold, et d'autres solides.

2.3.2.2 ENGEN (*Enabling Next GENeration mechanical design*)

ENGEN (Anderson, et al., 1998) est un consortium gouvernemental/industriel dirigé par l'ATI (*Advanced Technology Institute*), financé par DARP (*Defense Advanced Projects Research Agency*) et PDES Inc. Son but principal était d'étendre le schéma EXPRESS de STEP afin de supporter les paramètres et les contraintes (Baumann, 2004). Il vise également à accélérer le développement et le déploiement du standard STEP (Pratt, et al., 2001). La durée formelle du projet était de 1995-1998. Les transferts achevés dans (Anderson, et al., 1998) se sont focalisés sur les modèles explicites avec des paramètres et des contraintes. Une extension a été réalisée pour supporter le transfert des modèles à base de features (Pratt, et al., 2001). Le projet ENGEN propose le modèle EDM (*ENGEN data model*) qui est un modèle de données de produit définissant des paramètres, des features, l'historique de construction, et des contraintes basées sur la partie 42 de STEP. L'objectif était d'assurer la capacité d'échange de l'intention de conception.

A cet effet, un squelette « *Construct Module* » a été construit, spécifiant un nombre limité d'opérations de conception en 2D. Le langage EXPRESS (ISO 10303-11, 1992) ne peut pas représenter des fonctions mathématiques comme des procédures ayant des entrées, sorties, et actions (Pratt, et al., 2001). Ainsi, il sera impossible de transférer des procédures de construction sans une extension, d'une certaine façon, du langage. Pour éviter ce problème, ENGEN définit les fonctions de construction EDM avec une méthode à base d'entités (*entity-based approach*).

L'exemple suivant, présenté dans la figure 14, illustre l'approche définie par ce projet. Considérons la définition simplifiée d'une entité « *ligne* » dans ISO 10303-42 (Figure 14 - a). Cette spécification définit une classe de ligne non limitée, caractérisée par les attributs « *point* » et « *direction* ». Elle est décrite avec le langage EXPRESS qui fournit une relation d'héritage entre les classes et leurs sous-classes. Par conséquent, cette entité hérite de cette hiérarchie l'attribut « *representation_context* » relié au système local de coordonnées. Pour créer une instance de ligne, les attributs *point, direction,* et *representation_context* seront remplacés respectivement par des références sur des instances d'un point cartésien, un

vecteur, et un contexte de représentation spécifique. Cette instance de ligne est purement descriptive, définissant une relation *statique* entre la ligne, le point, le vecteur, et le système des coordonnées dans lequel ils existent. Dans le cas où cette ligne est transférée dans un système destinataire, et, si par exemple, la position du point est éditée, alors les données deviennent inconsistantes avec les données originales puisque l'intention de conception n'a pas été transmise.

```
ENTITY line
        SUBTYPE OF (curve);
        pnt : cartesian_point;
        dir : vector;
END_ENTITY;
```

(a)

```
ENTITY constructed_line
        SUBTYPE OF (constructed_curve);
        pnt : cartesian_point;
        dir : vector;
END_ENTITY;
```

(b)

Figure 14. Une "ligne" dans ISO 10303-42 (a) et dans ENGEN (b)

Maintenant, nous considérons la définition de la ligne dans le modèle EDM (simplifiée) illustrée dans (Figure 14 - b). Cette définition semble similaire à la définition précédente. Cependant, leur sémantique est complètement différente. Dans EDM, la transmission d'une instance, *dynamique*, de cette entité est considérée, dans le système destinataire, comme étant une instruction de construction de ligne passant par un point, et ayant une direction déjà définie dans le modèle généré. De plus, les relations entre le point, le vecteur, et la ligne sont traitées comme étant des contraintes une fois la ligne générée, de façon que si l'un des attributs est subséquemment édité dans le système destinataire, la ligne va être modifiée convenablement. C'est un exemple des contraintes implicites, autrement dit, les contraintes qui ne sont pas explicitement imposées sur des éléments du modèle.

(Pratt, et al., 2001) présentent les inconvénients du modèle EDM comme suivant : un ensemble rudimentaire d'opérations de construction 2D a été spécifié. Les opérations sont définies selon une approche à base d'entités plutôt que sur une approche à base de procédure de construction. Le module de construction «*Construct Module* » du modèle EDM s'applique à un domaine extrêmement limité. Par ailleurs, il utilise une représentation à base d'entités pour des opérations de construction qui peut engendrer certains problèmes d'incompatibilité.

2.3.2.3 CHAPS (*Construction History And ParametricS*)

C'est un projet créé par l'organisation PDES Inc., un consortium international basé aux Etats Unis qui vise à développer et à tester des logiciels et des méthodologies autour de

STEP. La première phase de ce projet concernait l'historique de construction, en utilisant une version de la partie 55 de STEP. Durant la deuxième phase, les paramètres et les contraintes ont été intégrés en utilisant la partie 108 de STEP. Ces deux phases utilisaient les versions de développement de la partie 111 pour transférer les informations concernant les features. Les majeurs problèmes rencontrés sont dus à la présence, dans certains fichiers de tests, des types de features en dehors du domaine défini pour le projet (Stiteler, 2004). Les résultats des tests effectués sur ce projet ont manifesté une réduction significative du temps entre la phase de conception et de fabrication, ainsi qu'une amélioration de la qualité de produit.

2.3.3 Limites de STEP

Les standards actuels d'échange de données servant de format neutre, tels que STEP, ont montré des performances remarquables lors de la conversion de la forme d'un produit à travers des entités géométriques et topologiques. Cependant, les restrictions d'échange sont liées aux informations de l'intention de conception, telles que la sémantique associée aux features et à d'autres informations non géométriques liées au modèle du produit.

En effet, les inconvénients sont liés à l'incapacité de STEP de représenter ou de convertir la sémantique des modèles CAO. Malgré ses différentes tentatives de conversion de l'intention de conception, STEP ne peut définir que de la représentation syntaxique des données d'un produit de sorte que l'intégration de la sémantique n'est pas prise en compte (Seo, et al., 2005). Par conséquent, STEP ne parvient pas à traduire la sémantique appropriée, entre les différents systèmes de développement d'un produit. Il s'intéresse uniquement à la traduction de la terminologie d'un système à un autre et ne permet pas de conserver la signification liée à la conception d'un contexte à un autre (Patil, et al., 2005). D'ailleurs, l'information, qui est perdue dans un contexte, pourrait être potentiellement nécessaire dans un autre système. Enfin, STEP ne fournit pas une infrastructure de représentation d'une base de connaissances, et par conséquent ne permet pas d'effectuer des opérations de raisonnement sur la base des données représentées.

Pour surmonter les limites des standards actuels d'échange de données, des travaux de recherche et des projets ont été développés visant à traiter la sémantique des données. Ces travaux seront présentés dans la section suivante de ce manuscrit.

2.4 Méthodes d'échange de la sémantique

2.4.1 Echange direct basé sur l'API des systèmes CAO

Pour surmonter les problèmes liés à l'échange indirect des données, certaines solutions proposées sont basées sur un échange direct entre deux applications de CAO. De nos jours, la plupart des systèmes commerciaux de CAO, tels que CatiaV5, SolidWorks, Pro/Engineer, et Unigraphics, permettent la conception à base de feature des modèles complexes (en pièces ou en assemblages). Les modèles résultants intègrent des séquences ordonnées d'opérations de modélisation utilisées pour la création du modèle. Cette notion d'historique d'opérations de modélisation permet de préserver les connaissances liées aux éléments du modèle, ainsi que les relations avec d'autres éléments.

D'ailleurs, l'historique de modélisation est complètement enregistré dans le fichier de données ayant un format spécifique au système CAO, par exemple les fichiers « .CATPart » pour les pièces conçues avec CatiaV5. De plus, ces données sont accessibles à travers les interfaces de programmation, APIs. Une API consiste en un ensemble large de fonctions et de procédures donnant accès à l'interface graphique du logiciel, le gestionnaire des fichiers, et la base de données. Elle fournit donc une interface simple entre le logiciel et le monde extérieur. Ainsi, différentes solutions d'échange de données, basées sur les API, ont été développées pour surmonter les limitations rencontrées lors de l'échange avec les formats neutres.

Un exemple de solutions basées sur l'API est le logiciel de conversion des features entre les deux systèmes CAO, CatiaV5 et SolidWorks, développé au sein du partenaire industriel de la thèse, à savoir Datakit SARL[11]; Nous avons participé au développement de cette interface dans le cadre de cette thèse. Cette solution consiste à échanger les modèles à base de features directement entre les deux systèmes CatiaV5 et SolidWorks. Elle permet donc aux utilisateurs SolidWorks de récupérer les informations liées à l'arbre de construction et aux features CatiaV5, c'est-à-dire aux opérations de construction géométrique mais aussi d'usinage, d'une pièce ou d'un assemblage CatiaV5. Cette interface a subi de nombreux tests et a été officiellement lancée sur le marché mondial en 2006 lors du salon « *SolidWorks World 2006* » à Las Vegas.

[11] www.datakit.com

La méthode retenue par Datakit se base essentiellement sur la lecture de l'arbre de construction des modèles exacts de CatiaV5 et ensuite sur une écriture directe dans SolidWorks via son API. Cette interface d'échange de features offre la possibilité de travailler sur la base de paramètres, bien plus simples à gérer et à éditer que des données complexes de construction de surfaces. A titre d'exemple, comparons la simplicité et la rapidité de choix des paramètres d'un rayon et d'une arête dans le feature « *congé* » par rapport à la complexité de la définition d'une surface explicite qui va construire ce même congé. Ainsi, L'utilisateur récupérant l'historique de construction et les features CatiaV5 dans son logiciel destinataire, va pouvoir très simplement éditer les attributs et les modifier dans SolidWorks. Un exemple d'une pièce convertie entre CatiaV5 et SolidWorks est illustré dans la figure 15.

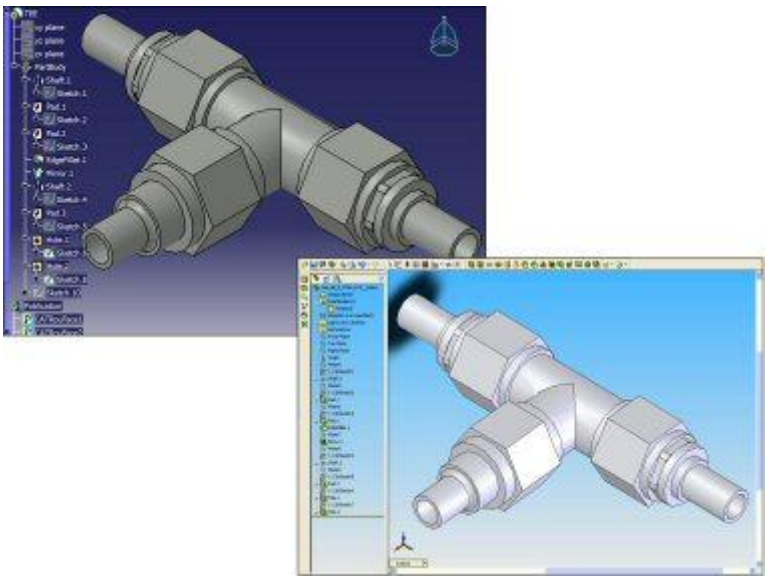


Figure 15. Exemple d'échange direct des features via l'API (Solution à Datakit)

Pourtant, certaines incompatibilités entre les deux logiciels pourront aboutir à l'échec de la construction d'un nouveau feature dans SolidWorks. Dans ce cas, le logiciel de conversion surmonte ces problèmes par la représentation B-REP correspondante au feature en question. Enfin, l'utilisateur pourra choisir d'activer ou de désactiver la fourniture de l'arbre et des features.

L'avantage principal des solutions directes réside dans la capacité du transfert, en plus des informations géométriques, de certains éléments de l'intention de conception qui ne sont pas transférés avec des standards actuels tels que ISO 10303, notamment l'historique de conception et d'autres informations supplémentaires qui seront utiles lors du développement du produit.

Cependant, le traitement des données reste toujours limité au niveau syntaxique basé sur des règles prédéfinies par un expert du domaine. Bien que l'approche directe permette le transfert de certaines informations sémantiques, elle ne permet pas la représentation formelle de cette sémantique associée aux données. De plus, les APIs connaissent les mêmes problèmes des formats neutres concernant l'incapacité de fournir des moyens de raisonnement automatique sur la base de connaissances d'un produit. Par conséquent, l'approche directe ne peut pas être considérée comme une solution optimale permettant l'amélioration de l'interopérabilité dans un environnement collaboratif.

2.4.2 Autres travaux sur l'interopérabilité sémantique

Certains travaux de développement et de recherche se sont intéressés à la problématique de l'interopérabilité sémantique des informations d'un produit. Le manque de représentation formelle de la sémantique des données d'un produit entrave l'efficacité de son échange. Plusieurs travaux de recherche se sont ainsi intéressés à la création d'une représentation d'un produit dans le domaine de la conception collaborative. Ces efforts visent à répondre aux questions de haut niveau de conception collaborative. Ceci est effectué par la prise en compte non seulement du «*quoi*», mais aussi du «*comment*» et du «*pourquoi*» d'une conception. Une étude détaillée de ces efforts de recherche est fournie dans (Shah, et al., 1995) (Szykman, et al., 2001).

D'un point de vue historique, EREP (*Editable REPresentation*) représente la première approche de définition des modèles procéduraux de CAO (Bianconi, et al., 2006). EREP décrit les problèmes fondamentaux liés à la définition d'un modèle procédural, et par conséquent au développement des systèmes d'échange de données des modèles procéduraux. EREP est un schéma de représentation des données de haut niveau d'un produit indépendamment de tout modeleur géométrique (Homann, et al.). L'objectif du projet EREP est de spécifier complètement et sans ambiguïté la géométrie sans se référer aux méthodes avec lesquelles la géométrie a été implémentée. Pratiquement, il s'agit de définir les spécifications textuelles des opérations de modélisation, telles que la création d'un feature et sa modification. Ainsi, EREP définit des concepts abstraits tels que assemblage, pièce, feature, profile, contraintes (algébrique et géométrique). Cependant, l'implémentation de cette approche avec les noyaux des systèmes de modélisation existants n'a pas été détaillée, et ainsi les activités expérimentales sur les pièces mécaniques n'ont pas été effectuées. En

conséquence, l'application de cette approche sur l'échange des données entre les systèmes CAO ne peut pas être évaluée (Bianconi, et al., 2006).

Un effort d'importance significative est le développement de PSL (*Process Specification Language*) au NIST « *National Institute of Standards and Technology* » (Gruninger, et al., 2003). Ce projet résulte des travaux de développement d'une ontologie dans le domaine de la fabrication. PSL définit un langage de représentation neutre pour l'interopérabilité d'informations entre différents systèmes liés aux processus de fabrication. L'objectif de PSL est d'axiomatiser un ensemble de primitives sémantiques nécessaire pour la description des concepts fondamentaux des processus de fabrication. Techniquement, les axiomes de PSL sont organisés en un noyau « PSL-Core », représentés avec le langage KIF (*Knowledge Interchange Format*) (Genesereth, et al., 1992), et un ensemble d'extensions. Bien que PSL se focalise sur la représentation des données spécifiques au processus de fabrication, des efforts ont été effectués afin d'assimiler des concepts de conception et de fabrication. Mais, ces efforts restent limités et ne sont pas suffisants. De plus, ces efforts ont été limités à la traduction de la géométrie et des informations connexes de la conception à l'usinage (Patil, 2005).

(Dartigues, et al., 2007) proposent une approche basée sur l'ontologie afin de permettre l'échange des features entre deux phases de développement d'un produit, la conception détaillée et la gamme d'usinage. Ils ont développé une ontologie partagée servant d'intermédiaire entre les différentes applications et des ontologies spécifiques aux domaines avec le langage KIF. Les ontologies spécifiques sont développées après l'analyse des systèmes CAO et des systèmes de gamme d'usinage où des règles de correspondance ont été spécifiées pour traduire des concepts d'un domaine à un autre.

(Patil, et al., 2005) définissent un environnement d'échange des données sémantiques d'un produit entre différents domaines d'applications de CAO. Le résultat de ce travail est le développement d'un outil de représentation de la sémantique d'un produit appelée PSRL « *Product Semantic Representation Language* » basé sur l'ontologie. Pour permettre l'interopérabilité sémantique, la logique mathématique et un mécanisme de raisonnement correspondant ont été utilisés pour déterminer les équivalences sémantiques entre des concepts des applications de CAO et ceux de l'ontologie PSRL. L'implémentation de ce prototype est discutée dans (Seo, et al., 2005), où des descriptions et des axiomes sont définis

à partir d'une étude comparative entre deux systèmes de conception à base de features, SolidWorks et Unigraphics.

(Spitz, et al., 2004) ont défini une méthodologie d'échange de données à base de features entre différents systèmes commerciaux de CAO. Leur approche est basée sur une architecture de représentation universelle de produit, appelée UPR (*Universal Product Representation*). (Kim, et al., 2003) ont proposé une ontologie de produit, et décrivent une méthode de partage de l'information par un processus de mise en correspondance sémantique basée sur l'ontologie développée. Cette méthode porte essentiellement sur les procédures de conception. (Choi, et al., 2002) définissent un ensemble neutre de commandes de modélisation décrivant l'historique de construction et les features de modélisation. L'échange de données se réalise par l'utilisation d'un fichier XML contenant l'historique de commandes utilisées lors de la conception d'un modèle. Bien que cette méthode permette la mise en correspondance entre les différentes terminologies, ayant une même signification mais une syntaxe différentes, la mise en correspondance n'est effectuée que syntaxiquement, mais pas sémantiquement. Pour tenir compte de l'aspect sémantique, (Seo, et al., 2005) étendent l'approche définie dans (Lee, et al., 2003) en utilisant l'ontologie pour réaliser l'interopérabilité sémantique des systèmes hétérogènes de CAO. L'ontologie développée est basée sur l'approche macro-paramétrique.

D'ailleurs, les systèmes *CAx* à base de feature n'ont pas de taxonomie explicite de features. Afin d'échanger les features entre les différents systèmes CAO, il est nécessaire d'organiser et de classifier les features de façon indépendante du domaine d'application. Ainsi, le problème de définition d'une taxonomie de features appropriée a été reconnu comme étant un des problèmes fondamentaux afin de pouvoir partager pertinemment les modèles CAO à base de features, y compris les informations sur l'historique de conception et les features. Plusieurs schémas de taxonomie de features ont été proposés dans la littérature comme le projet CAM-I (Butterfield, et al., 1986). CAM-I construit une taxonomie de features pour une représentation globale des données dans les systèmes CAO.

Les progrès réalisés sur le développement de ces taxonomies sont fondés sur l'utilisation des ontologies. (Horváth, et al., 1998) ont créé une ontologie pour la conception à base de features. Ils ont classifié les éléments de conception en termes d'entités, de phénomènes et de situations. Ces ontologies développées sont appliquées au niveau d'applications commerciales, et restent insuffisantes en termes de détails requis pour la

conception mécanique. (Mun, et al., 2003) ont défini une ontologie de feature basée sur la définition des commandes de modélisation des systèmes CAO. Cette ontologie est implémentée par une approche déclarative, qui a été développée avec le langage F-Logic (Angele, et al., 2004). Concrètement, le fonctionnement de l'approche définie se base sur les fichiers textes générés par certains systèmes CAO qui contiennent le journal de l'historique de commandes utilisées durant la modélisation d'un produit.

Certains travaux de recherche sur la représentation avec des ontologies ont été effectués pour la phase de conception conceptuelle et détaillée, par exemple l'ontologie développée par (Kitamura, et al., 2004) pour la représentation des données de la conception fonctionnelle. Cette ontologie a ses limitations quand il s'agit de la saisie de phénomènes mécaniques complexes.

2.5 Synthèse

Nous avons présenté dans ce chapitre une étude sur les différentes méthodes d'échange de données entre les systèmes CAO. Dans ce cadre, nous avons étudié les principaux problèmes liés à l'échange des données dans un développement coopératif de produits. Nous avons distingué l'échange horizontal, entre les systèmes CAO, et l'échange vertical entre un système CAO et une application de phase ultérieure telle que la fabrication. Nous avons restreint notre travail au problème liés à l'échange des modèles CAO à base de features durant la phase de conception du produit.

Actuellement, la plupart des approches existantes reposent sur l'échange des données géométriques d'un produit. Cela s'explique par le fait que la géométrie tient une partie importante dans le travail des concepteurs. Lors de cet échange, un problème significatif réside dans la difficulté d'édition des modèles transférés dans les applications destinataires. Cette difficulté provient de la perte des informations sémantiques durant le processus d'échange, notamment les features de conception. Ce processus d'échange engendre un modèle géométrique « *inerte* », dépourvu de l'intention de conception représentée par l'historique de construction, les features, les paramètres, *etc.* Or, dans le cas où un modèle doit subir des modifications nécessaires, le manque de ces informations pourra mener jusqu'à la recréation du modèle.

Ainsi, Il devient donc primordial de faire évoluer les méthodes existantes pour l'échange de données afin qu'elles prennent en considération non seulement l'information géométrique liée au produit (et dont les experts ont toujours besoin) mais également des informations liées au produit telles que les informations techniques ou encore la sémantique rattachée aux informations géométriques. Nous avons étudié dans ce cadre les standards actuels d'échange de données, tels que STEP. STEP est un format neutre permettant la représentation et l'échange des données entre différentes phases du cycle de vie d'un produit.

STEP a montré de bons résultats lorsqu'il s'agit d'un échange de la forme d'un produit par le traitement des entités géométriques et topologiques du modèle de produit. De nombreuses extensions de STEP visent à traiter les modèles procéduraux entre les systèmes CAO afin de conserver l'intention de conception. Cependant, STEP ne tient pas compte de certains aspects essentiels pour l'interopérabilité sémantique des données. En fait, STEP reste limité au traitement syntaxique des données sans prendre en considération la sémantique associée aux données. En outre, STEP ne fournit pas de moyens de représentation formelle des connaissances liées à un modèle d'un produit. Certaines informations utilisées durant le développement d'un produit peuvent avoir différentes significations en fonction du contexte dans lequel elles sont utilisées.

Dans la suite de notre travail, nous allons nous placer au niveau de la représentation des informations qui doivent être échangées. En effet, notre objectif est de pouvoir représenter les informations qui déterminent un produit et de pouvoir également caractériser le contexte dans lequel ces informations sont considérées. Pour cela, nous allons nous intéresser aux technologies du Web Sémantique, telles que les ontologies. Nous présentons dans le chapitre suivant une étude des modèles de représentation des connaissances ainsi que d'autres outils du Web Sémantique permettant l'élaboration des règles de correspondance. Ces études nous serviront de base pour l'élaboration de notre méthode d'échange de données, basée sur la prise en compte de la sémantique associée aux modèles de produits et du contexte dans lequel les données ont été utilisées.

Chapitre 3 Méthodologie d'échange basée sur le Web Sémantique

3.1 Introduction

Dans le chapitre précédent, nous avons présenté les solutions et les standards actuels permettant l'échange des modèles CAO à base de features. Nous avons également montré l'avantage de l'échange indirect basé sur un format neutre. Le défi principal réside dans l'expressivité du format neutre afin de pouvoir réduire la perte de la sémantique des données. Dans ce cadre, nous avons constaté que les standards actuels d'échange de données, tels que STEP, ne permettent pas l'échange de la sémantique définie par l'intention de conception, ainsi qu'ils ne considèrent pas le contexte dans lequel les données sont définies.

Dans un environnement collaboratif, les développeurs utilisent les technologies du Web et de l'Internet pour répondre aux attentes de l'industrie en termes de réduction du temps de mise sur le marché et du coût du développement de produit durant son cycle de vie (Manley, 2006). Ainsi, il est devenu de plus en plus primordial de mettre en place des solutions d'intégration des différents processus du développement de produit tout en tirant parti des nouvelles technologies de l'intelligence artificielle et de la représentation de connaissances. Pour réaliser cette intégration, il est nécessaire de créer et partager des modèles de représentation des données d'un produit, tout en tenant compte de la sémantique associée.

Afin de pouvoir traiter la sémantique des données d'un produit, nous avons besoin de trouver un moyen adapté pour représenter les features. Dans le domaine de la représentation des connaissances, les différents efforts académiques et travaux de recherche ont abouti à la définition des modèles de connaissances appelés ontologies. Une ontologie sert de moyen efficace pour la représentation de la sémantique des données de sorte qu'elle soit interprétable par la machine. Nous allons donc proposer une méthodologie d'échange de données basée sur la construction d'une ontologie commune servant d'intermédiaire entre les différentes applications de CAO.

D’abord, nous rappelons, dans la première section de ce chapitre, des technologies du Web Sémantique, notamment les langages d’ontologies et des règles. Ainsi, nous mettons en relief le bénéfice que pourraient apporter ces technologies pour l’interopérabilité de la sémantique des données. Ensuite, nous définissons la méthodologie de notre approche ontologique basée sur un échange indirect par l’intermédiaire d’une ontologie commune de features de conception, que nous avons appelée CDFO, *Common Design Features Ontology*.

3.2 Le Web Sémantique

Le Web Sémantique désigne un ensemble de technologies visant à rendre le contenu des ressources du *WWW (World Wide Web)* accessible et utilisable par des logiciels ou des agents, grâce à un système de métadonnées formelles, utilisant notamment la famille de langages développés par le consortium W3C[12]. Il a été défini par son créateur, *Tim Berners-Lee* (Berner-Lee, et al., 2001), comme une extension du web « *classique* » contenant non seulement des textes en langage naturel (français, espagnol, chinois, *etc.*) mais aussi de l’information formalisée et des services automatisés pour être traités automatiquement par la machine. En d’autres termes, le Web Sémantique est le *WWW* avec des capacités d’inférence (Manley, 2006). Il vise alors une meilleure maîtrise du contenu et non seulement de la syntaxe. L’ajout d’une sémantique explicite au contenu des ressources transforme le Web en une source globale de connaissances qui se révèle utile pour le partage et la réutilisation des données de différentes applications. L’essentiel du Web Sémantique ne consiste pas seulement à traiter « *intelligemment* » les applications, mais aussi à rendre les données plus «*intelligentes*» par l’utilisation des technologies de représentation de la sémantique telles que les ontologies (Daconta, et al., 2003). Cette structure enrichie par la sémantique contribue largement au développement de nouvelles approches plus flexibles pour l’intégration de données (Fensel, et al., 2002).

L’objectif du Web Sémantique consiste également à fournir un mécanisme de raisonnement sur les données avec moins d'intervention humaine. Sa syntaxe est fondée principalement sur l’utilisation des URIs (*Uniform Resource Identifier*) reliant les ressources et les termes dans un document à d'autres documents. Ainsi, les concepts d’un document ne consistent pas en une simple définition syntaxique, mais ils sont liés à des ressources partagées fournissant le contexte dans lequel ces concepts sont définis. Cette notion de liaison

[12] http://www.w3.org/

fournit le mécanisme de raisonnement par la déduction des faits sur un document, même si ces faits ne sont pas explicitement créés par l'auteur. Par conséquent, les technologies du Web Sémantique peuvent être utilisées pour l'échange des features des modèles CAO, permettant le partage et l'échange des données du produit de manière plus efficace et fiable. Or, si différents vocabulaires sont utilisés pour décrire les données d'un produit, l'ontologie fournit un mécanisme permettant de discerner l'équivalence des termes. Cela implique que les modèles peuvent éventuellement être partagés avec moins, voire sans intervention humaine et surtout avec moins de données perdues pendant le processus de transfert.

Le Web Sémantique se base principalement sur l'utilisation des langages standards permettant de décrire le contenu des ressources du web. Par exemple, le langage formel d'ontologies OWL (*Web Ontology Language*) a été conçu pour fournir des descriptions de haut niveau de ces ressources (Bechhofer, et al., 2004). OWL est un des langages les plus connus dans le domaine du Web Sémantique. Pourtant, OWL connait certaines limitations, dont la plupart concernent le développement de propriétés. Comme il n'y a pas de constructeur de composition de propriétés dans OWL, il n'est pas possible de saisir des relations composites des propriétés (O'Connor, et al., 2005). Dans ce cadre, des efforts ont été effectués sur l'augmentation de l'expressivité d'OWL par la définition des règles entre différents concepts. Ces efforts ont abouti au développement des langages de règles tels que SWRL (*Semantic Web Rule Language*). SWRL permet de définir des règles en fonction des concepts et des propriétés, afin d'effectuer du raisonnement sur les individus. La nécessité de règles dépend forcément de l'application utilisée intégrant une ontologie OWL. Dans la suite, nous allons présenter brièvement les ontologies et les langages des règles, notamment les langages utilisés dans notre approche OWL et SWRL.

3.2.1 Ontologie

Le terme « Ontologie » est emprunté à la philosophie qui implique une branche de la métaphysique traitant la nature et l'organisation de l'être[13]. Cependant, il est maintenant largement utilisé dans des domaines tels que la représentation des connaissances, la recherche d'informations et l'extraction de connaissances (Guarino, 1998). Ceci revient principalement au fait que les ontologies facilitent la représentation formelle de connaissances afin d'intégrer différents systèmes dans un environnement collaboratif (Ferreira da Silva, et al., 2006).

[13] Dictionnaire « Merriam-Webster ». Disponible en ligne sur : http://www.merriam-webster.com/

Plusieurs définitions d'ontologies ont été proposées dans la littérature, mais celle qui caractérise l'essentiel d'une ontologie dans l'intelligence artificielle est fondée sur une définition consensuelle fournie par *Tom Gruber* (Gruber, 1993): « *Une ontologie est une spécification formelle, et explicite d'une conceptualisation partagée* ». *Conceptualisation* réfère à un modèle abstrait de certains phénomènes dans le monde qui identifie les concepts appropriés de ce phénomène, ainsi que les relations entre ces concepts. *Explicite* signifie que le type de concepts utilisés et les contraintes sur leur utilisation doivent être explicitement définis. *Formelle* réfère au fait qu'une ontologie doit être compréhensible par la machine, c'est-à-dire que cette dernière soit capable d'interpréter la sémantique de l'information fournie. *Partagée* indique que l'ontologie supporte la connaissance consensuelle, et elle n'est pas restreinte à certains individus mais acceptée par un groupe (Broekstra, et al., 2002). Une ontologie sert donc à représenter les connaissances d'un domaine particulier par la spécification des différents concepts de ce domaine ainsi que leurs relations. Quel que soit le domaine, une ontologie se compose de plusieurs composants, les plus importants sont : les concepts, les relations, les attributs, les instances et les axiomes (Gruber, 1993).

Dans un environnement collaboratif, les ontologies jouent un rôle primordial dans l'interopérabilité des différents systèmes d'un domaine particulier en mettant en relief l'aspect sémantique d'intégration de données. Ainsi, l'interopérabilité sémantique ne peut être atteinte que lorsque les systèmes du domaine sont capables d'échanger des données de manière à ce que la signification précise des données soit facilement accessible et que ces données puissent être traduites dans un format compréhensible par d'autres systèmes (Heflin, et al., 2000). Un des problèmes fondamentaux de la réalisation de l'interopérabilité sémantique consiste à reconnaître que deux termes se réfèrent à la même entité, même si différentes terminologies sont utilisées. C'est à ce niveau que les ontologies peuvent être utilisées pour confronter cette problématique (Alesso, et al., 2005). Les ontologies sont ainsi développées pour fournir aux diverses applications un moyen de trouver des significations communes entre différents vocabulaires.

D'ailleurs, la sémantique des données doit être formellement représentée par une ontologie afin qu'elle soit interprétable par la machine. C'est un critère de base pour qu'une ontologie puisse jouer efficacement son rôle de représentation de connaissances. Les descriptions informelles conduisent aux ambigüités, ce qui entrave l'efficacité de représentation et d'intégration de données. En outre, les représentations informelles ne

peuvent pas être automatisées par les moteurs d'inférences puisqu'elles ne sont pas complètement interprétables par la machine. La sémantique représentée dans une ontologie peut être exprimée à l'aide de différentes syntaxes qui dépendent du langage de représentation. Différents langages ont été définis ayant chacune une syntaxe le définissant ainsi qu'une expressivité qui lui est associée. Dans la suite, nous citons brièvement quelques langages d'ontologies et nous consacrons une importante partie pour décrire le langage choisi pour notre approche, à savoir OWL, et plus précisément son sous-langage OWL DL basé sur les logiques de descriptions *LDs*.

3.2.1.1 Langages d'ontologies

Une ontologie peut être représentée formellement à l'aide d'un langage de représentation de connaissances. Les ontologies, leurs formalismes et leurs outils sont omniprésents dans les domaines de recherche du Web Sémantique, où plusieurs langages de représentation de l'ontologie ont été conçus et sont actuellement spécifiés et normalisés. Les langages les plus connus sont ceux basés sur le langage standard RDF (*Resource Description Framework*) ainsi que son extension RDFS (*RDF Schema*), qui ont été mis en œuvre par le consortium W3C. De plus de RDF(S), plusieurs autres normes ont été introduites comme DAML + OIL (Connolly, et al., 2001) ou OWL (Bechhofer, et al., 2004), qui étendent RDF(S) par un modèle plus riche en sémantique imposant certaines implications et des contraintes prédéfinies du langage.

En effet, un langage d'ontologie fournit un moyen de saisir et stocker la connaissance ainsi qu'un format pour représenter la connaissance acquise. Nous avons distingué plusieurs types de formalismes et de langages pour représenter une ontologie comme suit (Abdul Ghafour, 2004) :

a) *Les graphes*
   - *i.* Les réseaux sémantiques, Topic Maps[14]
   - *ii.* Orienté-Web : RDF et RDF Schéma

b) *Logique*
   - *i.* Du premier ordre : KIF
   - *ii.* Descriptive : KL-One, OIL, DAML+OIL, OWL

c) *Orienté objet :* UML + OCL

[14] http://www.topicmaps.org/

Le projet AIMS (Barkmeyer, et al., 2003) au NIST[15] classifie globalement les langages de représentation des ontologies en plusieurs catégories : les langages à base de frame tels que KL-One (Brachman, et al., 1985) et F-Logic (Angele, et al., 2004), les logiques des prédicats, les logiques de descriptions, et hybride. (Dartigues, 2003) ont regroupé les langages d'ontologies en plusieurs catégories : les langages orientés objets tels que UML, les langages à balise tels que XML, les langages basés sur une représentation graphique et textuelle tels que IDEF5 (Benjamin, et al., 1994), et les langages basés sur la logique du premier ordre tels que KIF.

KIF (*Knowledge Interchange Format*) est un langage basé sur les prédicats du premier ordre avec des extensions pour représenter des définitions et des méta-connaissances (Genesereth, et al., 1992). Basé sur le langage KIF, l'outil Ontolingua (Farquhar, et al., 1997) permet aux utilisateurs de construire des ontologies KIF à un niveau plus élevé de la description par l'importation des ontologies prédéfinies. Cependant, la représentation des ontologies à l'aide des langages basés sur la logique du premier ordre n'est pas décidable : il n'existe pas d'outils qui permettent de vérifier que l'ontologie ne contient pas d'inconsistances ou d'incohérences (Dartigues, 2003).

Les langages basés sur les logiques de descriptions correspondent à des langages puissants en termes d'expressivité de représentation de connaissances. Ce sont des langages appropriés à la spécification des concepts et leurs hiérarchies (Baader, et al., 2003). Ils permettent la définition des concepts par des descriptions qui déterminent les critères d'appartenance des instances à des classes de concepts selon leurs propriétés définies. Les logiques de descriptions définissent également un ensemble de constructions de description utilisées pour la définition des nouveaux concepts et rôles, tels que la conjonction ($\sqcap$), la disjonction ($\sqcup$), la négation ($\neg$), les quantificateurs ($\forall, \exists$) *etc*. En effet, la logique descriptive correspond à un sous ensemble de la logique du premier ordre en restreignant la syntaxe de certaines formules (Borgida, 1996), par exemple en n'ayant que des prédicats unaires (*concepts*) et binaires (*rôles*). Néanmoins, cette restriction affecte leur expressivité.

[15] National Institute of Standards and Technology http://www.nist.gov/index.html

KL-ONE est un langage basé sur les logiques de descriptions (Baader, et al., 2007). Il est une formalisation de représentation de la connaissance à base de cadres[16] « *frame-based* ». Ce système maintient la définition des concepts par un simple nommage, et l'indication de la correspondance des concepts dans une hiérarchie de généralisation/spécialisation. De nouveaux termes peuvent être définis par des opérations de conjonction des concepts. Par exemple l'opérateur « *and* » peut être utilisé pour préciser qu'un nouveau concept est une spécialisation commune de plusieurs autres concepts. De nouveaux rôles peuvent être introduits pour représenter les relations qui peuvent exister entre des individus dans le domaine modélisé. Les définitions des concepts peuvent inclure des restrictions sur les valeurs possibles, sur les nombres de valeurs, ou sur le type de valeurs qu'un rôle peut avoir pour un concept.

Dans le cadre des langages développés pour la représentation des données du WWW, *Berners-Lee* (Berner-Lee, et al., 2001) a proposé une architecture en couches d'un ensemble de technologies du Web Sémantique (Figure 16).

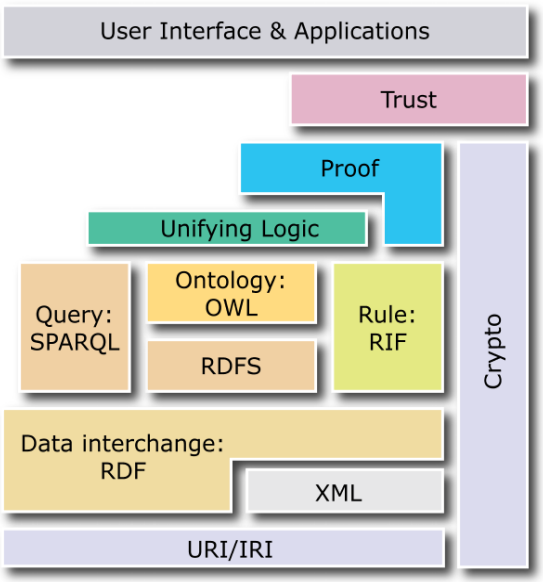


Figure 16. Architecture des technologies du Web Sémantique (Berner-Lee, et al., 2001)

Cette architecture est composée d'une modèle en couches de protocoles qui sont construits allant de la couche syntaxique, à travers les couches sémantiques et logiques, jusqu'aux couches du haut niveau telles que Trust. Au premier niveau de cette figure nous

[16] Un cadre (frame) est un objet nommé représentant un concept générique. Il est doté d'attributs (slots), qui peuvent posséder différentes valeurs (facets), et est destiné à être instancié. F-LOGIC est l'exemple le plus connu de langage à base de cadres (frames).

trouvons les URIs qui permettent d'identifier les ressources (sujet ou, éventuellement, objet d'un triple) et les prédicats. RDF est le modèle de base pour l'échange des données et peut éventuellement s'appuyer sur XML, dans le cas de l'utilisation de la syntaxe RDF/XML. Au-dessus, on trouve SPARQL pour effectuer des requêtes, RDF-S et OWL pour définir des vocabulaires RDF, assimilables à des ontologies et RIF, un langage de définition de règles. Les couches RIF (*Rule Interchange Format*), *Unifying logic*, *Proof* et *Trust* ne connaissent pas encore de standards stabilisés.

XML (*eXtensible Markup Language*) (Bray, et al., 2000), recommandé du consortium W3C[17] depuis le 10 février 1998, est un langage qui vise à décrire l'information de manière structurée et est adapté au stockage et à l'échange de données descriptives. XML est un méta-langage à balises qui fournit une syntaxe pour les documents structurés hiérarchiquement, i.e. dans un arbre, mais n'impose aucune contrainte sémantique à la signification de ces documents. Un exemple simple de XML permettant de décrire une personne est illustré dans la figure 17 (Lacot, 2005).

```
<?xml version="1.0" encoding="UTF-8"?>
<personne>
      <nom>Dupond</nom>
      <prenom>Jean</prenom>
      <naissance>
            <lieu>
                  <ville>Paris</ville>
                  <pays>France</pays>
            </lieu>
            <date>
                  <jour>14</jour>
                  <mois>7</mois>
                  <annee>1789</annee>
            </date>
      </naissance>
</personne>
```

Figure 17. Exemple en XML pour la description d'une personne (Lacot, 2005)

RDF (Resource Description Framework) (Brickley, et al., 2003) est un formalisme graphique pour représenter des méta-données. Il est basé sur la notion de triplet (sujet, prédicat, objet), illustré dans la figure 18 (Lacot, 2005). Le sujet et l'objet sont des ressources liées par le prédicat. Ceci permet de représenter les ressources comme des graphes, dont le sujet et l'objet sont des nœuds ; le prédicat est l'arc qui relie ces nœuds, dirigé du nœud sujet vers le nœud objet. RDF utilise la syntaxe XML, mais il ne donne aucune signification

[17] http://www.w3.org

spécifique pour le vocabulaire comme *sous-classe* ou *type*. Les primitives de modélisation offertes par RDF sont très basiques, ce qui a conduit au développement du RDF-S.

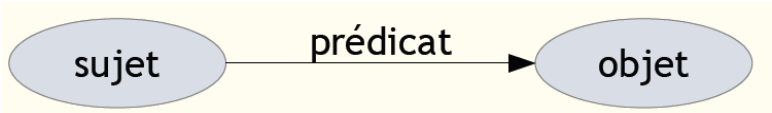


Figure 18. Modèle de Graphe de données (Lacot, 2005)

RDF-S (*RDF Schema*) (Brickley, et al., 2003) étend RDF par un langage de définition de vocabulaires pour la description de propriétés et de classes représentées par des ressources RDF. RDF-S fournit une structure et une sémantique du langage pour spécifier les types de classes et de propriétés telles que *Class*, *Property*, *type*, *subClassOf*, *subPropertyOf*, *range* et *domain*. Ainsi, RDFS permet de définir des graphes de triplets RDF, avec une sémantique de généralisation/hiérarchisation de ces propriétés et de ces classes. Cependant, RDF et RDF-S ont certaines limitations en termes de représentation de la cardinalité et des caractéristiques d'une propriété telles que la symétrie, la transitivité, etc. En outre, Le langage RDF-S n'est pas adapté pour effectuer des tâches de raisonnement déductif.

3.2.1.2 Langage OWL

Le langage OWL (*Web Ontology Language*), basé sur la recherche effectuée dans le domaine de la logique de description, est recommandé par le W3C depuis le 10 février 2004 pour la définition des ontologies pour le Web Sémantique (McGuinness, et al., 2004). Il est le fruit des efforts effectués aux Etats Unis et à l'Union Européenne et a été créé à partir de son prédécesseur, le langage DAML + OIL (*DARPA Agent Markup Language + Ontology Inference Layer*) (Connolly, et al., 2001). OWL permet de définir des terminologies pour décrire des domaines concrets. Une terminologie se constitue de concepts et de propriétés (aussi appelées rôles en logiques de description). Un domaine se compose des instances de concepts.

OWL vise principalement à faciliter la représentation et le traitement automatique du contenu du web en fournissant, par rapport aux langages XML, RDF et RDF-S, du vocabulaire supplémentaire et une sémantique formelle, permettant la création des relations sémantiques entre les concepts. Ainsi, OWL permet de représenter explicitement le sens des termes et les relations entre ces termes. OWL réutilise donc certains constructeurs de RDF/RDFS tels que *rdfs:subClassOf*, *rdfs:domain*, et *rdfs:range* , mais ajoute aussi un

vocabulaire supplémentaire afin de représenter les combinaisons logiques entre classes (l'intersection, l'union, et le complément), et la spécification de certaines caractéristiques des propriétés (symétrie, transitivité, *etc*.), des restrictions d'appartenance à une classe, des contraintes de cardinalité, des classes énumérées, et des relations d'équivalence entre les classes ou les propriétés ou des relations de disjonction entre les classes.

OWL vise également à rendre les ressources sur le web aisément accessibles aux processus automatisés (McGuinness, et al., 2004), d'une part en les structurant d'une façon compréhensible et standardisée en tirant parti de la structuration balisée de la syntaxe XML, et d'autre part en leur ajoutant des méta-informations. Pour cela, OWL dispose des moyens plus puissants pour exprimer la signification et la sémantique que XML, RDF, et RDF-S. De plus, OWL considère l'aspect diffus des sources de connaissances et permet à l'information d'être recueillie à partir de sources distribuées, notamment en permettant la mise en relation des ontologies et l'importation des informations provenant explicitement d'autres ontologies (Smith, et al., 2004).

3.2.1.2.1 Sous Langages d'OWL

OWL permet, grâce à sa sémantique formelle basée sur une fondation logique largement étudiée, de définir des associations plus complexes des ressources ainsi que les propriétés de leurs classes respectives. OWL définit trois sous-langages, du moins expressif au plus expressif : OWL-Lite, OWL-DL et OWL-Full. Chacun de ces sous-langages est une extension de son prédécesseur, par rapport à ce qui peut être exprimé et ce qui peut être déduit.

- **OWL Lite** : est le sous langage de OWL le plus simple. Il supporte seulement un sous-ensemble de constructeurs du langage OWL et a une complexité formelle inférieure à celle de OWL DL. Il est destiné aux utilisateurs ayant besoin principalement d'une hiérarchie de classification et des contraintes simples. Par exemple, il permet les valeurs de cardinalité 0 et 1 seulement, interdit l'utilisation des connecteurs d'union et de complément, la description d'une classe à travers l'énumération de ses individus (avec le constructeur *owl:oneOf*) ou l'énumération de types de données. OWL Lite correspond à la variante de la logique de description $SHIF(D)$ (Horrocks, et al., 2003) dont la complexité temporelle est

dans le pire des cas exponentielle déterministe ($ExpTIME$). OWL Lite est adapté, par exemple, aux migrations rapides depuis d'anciens thésaurus.

- **OWL DL** : OWL DL se base, comme l'indique son nom, sur la logique de description, DL. Il est destiné aux utilisateurs qui réclament une expressivité maximale tout en retenant toutefois la complétude computationnelle des raisonnements (*i.e.* toutes les références sont garanties d'être calculables), et leur décidabilité (*i.e.* leur calcul se fait en une durée finie). Il inclut tous les constructeurs du langage OWL, qui ne peuvent être utilisés que sous certaines restrictions, telles que la disjonction des classes, des propriétés, des individus, et des valeurs de données. OWL DL correspond à la variante de la logique de description $SHOIN(D)$ (Horrocks, et al., 2003) dont la complexité temporelle est dans le pire des cas exponentielle non-deterministe (*NExpTIME*).
- **OWL Full** : OWL Full est la version la plus complexe de OWL, mais également celle qui permet le plus haut niveau d'expressivité. OWL Full est destiné aux situations où il est plus important d'avoir un haut niveau de capacité de description, quitte à ne pas pouvoir garantir la complétude et la décidabilité des calculs liés à l'ontologie. OWL Full et OWL DL maintiennent le même ensemble de constructeurs du langage OWL. La différence réside dans les restrictions d'utilisation sur certaines fonctionnalités et par l'emploi de fonctionnalités de RDF. D'ailleurs, OWL Full permet un mélange libre du langage OWL avec RDFS et, comme RDFS, n'impose pas une séparation stricte des classes, des propriétés, des individus, et des valeurs de données. Il est peu probable que n'importe quel logiciel de raisonnement soit capable de supporter le raisonnement complet de chaque caractéristique du langage OWL Full. Autrement dit, en utilisant OWL Full en comparaison avec OWL DL, le support de raisonnement est moins prévisible puisque l'implémentation complète du langage OWL Full n'existe pas actuellement.

Par ailleurs, il existe entre ces trois sous-langages une dépendance de nature hiérarchique: toute ontologie OWL Lite valide est également une ontologie OWL DL valide, et toute ontologie OWL DL valide est également une ontologie OWL Full valide. D'ailleurs, toutes les ontologies OWL (Lite, DL, et Full) sont valides en RDF. En revanche, OWL Full est considéré comme étant une extension complète de RDF où tout document RDF l'est aussi

en OWL Full, tandis que OWL Lite et OWL DL peuvent être vus comme des extensions d'une vue restreinte de RDF (par exemple, distinction des classes, des propriétés, etc.).

Une extension du langage OWL DL a été développée, connue sous le nom OWL 1.1 (Patel-Schneider, et al., 2006). Cette extension, orientée axiome, est basée sur la variante $SROIQ$ de la logique descriptive. Ainsi, des nouveaux constructeurs sont ajoutés, tels que les restrictions de cardinalités qualifiées ($\geq n\ R.C$ et $\leq n\ R.C$), autrement dit les cardinalités sur des concepts autres que $owl{:}Thing$. De plus, OWL 1.1 permet l'inclusion des rôles complexes, ainsi que la définition des caractéristiques supplémentaires des rôles, telles que la réflexivité et la disjonction des rôles. La représentation syntaxique de OWL 1.1 est fondée sur les axiomes, qui est plus simple à traiter et moins verbeuse que les représentations RDF ou celles fondées sur les frames (Ferreira Da Silva, 2007). Une syntaxe dans le style « frames » est par exemple $Class(CLsA\ partial\ CLsB)$ est équivalente dans la représentation orientée axiomes à $SubClassOf(CLsA\ CLsB)$.

Dans notre travail, nous nous focalisons sur le langage OWL DL parce que ceci est le sous-langage qui permet de spécifier une ontologie d'expressivité maximale tout en restant traitable par un moteur d'inférences. Cependant, OWL DL a des limitations, et nous avons recours aux langages de règles tels que SWRL présenté dans la section suivante.

3.2.2 Le langage des règles SWRL

Bien que l'expressivité du langage OWL accomplisse considérablement des fonctions du Web Sémantique, ce langage a pourtant ses propres limitations surtout en ce qui concerne les capacités de raisonnement déductif (O'Connor, et al., 2005). D'autres limitations du langage OWL sont liées à la description des propriétés dans une ontologie OWL. En l'occurrence, comme il n'ya pas de constructeur de composition pour les propriétés, la saisie des relations entre des propriétés composites n'est pas actuellement possible (Horrocks, et al., 2005).

Cependant, les travaux effectués pour surmonter ces limites ont conduit à l'élaboration du langage des règles du Web Sémantique SWRL (*Semantic Web Rule Language*). SWRL est basée sur une combinaison des sous langages OWL DL et OWL Lite avec le sous langage unaire/binaire du langage RuleML (*Rule Markup Language*) (Horrocks, et al., 2004).

La structure des règles SWRL est composée d'un antécédent ou corps (*body*) et d'un conséquent ou tête (*head*). Effectivement, une règle signifie « *Si les conditions de l'antécédent sont maintenues, alors les conditions du conséquent doivent également se tenir* ». La tête et le corps d'une règle sont composés d'une conjonction d'un ou de plusieurs atomes.

$$\boldsymbol{If}\ Antecedent\ \boldsymbol{Then}\ Consequent$$

$$Antecedent\ (\boldsymbol{Body}) \rightarrow Consequent(\boldsymbol{Head})$$

En effet, SWRL intègre une syntaxe abstraite de haut niveau pour les règles à base de Horn (*Horn-Like Rules*) dans les sous langages OWL DL et OWL Lite de OWL. Ainsi, SWRL étend les logiques de descriptions et les axiomes du langage OWL avec la possibilité de création des règles utilisées pour augmenter les capacités de raisonnement déductif. Le raisonnement permet de déduire des nouvelles connaissances à partir des ontologies OWL (O'Connor, et al., 2005 - b). Par exemple, la création d'une règle SWRL exprime que si une variable X a comme père la variable Y, et si Y a comme frère la variable Z alors X a comme oncle Z. Cette règle exige l'existence des propriétés « *aPere*», « *aFrere* », et « *aOncle* » dans l'ontologie OWL. Cette règle est exprimée en SWRL comme suit :

$$aPere\ (?X, ?Y) \wedge aFrere(?Y, ?Z) \rightarrow aOncle(?X, ?Z)$$

L'exécution de cette règle permet au raisonneur de déduire des relations du type « *aOncle*» entre les instances définies dans la base de connaissances représentées par l'ontologie.

D'ailleurs, une autre fonctionnalité puissante du langage SWRL réside dans sa capacité à supporter une large gamme de compléments définis par l'utilisateur appelés aussi *builts-ins*[18]. Ces compléments permettent d'étendre significativement l'expressivité du langage SWRL. Certains compléments sont basés sur des fonctions de comparaison (*equal*, *lessThan*, *etc*.), des fonctions mathématiques (*add*, *multiply*, *sin*, *etc*.), des fonctions booléennes (*booleanNot*), des fonctions de traitement des chaines de caractères (*stringLength*, *substring*, *etc*.) et d'autres fonctions[19]. L'exemple suivant montre l'utilisation d'une règle SWRL avec le complément de comparaison *greaterThan* permettant de déterminer si un

[18] SWRL Built-ins: http://www.daml.org/2004/04/swrl/builtins

[19] Une liste détaillée des spécifications des compléments est disponible sur la page: http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/#8

individu a un frère plus âgé. Notons l'utilisation de l'espace de nommage « *swrlb* » pour tous les compléments.

$$hasBrother\ (?x_1, ?x_2) \wedge hasAge(?x_1, ?age_1) \wedge hasAge(?x_2, ?age_2) \\ \wedge swrlb{:}\ greaterThan(?age_2, ?age_1) \rightarrow hasOlderBrother(?x_1, ?x_2)$$

3.2.3 Le raisonnement basé sur les logiques descriptives

Un avantage significatif dans la représentation des connaissances avec une ontologie OWL DL réside dans les capacités de raisonnement fournies par un raisonneur. Par comparaison avec les bases de données classiques BD, l'hypothèse du monde ouvert implique que répondre à une requête par un système bâti sur les LDs nécessite d'effectuer un raisonnement logique souvent plus complexe qu'une simple recherche pour vérifier la présence d'une information, car le système doit souvent considérer plusieurs interprétations possibles (Ferreira Da Silva, 2007). Par conséquent, l'absence d'une information dans une BD implique systématiquement que cette information est négative, tandis que dans un système LD, elle est interprétée par une manque de connaissances (Baader, et al., 2003). Un raisonneur permet entre autres de fournir le test de subsomption pour déterminer la hiérarchie des classes (superclasses ou sous-classes) entre les différents concepts de l'ontologie. De plus, un raisonneur permet de vérifier la consistance d'une ontologie au fur et à mesure de sa construction.

Généralement, une ontologie typique est composée d'une taxonomie de classes et un ensemble de règles d'inférence. Une règle décrit une conclusion que l'on en tire à partir d'une condition. Il peut s'agir d'une déclaration traitée par un raisonneur ou un moteur d'inférence qui peut générer une inférence à partir d'une règle définie. Avec l'utilisation des règles dans une base de connaissances, les moteurs d'inférences peuvent déduire des nouvelles connaissances générées à partir des connaissances déjà existantes. Ainsi, il est maintenant possible que la machine puisse manipuler les informations d'une manière plus significative. Dans notre travail, nous nous intéressons au raisonnement effectué dans le cadre des ontologies basées sur les logiques de descriptions telles que celles représentées avec le langage OWL DL.

3.2.3.1 Notions de base

Dans le formalisme des logiques de descriptions, un concept représente un ensemble d'individus ayant des caractéristiques communes, tandis qu'un rôle représente une relation binaire entre les individus. Ainsi, un concept correspond à une entité générique et un individu à une entité particulière, autrement dit instance d'un concept (Napoli, 2005). Un concept et un rôle sont définis par une description structurée, élaborée à partir d'un certain nombre de constructeurs de description comme l'intersection ($\sqcap$), l'union ($\sqcup$), la négation ($\neg$), et les quantificateurs ($\forall, \exists$).Ces constructeurs permettent la définition des nouveaux concepts et rôles complexes en fonction d'entités plus simples.

Les concepts peuvent être primitifs ou définis. Un concept primitif ou partiel ($C \sqsubseteq D$) sert de base pour la construction des concepts définis. Sa description contient seulement des « *conditions nécessaires* ». Selon une condition nécessaire, si un individu appartient à une classe, alors il doit satisfaire cette condition. D'autre part, un concept est défini ou complet ($C \equiv D$) si sa description contient au moins un ensemble de conditions nécessaires et suffisantes. Par conséquent, ces conditions sont suffisantes pour déterminer que n'importe quel individu qui satisfait ces conditions doit être une instance ou une extension de la classe définie. Il est également important de comprendre que le raisonneur peut classifier uniquement les classes définies.

D'ailleurs, les connaissances en LDs sont généralement définies à deux niveaux (Baader, et al., 2003) :

- Le niveau terminologique ($TBox$): C'est à ce niveau que s'effectuent la représentation et la manipulation des concepts et des rôles. Ainsi, des axiomes terminologiques d'inclusion ($\sqsubseteq$) ou d'égalité ($\equiv$) peuvent être définis entre des concepts ou des rôles, par exemple $C \sqsubseteq D$ ou $C \equiv D$ pour deux concepts C et D, et $R \sqsubseteq S$ $(R \equiv S)$ pour deux rôles R et S.
- Le niveau factuel ou des assertions ($ABox$): la description et la manipulation des individus relèvent du niveau factuel qui représente des instances du ($TBox$). Par exemple $C(a)$ et $R(a, b)$ représentent respectivement l'assertion du concept C et du rôle R.

3.2.3.2 Les types d'inférences basées sur les LDs

Selon (Napoli, 2005), les opérations qui sont à la base du raisonnement sont la classification et l'instanciation :

- Classification : cette opération permet de déterminer la position d'un concept ou d'un rôle dans leurs hiérarchies.
- Instanciation : Elle permet de retrouver les concepts dont un individu est susceptible d'être une instance.

Dans un système de représentation de connaissances basé sur la logique descriptive, différents types de raisonnement peuvent être effectués. Dans (Baader, et al., 2003), quatre types d'inférences peuvent être distingués au niveau terminologique $(TBox)$:

- **Subsomption** : Un concept C subsume un concept D, ou D est subsumé par C, $D \sqsubseteq C$, si C est plus général que D au sens où l'ensemble d'individus représenté par C contient l'ensemble d'individus représenté par D. Autrement dit, pour chaque interprétation I, $D^I \subseteq C^I$
- **Equivalence** : Un concept C est équivalent à un concept D si et seulement si pour chaque interprétation I, $C^I = D^I$. Ceci peut être interprété par une subsomption bidirectionnelle, c'est-à-dire $C^I \subseteq D^I$ et $D^I \subseteq C^I$.
- **Satisfiabilité** : Un concept C est satisfiable s'il existe au moins une interprétation I, tel que $C^I \vDash \top$. Lors de la modélisation des nouveaux concepts, il est particulièrement utile de vérifier si le nouveau concept est cohérent dans la base de connaissances.
- **Disjonction** : Deux concepts C et D sont disjoints si et seulement si pour toute interprétation I, $C^I \cap D^I \vDash \perp$

Les trois derniers types de raisonnement, notamment l'équivalence, la satisfiabilité et la disjonction peuvent être exprimés en fonction du premier type « la subsomption » (Baader, et al., 2003). Bien que la logique descriptive soit moins expressive que la logique du premier ordre, la décidabilité et la traçabilité de ces services d'inférence l'ont rendue largement utilisée comme outil efficace de représentation des ontologies pour l'interopérabilité sémantique. Ces capacités de raisonnement peuvent être exploitées dans la représentation des connaissances des modèles CAO, par exemple, afin de traduire les informations d'un produit, ainsi que la sémantique associée, entre différentes ontologies de systèmes CAO.

3.3 Proposition d'échange basée sur l'ontologie

Dans la section 3.2 de ce chapitre, nous avons présenté le Web Sémantique et ses technologies, notamment les ontologies et les langages de règles, qui permettent d'améliorer l'interopérabilité des systèmes d'un domaine particulier tenant compte de la sémantique associée aux données. Il s'agit donc de représenter formellement cette sémantique où le contexte des données doit être considéré et interprété (Dartigues, et al., 2007). En effet, une même information pourrait être représentée par différents concepts et terminologies, et inversement, un même terme utilisé dans différents contextes pourrait désigner différents concepts. Pour faire face aux limites des solutions actuelles, notre objectif consiste à définir une approche ontologique d'échange de données par la traduction de la sémantique associée aux données d'un produit. Cette approche repose principalement sur les avancées technologiques dans le domaine de la représentation des connaissances et du Web Sémantique pour le partage et l'échange des données des systèmes CAO. Notre méthode consiste principalement à décrire formellement les concepts liés au domaine de la conception par features, et leurs propriétés ainsi que les relations sémantiques existantes entre ces différents concepts. Cette notion de description des connaissances des modèles CAO est illustrée dans la figure 19.

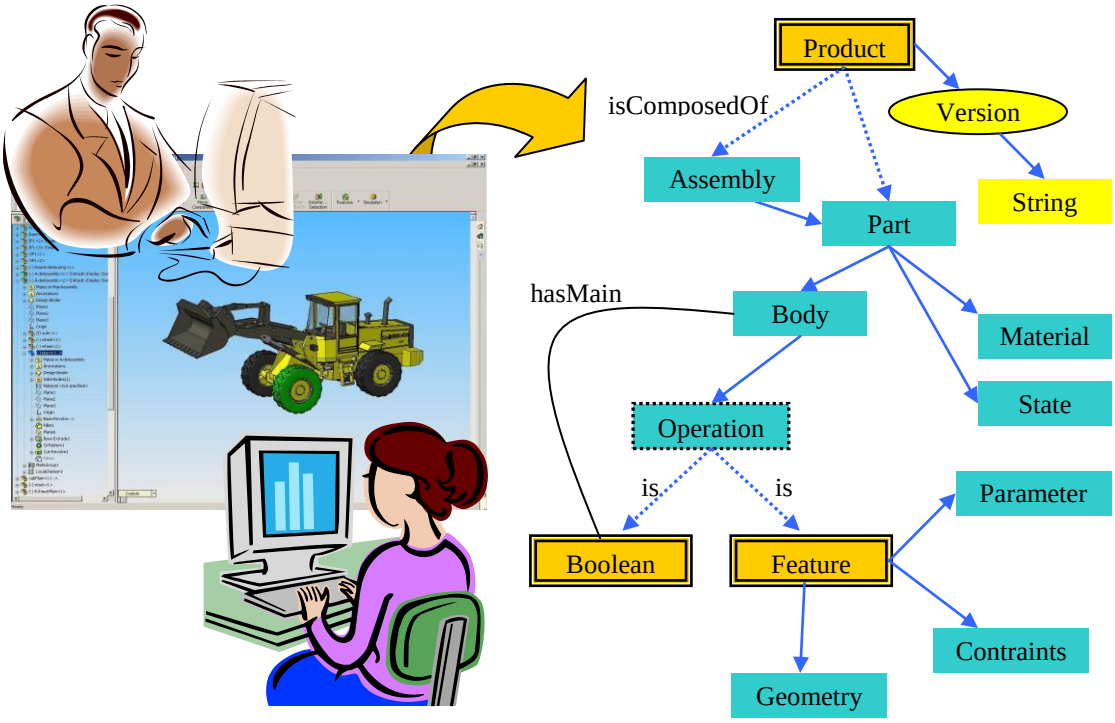


Figure 19. Approche ontologique d'échange de modèles à base de features

Par conséquent, la représentation des connaissances des modèles CAO moyennant l'ontologie rend la sémantique du domaine à la fois explicite et accessible pour l'interprétation par les machines. De plus, ces connaissances explicites peuvent être exploitées par des algorithmes de raisonnement pour en déduire de la connaissance implicite, c'est-à-dire des déclarations qui n'ont pas été explicitement modélisées. Dans la section suivante, nous décrivons notre proposition d'échange de données basée sur les ontologies.

3.3.1 Description de l'approche ontologique

L'approche que nous proposons se base sur la méthode d'échange indirect de données, autrement dit, basée sur l'utilisation d'un format neutre entre les différentes applications. Dans ce cadre, une étape fondamentale de notre approche consiste à développer une ontologie commune de features, que nous avons appelée CDFO (*Common Design Features Ontology*), servant d'intermédiaire entre les différents systèmes CAO à base de features. Toutefois, la définition et le développement de notre ontologie commune seront détaillés dans le chapitre 4.

Cette ontologie commune est conçue pour représenter des informations spécifiques liées à l'intention de conception du produit. Elle contient un ensemble de features de conception utilisé dans la plupart des systèmes actuels de CAO. Ceci permettra aux concepteurs de communiquer avec le système collaboratif à un niveau d'abstraction élevé tel que les trous, les poches, les rainures, plutôt que les entités géométriques et topologiques de bas niveau telles que les sommets, les arêtes et les faces. En conséquence, pour se rendre interopérable avec les autres applications, chaque application CAO nécessite simplement un convertisseur entre la terminologie de l'application et l'ontologie commune des features.

Afin de parvenir à la réalisation de l'interopérabilité sémantique des modèles de produit, nous avons développé une méthodologie d'échange indirect illustrée dans la figure 20. Comme le montre cette figure, nous considérons l'exemple d'échange entre deux systèmes CAO à base de features *X* et *Y*. Chaque système contient une librairie de features de conception ayant sa propre syntaxe, une terminologie associée à sa représentation, ainsi qu'une sémantique définie dans son ontologie spécifique. Ces ontologies spécifiques aux systèmes CAO, que nous appelons « *ontologie d'application* », contiennent des connaissances liées à chaque système et peuvent exister sous différents formats, par exemple sous forme de fichiers textuels, fichiers en langage XML, diagrammes UML, *etc*. Cependant, la définition des ontologies d'application présente un défi important lié au fait que pour la plupart des

systèmes actuels de CAO, il n'existe pas d'ontologie explicitement définie pour chaque système. Pour la mise en œuvre de notre approche, nous avons surmonté ce problème par une analyse détaillée de plusieurs systèmes CAO. Cette étape est indispensable pour l'extraction de la sémantique représentée implicitement dans l'API, par exemple, ou dans la documentation du système. Elle conduit, en conséquence, à la construction d'une ontologie explicite spécifique au système CAO. L'objectif essentiel de l'ontologie d'application est d'avoir une représentation explicite des concepts utilisés dans un système CAO ainsi que leurs relations. Il s'agit donc de représenter non seulement les classes des features définis dans un système, mais aussi leurs attributs, les contraintes et les relations existantes entre ces concepts.

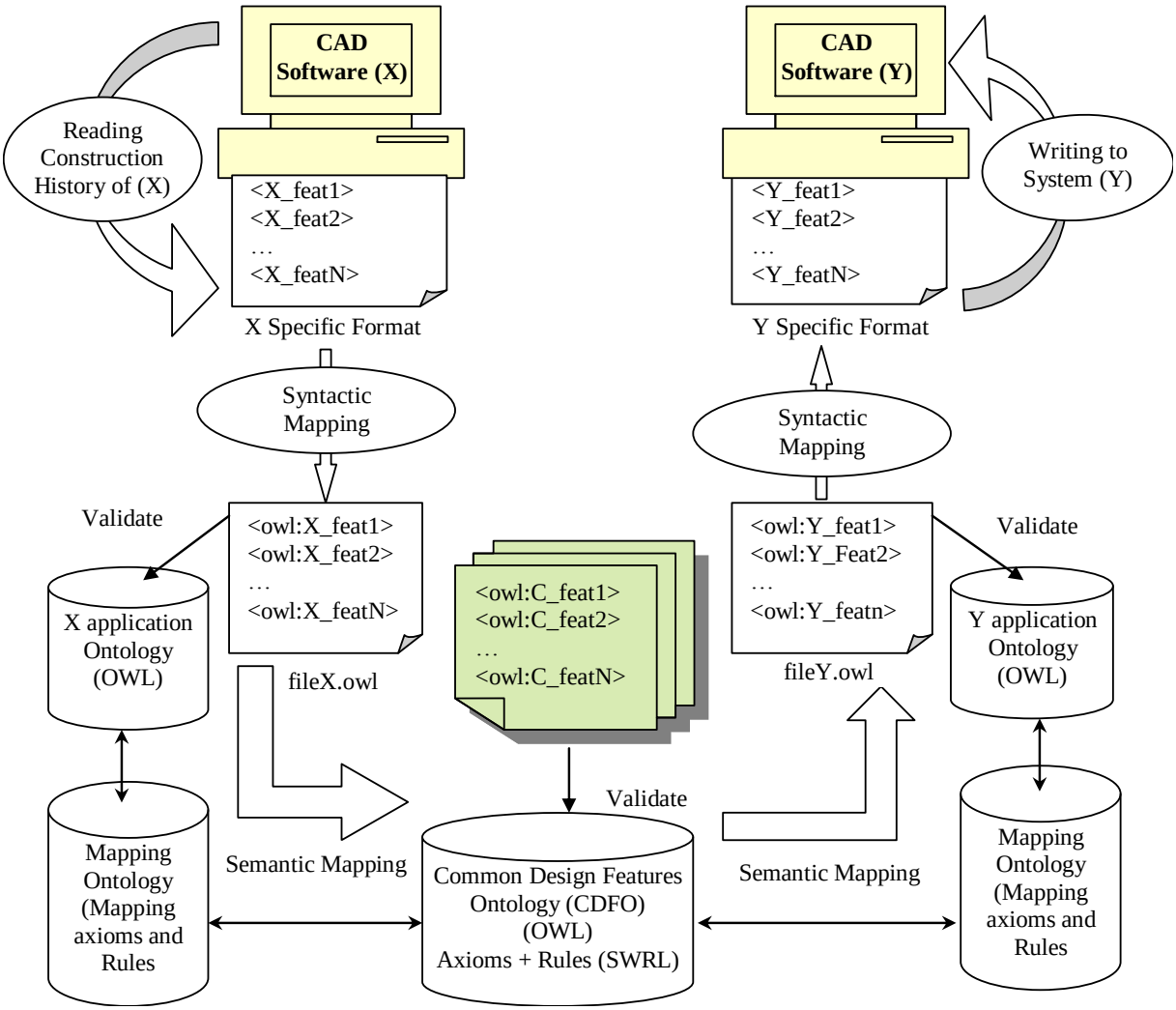


Figure 20. Méthodologie de notre proposition d'échange des modèles CAO

3.3.2 Méthodologie de notre proposition d'échange

La méthodologie d'échange des modèles CAO entre deux systèmes X et Y est décrite de la façon suivante (illustrée dans la figure 20) :

- **Niveau de l'application CAO** : les concepteurs créent des modèles procéduraux de produits à l'aide de la modélisation par features. En effet, les applications actuelles de conception fournissent une vaste gamme d'interfaces riche en fonctions d'ingénierie visant à la création et la manipulation des instances des features dans les modèles conçus. Les modèles créés sont ensuite sauvegardés dans des fichiers ayant des formats spécifiques à leurs applications. Ces fichiers doivent maintenir l'ensemble des entités de haut niveau du modèle, telles que l'historique de construction, les instances des features, les paramètres et leurs contraintes. Cette base de données d'un fichier peut être accessible par le biais de l'API de l'application qui permet l'extraction des informations nécessaires des modèles procéduraux.
- **Niveau de l'interopérabilité** : cette phase consiste à mettre en place un processus de conversion des modèles CAO provenant des différents systèmes vers l'ontologie commune CDFO. Ceci est réalisé par le développement d'un mécanisme traitant les problèmes d'hétérogénéité entre les différents formats, notamment aux niveaux syntaxique et sémantique. Ainsi, notre proposition consiste, dans un premier temps, à homogénéiser les représentations syntaxiques des formats par la conversion en un langage pivot, en l'occurrence le langage d'ontologie OWL. Les modèles CAO résultants représentent donc des instances des ontologies d'application en OWL. Ensuite, nous procédons d'une part à l'intégration sémantique entre les ontologies d'application et d'autre part à l'ontologie commune CDFO. Ceci se traduit par le développement des règles de correspondance sémantique, établies dans une ontologie de correspondance « *Mapping Ontology* », pour prendre en considération les problèmes d'hétérogénéité sémantique entre les différentes ontologies OWL. Ainsi, une caractéristique essentielle de notre ontologie réside dans son extensibilité afin de pouvoir ajouter des concepts supplémentaires liés aux systèmes CAO non représentés dans notre ontologie. Ce traitement à deux niveaux d'hétérogénéités est détaillé dans la section suivante.

3.3.3 Traitement d'hétérogénéités des formats

La diversité des formats de représentation des connaissances entre les ontologies d'application et l'ontologie commune pose différents problèmes d'hétérogénéité. En effet, nous constatons que les types d'hétérogénéité entre les ontologies sont classés en deux grandes catégories : les discordances liées à la syntaxe, et celles liées à la sémantique (Ferreira da Silva, et al., 2006). Ces dernières sont beaucoup plus complexes, et leur résolution nécessite beaucoup de travaux de recherche.

Ces problèmes d'hétérogénéités affectent le processus de communication, et par conséquent entravent l'échange des données entre les différentes applications. En partant de l'expressivité riche du langage OWL, surtout au niveau de la représentation de la sémantique, (*cf.* section 3.2.1.2) et afin de traiter les différents problèmes posés par les discordances des formats, nous avons proposé la mise en place d'un processus d'échange basé essentiellement sur les deux étapes suivantes : homogénéisation syntaxique *(Syntactic Mapping)* et mise en correspondance sémantique (*Semantic Mapping)* (Abdul Ghafour, et al., 2007). Ces deux étapes sont mises en relief dans la figure 21.

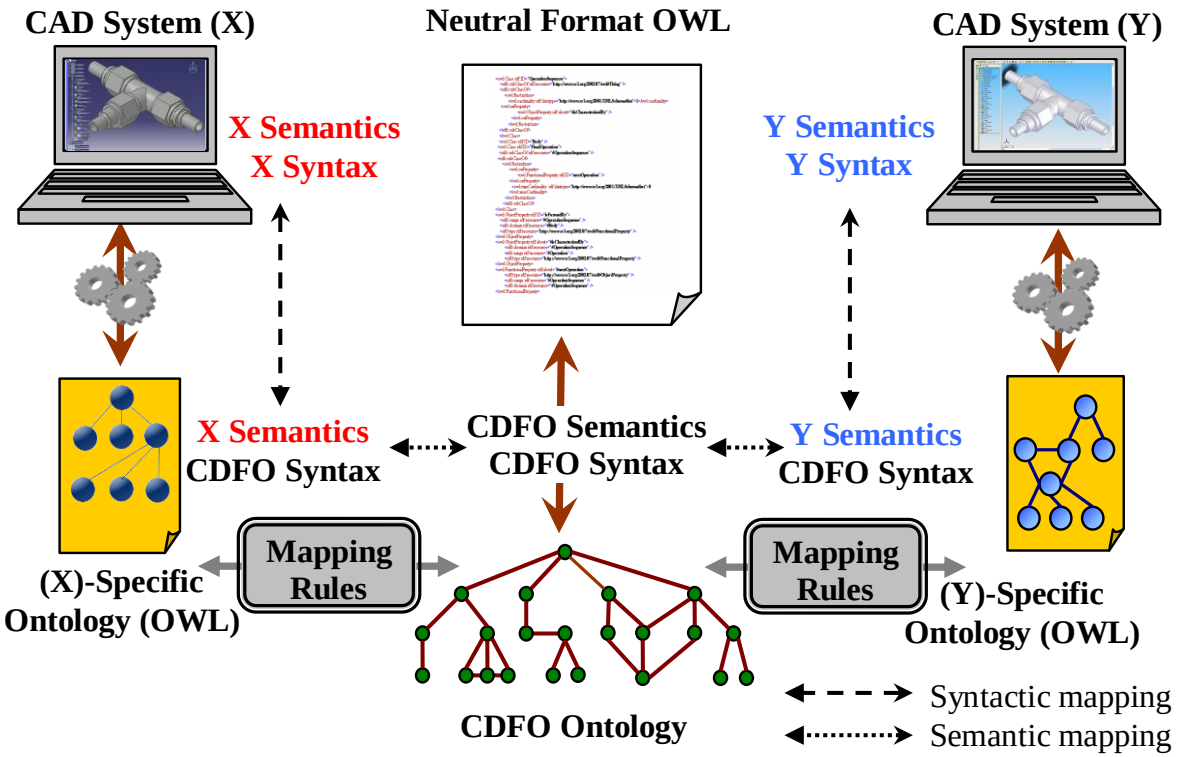


Figure 21. Processus de traitement d'hétérogénéités syntaxique et sémantique

3.3.3.1 Homogénéisation syntaxique

Les ontologies des différentes applications peuvent exister sous des formats hétérogènes de représentation. Le mécanisme de conversion présenté dans cette étape vise à homogénéiser ces différents formats en les convertissant vers un format pivot, en l'occurrence OWL DL. Ceci permet de résoudre les discordances syntaxiques entre ces différentes ontologies.

Notre choix du langage OWL DL repose sur plusieurs raisons. D'abord, la structure du langage OWL permet de définir des concepts sous forme de classes et de relations sémantiques explicites entre ces concepts. La syntaxe du langage OWL est basée sur le langage XML. Ainsi, les descriptions codées avec ce langage peuvent être exploitées par des procédures automatisées de recherche et d'analyse du contenu. De plus, ce langage est largement utilisé dans les domaines de représentation des connaissances en raison de son expressivité riche pour la représentation de la sémantique et de ses capacités d'inférence basée sur les logiques descriptives. Un autre facteur clé de notre choix est l'acceptation considérable du langage OWL par les secteurs académiques et les marchés industriels. En effet, les recherches et les développements basés sur OWL sont largement soutenus par les milieux universitaires aussi bien que les industriels. Enfin, de nombreux logiciels libres « *open-source* » sont actuellement disponibles permettant de créer et de maintenir des ontologies en OWL, par exemple l'éditeur des ontologies *Protégé* (Holger, et al., 2004).

Comme illustré dans la figure 21, la syntaxe du système de CAO X doit être traduite en même langage de représentation que l'ontologie commune, à savoir OWL DL. A cet effet, seule la syntaxe est changée, mais la terminologie reste intacte. Des convertisseurs syntaxiques doivent être exploités afin de parvenir à la réalisation de cette étape. Dans ce contexte, nous avons effectué des études sur la conversion syntaxique à partir des formalismes de représentation, notamment XML, DAML+OIL, *etc*. (Abdul Ghafour, 2004), (Ferreira da Silva, et al., 2006). DAML+OIL et OWL partagent le même paradigme de représentation, ce qui entraîne une conversion facile entre les deux langages. D'autres travaux de conversion syntaxique ont été cités dans (Ferreira Da Silva, 2007), tels que la conversion de UML vers OWL (Djuric, 2004), de XML vers OWL (Bohring, et al., 2005) et de XSD vers OWL (Gil, et al., 2005). De plus, la conversion du langage EXPRESS vers OWL-DL est détaillée dans (Ferreira Da Silva, 2007).

3.3.3.2 Mise en correspondance sémantique

La phase de conversion syntaxique engendre la construction des ontologies d'application représentées avec le langage OWL DL. Cette réduction d'hétérogénéité rend les ontologies d'application plus prêtes à communiquer avec l'ontologie commune CDFO. Toutefois, les hétérogénéités structurelles et sémantiques ne peuvent pas être traitées par des processus de conversions syntaxiques simples. Il est donc nécessaire de surmonter cette problématique par la définition des axiomes et des règles de correspondance mettant en œuvre un processus de communication entre les ontologies d'application et l'ontologie commune. Cette phase de conversion se focalise particulièrement sur l'aspect sémantique durant le processus d'échange de données entre plusieurs ontologies.

D'ailleurs, des facteurs clés dans la réalisation de cette mise en correspondance résident dans l'expressivité riche de la sémantique du langage OWL DL et dans ses capacités de raisonnement basé sur la logique descriptive. D'une part, cette expressivité permet, par exemple, de définir des classes complexes en utilisant des opérateurs logiques (intersection, union et complément). De plus, les primitives de restriction du langage OWL peuvent être utilisées pour créer de nouvelles spécifications des informations d'un produit en contraignant les caractéristiques de leurs propriétés (*rôles*), telles que le domaine, le co-domaine, la cardinalité, ou la valeur.

D'autre part, la mise en correspondance des ontologies est réalisée par la définition des axiomes qui permettent, par exemple, de raisonner que deux termes sont équivalents sémantiquement, même s'ils utilisent des terminologies différentes. En effet, l'instauration des règles de correspondance entre différentes entités et l'appel aux services d'inférence aboutissent à la traduction automatique de la terminologie d'un système X vers la terminologie commune de CDFO. Le résultat de cette traduction consiste donc en des instances de l'ontologie commune CDFO, intégrant les informations sémantiques transmises du système X, et qui seront converties à leur tour vers le système Y.

Par exemple, une nouvelle classe des trous taraudés $ThreadedHole$ est définie comme une sous-classe de l'intersection du concept trou $Hole$ avec une restriction sur des valeurs de propriétés. La description de cette classe, en LD, est définie comme suit :

$$ThreadedHole \equiv Hole \sqcap (\exists\ isThread\ \{true\} \sqcap \exists\ hasThread\ HoleThread)$$

Cette description implique que toutes les instances du concept trou, $Hole$, liées à des instances de taraudage, $HoleThread$, peuvent être classifiées automatiquement comme étant des instances de la classe définie des trous taraudés, $ThreadedHole$. Les instances de cette classe définie peuvent être déduites automatiquement grâce aux services d'inférence du langage OWL DL.

D'ailleurs, le langage des règles SWRL est utilisé dans notre méthode de mise en correspondance pour définir formellement des relations plus génériques entre les concepts, et avec plus de souplesse. En particulier, en utilisant OWL et SWRL, des faits déduits pourraient être détectés pour découvrir de manière efficace des relations implicites à partir de descriptions explicites. Notons que notre méthode de mise en correspondance, présentée ci-dessus, sera détaillée dans le chapitre 5 de ce manuscrit.

3.4 Synthèse

Nous avons présenté dans ce chapitre notre approche d'échange de la sémantique des données entre différentes applications de CAO à base de features. La méthodologie de cette approche est basée sur un échange indirect, par l'intermédiaire d'un format neutre. Pour surmonter les limites des standards actuels d'échange de données, tels que STEP, nous nous sommes intéressé aux technologies du Web Sémantique permettant la représentation des connaissances dans un environnement collaboratif. Concrètement, notre méthodologie repose sur la représentation des données techniques des modèles de produit dans une ontologie commune de features de conception, CDFO '*Common Design Features Ontology*'. Cette ontologie commune, représentée avec le langage OWL DL, est assez expressive pour considérer la sémantique associée aux données des produits, telles que l'intention de conception. L'expressivité de cette ontologie est d'ailleurs enrichie par la création des règles de correspondance en utilisant le langage SWRL.

Dans ce contexte, nous avons abordé ce chapitre par une étude sur les travaux effectués dans le domaine du Web Sémantique. Nous avons montré l'utilité de ses technologies pour pallier les problèmes actuels d'échange de données d'un produit. Ainsi, nous avons présenté les différents formalismes et langages définis pour la représentation des ontologies, tout en mettant l'accent sur le langage OWL. Tenant compte de certaines limitations de ce dernier en termes de description des propriétés complexes, nous avons

introduit le langage des règles du Web Sémantique, SWRL, permettant l'extension de l'expressivité du langage OWL. Un avantage significatif du langage OWL réside dans les capacités de raisonnement fournies par les logiques de descriptions.

Nous avons présenté dans la deuxième section de ce chapitre la méthodologie de notre approche ontologique d'échange de données. Cette méthodologie est basée principalement sur deux grandes étapes : homogénéisation syntaxique et mise en correspondance. La première étape permet de traiter les hétérogénéités syntaxiques entre les différents formats de données d'un produit. Il s'agit particulièrement de transformer les représentations des modèles de produits de leur format d'origine vers un format commun, à savoir OWL DL. Cette phase facilite la deuxième étape, surtout que celle-ci se focalise sur les aspects sémantiques. Elle consiste à traduire la sémantique entre les ontologies d'application OWL résultant de la première étape et notre ontologie commune. Cette étape nécessite l'élaboration d'axiomes et de règles de correspondance permettant d'établir des liens entre les différentes entités des ontologies.

Cependant, la mise en place de notre approche nécessite d'abord la construction de notre ontologie commune. Cette ontologie servira de noyau pour notre prototype d'échange de données. Cette construction sera étudiée en détail dans le chapitre suivant.

Chapitre 4 Construction de notre ontologie commune « *CDFO* »

4.1 Introduction

Nous avons défini dans le chapitre précédent notre approche ontologique d'échange de données basée sur la création d'une ontologie commune servant d'intermédiaire entre les différentes applications de CAO. L'objectif de ce chapitre est de décrire la définition et le développement de notre ontologie commune de features de conception que nous avons appelée CDFO, « *Common Design Features Ontology* ». Dans ce contexte, nous décrivons la méthodologie de développement de notre ontologie commune (*cf.* section 4.2), ainsi que la représentation graphique (*cf.* section 4.3) et la description formelle (*cf.* section 0) de la structure des données représentées dans cette ontologie. Cette ontologie est représentée formellement avec le langage OWL DL. Cependant, sa représentation sera enrichie sémantiquement par la création d'axiomes et de règles définies avec le langage SWRL (*cf.* section 4.4.2.1).

4.2 Méthodologie de construction de l'ontologie

Le développement d'une ontologie consensuelle et partagée d'un domaine spécifique est un grand défi en soi quelque soit le domaine utilisé. Ce développement se réalise par un processus long et complexe qui se compose de plusieurs étapes. D'autre part, la difficulté de la construction d'une ontologie est proportionnellement liée au volume considérable de données qui doivent être gérées et auxquelles il faut rajouter les relations qui connectent les concepts les uns aux autres (généralisation, héritage, agrégation, instanciation, *etc.*) (Dartigues, et al., 2007). De plus, la construction d'une ontologie nécessite la participation des experts du domaine jouant un rôle primordial dans l'extraction des concepts et des relations du domaine grâce à leurs connaissances. Ensuite, ces concepts et leurs relations seront décrits formellement moyennant les technologies de représentation des connaissances afin de rendre la sémantique des données interprétable par la machine.

Différentes méthodes et techniques de construction d'ontologies ont été proposées dans la littérature (Uschold, et al., 1996), (Gruninger, et al., 1995), (Fernandez-Lopez, et al., 2003). En effet, les travaux de recherche sur les ontologies n'ont pas abouti à un consensus sur une méthodologie « *unique* » et « *correcte* » de construction d'ontologie, mais à la détermination des étapes nécessaires liées aux caractéristiques de l'ontologie elle-même. Il s'agit particulièrement de considérer sa définition, son domaine d'application, le type d'information que l'ontologie doit contenir, l'expressivité de son langage de représentation, *etc*. Cependant, la plupart de ces techniques convergent vers une méthodologie assez générique pour la construction d'ontologies (Uschold, et al., 1996). Cette méthodologie comprend principalement les étapes suivantes :

1) **Identification du domaine de l'ontologie** : Elle permet d'identifier les objectifs et les raisons de la construction de l'ontologie.
2) **Construction de l'ontologie :** Elle est composée de plusieurs sous-étapes :
    a. *Saisie de l'ontologie* : A ce niveau s'effectuent l'identification et la définition des concepts clés et leurs caractéristiques dans un domaine d'intérêt ainsi que les relations existantes entre ces concepts.
    b. *Codage de l'ontologie* : Il concerne la représentation de ces concepts et ces relations dans un langage formel, c'est-à-dire interprétable par la machine.
    c. *Intégration des ontologies existantes* : consiste à associer certains concepts et termes d'une ontologie avec des concepts et termes d'autres ontologies.
3) **Evaluation :** Cette étape permet de vérifier la cohérence et la complétude de l'ontologie par rapport aux objectifs qui lui sont définis.
4) **Documentation :** Il s'avère indispensable pour l'utilisation de l'ontologie de mettre à disposition de ses utilisateurs une documentation sur les différents éléments définis dans cette ontologie.

Ainsi, pour la construction de notre ontologie commune, nous avons adopté une méthode basique et simple, illustrée dans la figure 22, basée sur la méthodologie générique présentée ci-dessus. A cet effet, nous avons identifié, dans un premier temps, notre domaine d'application par une analyse effectuée sur plusieurs systèmes commerciaux de CAO à base de features. Cette analyse mène à l'apprentissage du domaine d'application, par l'extraction des concepts du domaine, ainsi que les relations entre ces concepts. L'objectif est de définir un vocabulaire commun de features de conception partagé par différents ingénieurs dans un

environnement collaboratif. Ensuite, nous avons élaboré une représentation graphique de notre ontologie permettant de visualiser les concepts et leurs relations dans l'ontologie. Enfin, nous avons représenté formellement notre ontologie commune avec le langage OWL DL, où la sémantique des données est définie explicitement et traitable par la machine. Ces deux représentations graphique et formelle seront détaillées dans les sections suivantes de ce chapitre.

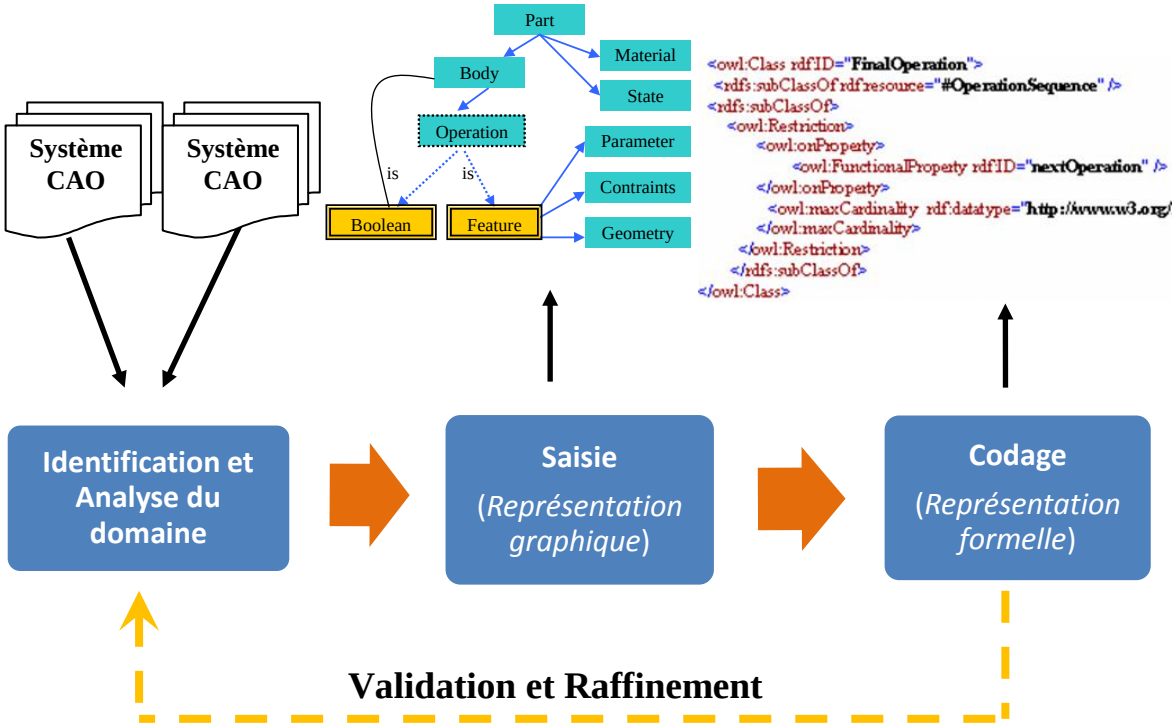


Figure 22. Méthodologie de construction de l'ontologie commune

D'ailleurs, comme le montre la figure 22, le processus de construction d'une ontologie est principalement caractérisé par l'application d'une méthode itérative de développement. Initialement, nous définissons les concepts de base de l'ontologie. Nous parcourons ensuite un processus d'évolution progressive de l'ontologie par une séquence de révisions et d'opérations de raffinement en détaillant davantage les descriptions de ces concepts. Nous avons eu recours à des experts du domaine de la CAO durant la phase d'apprentissage afin de pouvoir évaluer notre ontologie développée en l'intégrant dans plusieurs applications de CAO. Ainsi, nous déterminons les solutions les plus adaptées à notre ontologie de sorte qu'elle puisse parvenir aux objectifs d'échange en termes de réflexion d'une image réelle du domaine de la CAO, de maintenance, et d'extensibilité. Ce processus itératif de développement s'effectue éventuellement tout au long du cycle de vie de l'ontologie.

4.2.1 Analyse du Domaine d'étude

L'identification du domaine de l'ontologie est une étape primordiale pour la construction d'une ontologie. Cette première étape commence généralement par une abstraction de connaissance afin d'identifier les parties pertinentes au domaine concerné. Ceci implique la spécification des raisons pour lesquelles une ontologie est en cours de construction ainsi que son usage envisagé. Dans ce cadre, une série d'opérations de raffinement est effectuée pour extraire les concepts, leurs attributs, ainsi que les relations entre ces concepts. C'est durant cette phase qu'on détermine par exemple les informations qui vont être représentées dans l'ontologie.

Dans le cadre de notre thèse, notre ontologie commune CDFO est destinée principalement à représenter les informations sémantiques des modèles procéduraux de CAO permettant leur échange dans un environnement collaboratif. Ces informations sont définies par l'intention de conception. Plus précisément, l'intention de conception dans un modèle CAO peut être définie par l'historique de construction du modèle, les features, leurs paramètres, ainsi que les contraintes définies dans le modèle. Nous avons effectué une étude plus détaillée sur l'intention de conception et les features dans le premier chapitre (*cf.* section 1.2.3).

D'ailleurs, le développement de notre ontologie doit prendre en considération certains défis liés au domaine de la conception ainsi qu'au domaine de construction d'une ontologie. D'abord, comme nous l'avons mentionné dans le premier chapitre, les features sont largement utilisés dans la plupart des systèmes commerciaux de CAO. Cependant, il n'existe pas de consensus sur la définition des features ou sur leur classification. Chaque système CAO définit son propre vocabulaire et un ensemble de features qui lui sont spécifiques. Par conséquent, il n'existe pas de représentation unique d'un modèle de features. Pour surmonter cette problématique, nous avons été amenés à étudier la manière dont les features de conception sont représentés dans plusieurs systèmes commerciaux de CAO, notamment CatiaV5[20], SolidWorks[21], et Pro/Engineer[22]. Les classifications des features dans ces trois systèmes sont respectivement présentées dans les figure 54, figure 55, et figure 56 de l' « annexe A ». Toutefois, cette classification de features n'est pas exhaustive, et d'autres

[20] http://www.3ds.com/fr/products/catia/welcome/
[21] http://www.solidworks.fr/
[22] http://www.ptc.com/products/proengineer/

concepts supplémentaires pourraient être ajoutés au fur et à mesure de la construction de l'ontologie. Ces concepts extraits constituent la terminologie de base de notre ontologie commune.

4.2.2 Caractéristiques de l'ontologie développée

Le but de notre ontologie commune, CDFO, est de servir de format neutre, permettant l'interopérabilité sémantique entre les différents systèmes CAO à base de features. Quant au développement de cette ontologie, nous avons défini certains de ses critères essentiels dans le but de pouvoir répondre à des contraintes liées aux systèmes actuels de CAO et leur évolution. Nous présentons les critères de notre ontologie comme suit :

a) **Indépendance** : notre ontologie commune doit être indépendante de tout système de CAO afin de pouvoir représenter de la manière la plus générique possible les informations liées aux features de conception utilisés dans différentes applications. Cette contrainte est liée essentiellement aux changements perpétuels des systèmes actuels de CAO où des versions successives sont souvent générées pour chaque application. De plus, elle est liée au nombre croissant d'applications intervenant au processus de développement d'un produit. Par conséquent, l'indépendance de notre ontologie implique sa capacité à supporter l'évolution dynamique de ces nouvelles applications.

b) **Extensibilité** : L'ensemble des features utilisés dans chaque application est considérable et est susceptible de changer à mesure que les concepteurs auront des nouveaux besoins. Ainsi, l'extensibilité de l'ontologie commune est indispensable afin de pouvoir lui ajouter de nouveaux éléments à condition de ne pas violer la validité des éléments existants. Des raisonnements peuvent être effectués, avec le langage OWL DL, permettant la détection des conflits générés. Toutefois, la gestion des conflits n'est pas considérée dans notre travail.

c) **Ambigüité syntaxique** : Il peut y avoir différents types d'ambiguïtés de la sémantique entre les différentes ressources d'information. Même si deux systèmes utilisent une même terminologie, ils peuvent avoir une différence dans la signification qui leur est assignée. Un système qui convertit la sémantique avec succès devrait être capable d'identifier les différentes significations des termes syntaxiquement similaires. Une autre ambigüité peut se produire entre deux termes différents syntaxiquement, mais ayant une même signification.

d) **Formalité** : l'ontologie commune doit représenter formellement la sémantique associée à une terminologie c'est-à-dire d'une manière compréhensible par la machine et accessible durant tout le cycle de vie d'un produit. C'est une spécificité de l'ontologie elle-même surtout avec un langage puissant en termes d'expressivité tel que le langage OWL DL.

e) **Inférence** : Un des grands avantages de l'utilisation de l'ontologie réside dans les capacités de raisonnement effectuées sur les bases de connaissances. Pour une représentation efficace de la sémantique, une ontologie doit permettre de détecter automatiquement des correspondances sémantiques entre les différents concepts. Ainsi, de nouvelles connaissances pourront être déduites à partir des connaissances explicitement définies. Un autre avantage du raisonnement réside également dans la capacité de maintenir la consistance de l'ontologie durant son évolution.

4.3 Représentation graphique

Lors de la phase précédente d'étude du domaine, notre analyse des systèmes CAO à base de features a conduit à l'extraction d'un certain nombre de concepts considérés dans notre ontologie commune. Nous nous intéressons dans cette phase à décrire les éléments fondamentaux de notre ontologie de features, avec une attention particulière aux features de conception dans une pièce mécanique en 3D. L'objectif est de parvenir à échanger l'ensemble des données sémantiques des modèles CAO comme étant des instances de cette ontologie commune. Par conséquent, les utilisateurs peuvent accéder aux connaissances de la conception collaborative en utilisant, par exemple, des requêtes sémantiques.

Dans la suite, nous décrivons la représentation graphique de notre ontologie afin de mettre en relief certaines caractéristiques et relations existantes entre les différents concepts. Cependant, nous n'avons pas l'intention de fournir une représentation complète de notre ontologie, mais nous nous contentons de présenter un schéma au niveau générique.

4.3.1 Schéma générique de l'ontologie commune

La figure 23 illustre un schéma générique des principaux concepts de notre ontologie CDFO.

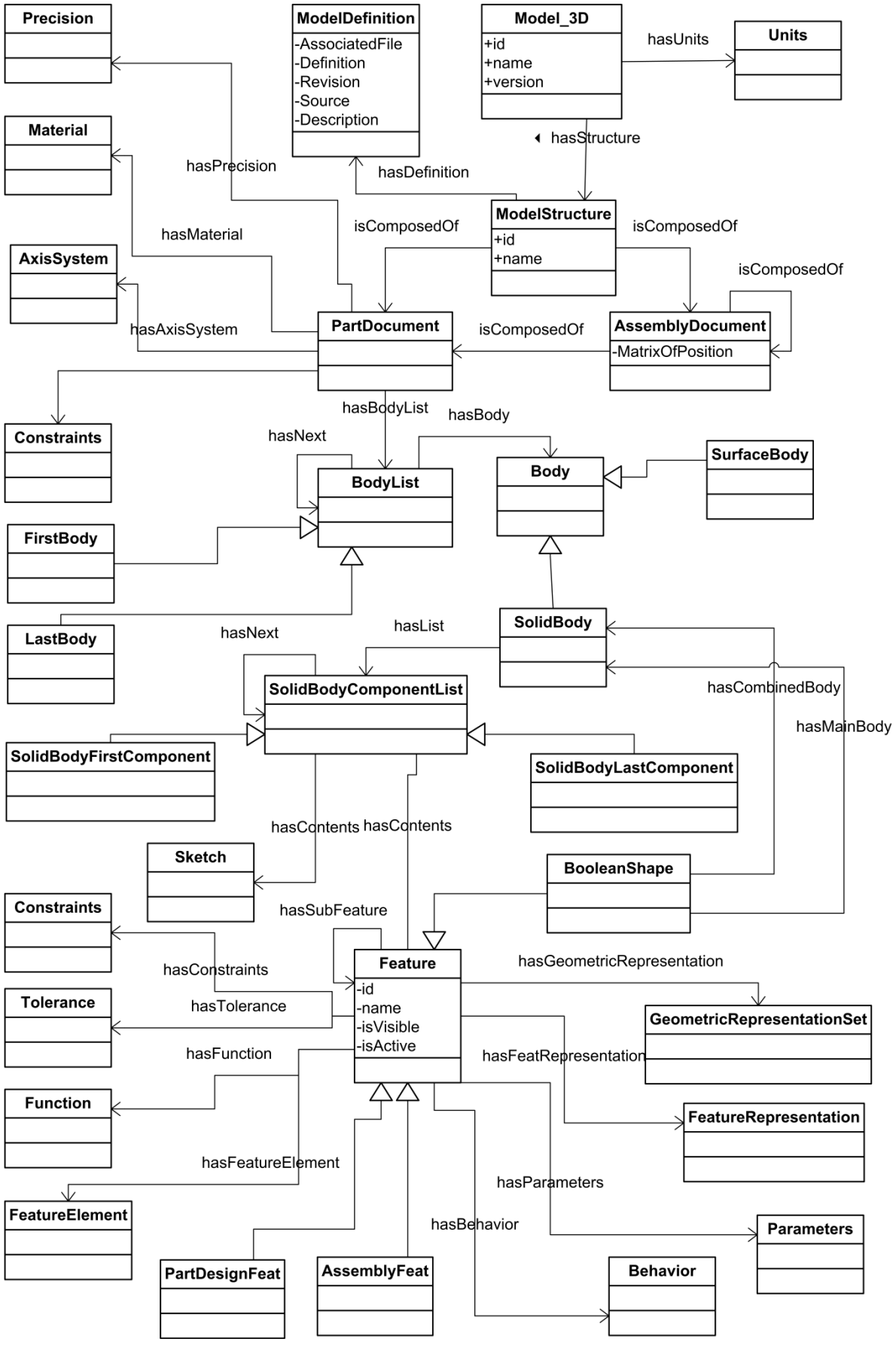


Figure 23. Schéma générique de notre ontologie commune CDFO

Ce schéma générique ne considère pas uniquement les données géométriques, mais aussi l'information technique relative à une pièce, ou un assemblage représentant le produit. Les concepts sont représentés sous forme de classes et leurs divers types de relations sont définis. Par la suite, nous décrivons les principaux concepts définis dans notre ontologie.

La classe principale de notre ontologie est "$Model\_3D$" qui désigne la représentation volumique, en 3D, du modèle CAO d'un produit. Elle constitue la racine de l'arbre de spécification définissant la structure du produit. Cette structure comprend toutes les informations du modèle, y compris l'historique de construction, les features, et les paramètres. Un modèle de produit est conçu et assemblé pour répondre à un besoin fonctionnel. Il peut définir une pièce "$PartDocument$", représentant un composant destiné à la fabrication, ou un assemblage "$AssemblyDocument$", représentant un assemblage de composants réunis sous certaines conditions pour former un produit et s'acquitter certaines fonctionnalités.

Un modèle est caractérisé par une liste d'attributs, par exemple l'identificateur, le nom, ou la version du modèle. Il peut être également lié à des unités de mesure. La classe "$Units$" est essentielle pour la définition des unités de mesure utilisées dans l'ontologie et qui varient en fonction de domaines différents, par exemple les longueurs exprimées en pouces ou mètres.

Un assemblage est composé d'une liste de sous-assemblages ou de pièces. Chaque composant d'un assemblage est caractérisé par une matrice de position "$MatrixOfPosition$" , définissant sa position dans l'assemblage qui le contient. Une pièce est caractérisée par un matériel de constitution "$Material$" et référencée par un système de coordonnées "$AxisSystem$". De plus, une pièce est composée d'une séquence ordonnée "$BodyList$" de corps "$Body$", qui peuvent être des corps volumiques "$SolidBody$" ou surfaciques "$SurfaceBody$". Un corps volumique "$SolidBody$" est à son tour défini par une séquence de composants de solide "$SolidBodyComponentList$", où chaque composant peut désigner une esquisse "$Sketch$", ou un feature "$Feature$" caractérisant la forme de ce corps.

Des opérations booléennes "$BooleanShape$" peuvent être effectuées entre plusieurs corps, par exemple l'ajout, le retrait ou l'intersection de deux corps. Nous considérons les opérations booléennes comme des sous-classes particulières des features, liées aux corps constituant les opérandes de cette opération booléenne, c'est-à-dire les corps liées à l'opération par le biais des deux relations "$hasMainBody$" et "$hasCombinedBody$".

D'ailleurs, le concept de base de notre ontologie est la classe "$Feature$". Il peut désigner différentes sous-classes de features, telles que les features de conception "$PartDesignFeat$", les features d'assemblage "$AssemblyFeat$", ainsi que d'autres types de features que nous n'avons pas introduits dans cette représentation graphique pour des raisons de lisibilité, par exemple les features de tôlerie, des features de soudure, *etc*. Nous nous intéressons particulièrement aux features de conception, qui se réfèrent aux features de forme que nous avons décrits dans le premier chapitre de ce manuscrit. Un feature de conception constitue un sous-ensemble de la forme d'un objet de conception ayant une fonction qui lui est assignée. Il intègre dans sa définition la représentation géométrique, et détient également un contenu plus riche tel que les informations sémantiques plus complexes (fonctionnelles, structurelles, comportementales, technologiques…). Différents concepts ont été ajoutés dans notre ontologie pour décrire les propriétés des features. La relation "$hasSubFeature$" permet d'établir une relation de composition entre des features complexes pouvant être générés à partir des features simples ou features primitifs. Par exemple, un feature trou complexe peut être défini par une combinaison de trous plus simples.

Le concept "$Function$" décrit la fonctionnalité d'un feature en termes de satisfaction des exigences d'ingénierie grâce à sa fonction. La forme du feature peut être considérée comme la conception proposée pour répondre au besoin spécifié par la fonction. D'ailleurs, il est nécessaire de traiter les entités géométriques qui constituent le feature. Par conséquent, le concept "$GeometricRepresentationSet$" est ajouté à notre ontologie pour décrire la représentation géométrique d'une fonction, comme par exemple la représentation en B-REP. Ces éléments géométriques sont classifiés par leurs dimensions : 0, 1 ou 2 se référant respectivement aux sommets, arêtes, et faces. Un ensemble générique des éléments géométriques de features est déjà défini dans la partie 48 de STEP (ISO 10303-48, 1992) .

D'autres aspects peuvent être liés à un feature tels que les contraintes "$Constraints$", les paramètres "$Parameters$", la tolérance "$Tolerance$" et le comportement "$Behavior$" qui désigne un ajout ou un enlèvement de la matière. Chaque feature peut avoir de nombreuses contraintes appliquées entre les éléments géométriques du même feature ou entre deux features distincts. Une description détaillée des contraintes de features est effectuée dans (Dartigues, 2003). Une tolérance peut être associée à un feature pour représenter une partie variante d'un produit. Les paramètres sont des données essentielles qui doivent être représentées et associées pour spécifier la définition d'une instance de feature. Les paramètres

sont spécifiques à chaque type de feature, par exemple un trou peut être défini par une profondeur et un rayon. D'autres paramètres non-géométriques peuvent être définis, par exemple la norme associée à un taraudage d'un trou définissant ses caractéristiques.

4.3.2 Schéma hiérarchique des features de conception

Dans le schéma générique de notre ontologie, présenté dans la figure 23, nous nous sommes intéressés à une vue globale de notre ontologie. Nous présentons dans cette partie une classification hiérarchique des features de conception les plus utilisés dans la plupart des systèmes CAO à base de features. Cette classification, illustrée dans la figure 24, contient les catégories suivantes de features :

- "$SketchBasedShape$" : Cette catégorie contient les features à base d'esquisse, qui constituent les features de base pour la création d'un solide. Certains de ces features ajoutent de la matière et d'autres en enlèvent. Par exemple, les features "$Extrude$", "$Revolve$", et "Rib" définissent trois formes de création des solides extrudés, en suivant respectivement des directions linéaires, circulaires, ou des courbes prédéfinies par l'utilisateur. Les features correspondants d'enlèvement de la matière sont respectivement les poches "$Pocket$", les gorges "$Groove$", et les rainures "$Slot$". Le feature trou "$Hole$" est souvent utilisé dans la conception des pièces pour effectuer des coupes cylindriques.
- "$DressUpShape$" : Elle correspond aux features appliqués sur des solides existants afin d'effectuer des transitions des surfaces du solide. Par exemple, le congé "$EdgeFillet$" correspond à la transition circulaire de deux faces d'une arête.
- "$SurfaceBasedShape$" : Elle correspond aux features appliqués sur des faces, tels que la division d'une face ou la couture de deux faces adjacentes.
- "$DressUpShape$" : Elle correspond principalement aux opérations effectuées sur d'autres features de la pièce, par exemple le déplacement d'un trou, ou la création des trous par une répétition circulaire.
- "$BooleanShape$" : désigne une opération booléenne entre plusieurs corps, présentée dans le schéma générique de notre ontologie (*cf.* section4.3.2)

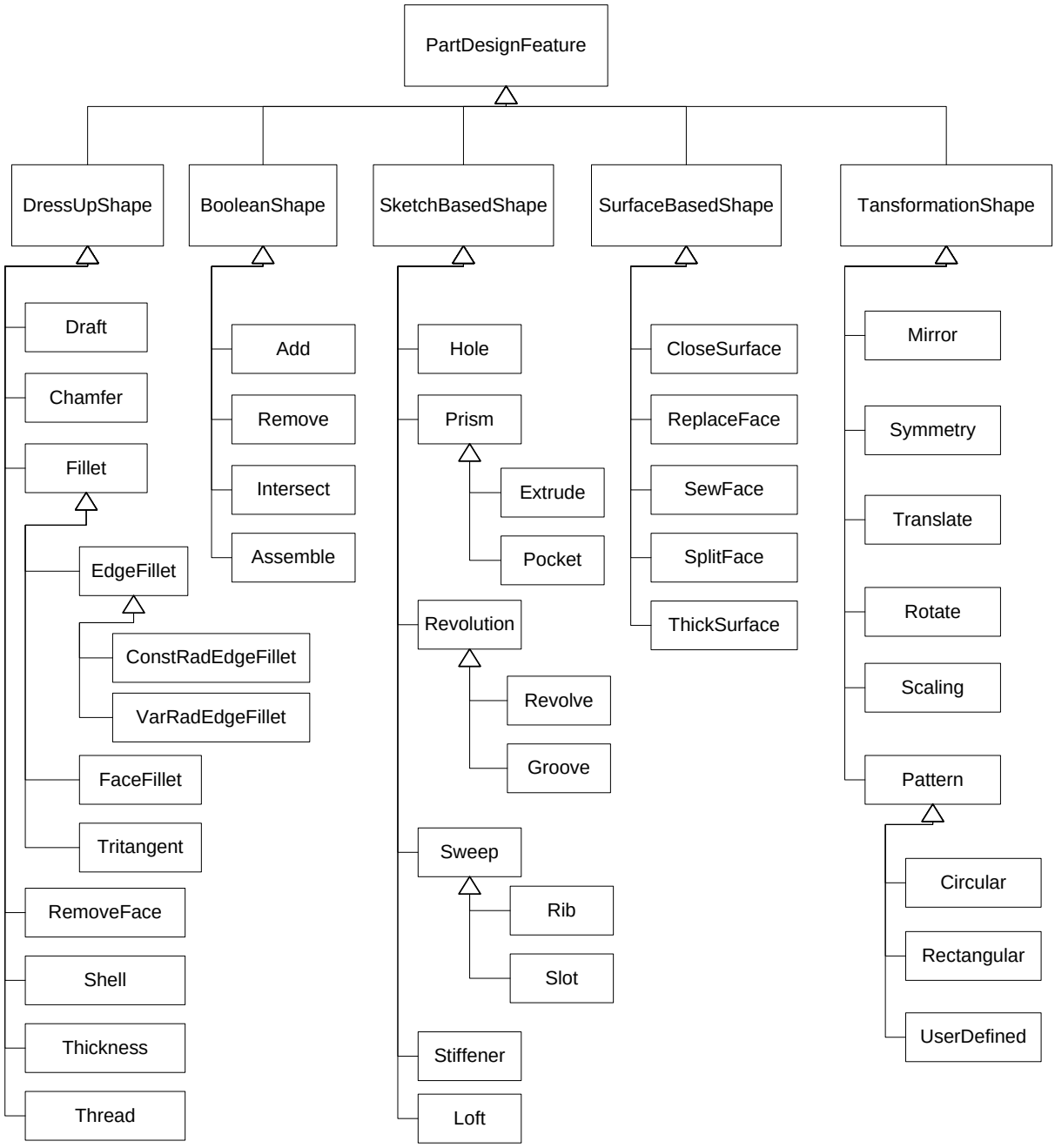


Figure 24. Hiérarchie des features de conception dans l'ontologie commune

Jusqu'ici, nous avons décrit la représentation graphique de notre ontologie qui nous a permis de visualiser ses concepts fondamentaux, ainsi que leurs relations principales. Cependant, cette représentation n'est pas interprétable par la machine. L'avantage principal d'une ontologie réside dans sa capacité de représenter formellement les connaissances d'un domaine et d'interpréter la sémantique des données. Dans la section suivante, nous procédons à l'élaboration d'une représentation formelle de notre ontologie.

4.4 Représentation Formelle

La construction de l'ontologie CDFO vise à fournir un modèle de connaissances pour l'échange des modèles procéduraux de CAO, en considérant la sémantique des données. Afin d'atteindre cet objectif, nous allons décrire dans cette section la représentation formelle de notre ontologie commune avec le langage d'ontologie OWL DL. En effet, les ontologies doivent être formellement définies pour que les informations représentées soient interprétables par la machine et, par conséquent, soient exploitées par des processus d'automatisation ou de raisonnement. Ainsi, notre ontologie est enrichie sémantiquement par la définition des axiomes constituant un élément de base et ceci pour effectuer des opérations de raisonnement. Nous n'avons pas l'intention de représenter dans la suite tous les concepts et les relations déjà définis. Nous nous contentons d'une simple partie permettant de montrer la façon dont la sémantique des données a été représentée.

4.4.1 Choix du langage

Pour élaborer cette représentation formelle, il faut tout d'abord choisir un langage adéquat. Actuellement, il existe différents types de formalismes et de langages permettant la représentation formelle des connaissances dans une ontologie. Une étude de ces langages a été réalisée dans le chapitre 3 de ce manuscrit (*cf.* section 3.2.1.1). Parmi ces différents langages, nous avons choisi le langage OWL DL, basé sur les logiques de description, LDs.

En effet, OWL DL fournit un ensemble riche de primitives de construction d'une ontologie, tout en permettant la déduction automatique. L'utilisation d'un raisonneur permet de vérifier la cohérence des descriptions de l'ontologie et de classifier les concepts définis. Le raisonneur peut donc contribuer à maintenir correctement cette hiérarchie de concepts. Par conséquent, la consistance de notre ontologie pourrait être vérifiée au fur et à mesure de sa construction. De plus, OWL DL offre une expressivité maximale tout en conservant la complétude de calcul « *computational completeness* » (l'ensemble des conclusions est garanti d'être calculable) et la décidabilité « *decidability* » (tous les calculs se termineront en temps fini) (Bechhofer, et al., 2004). D'ailleurs, notre ontologie sera enrichie sémantiquement par la définition des axiomes et des règles définies avec le langage SWRL.

4.4.2 Codage

Après avoir choisi le langage d'ontologies OWL, nous allons présenter quelques exemples de codage de notre ontologie dans un fichier ayant l'extension « *.owl* ». Nous avons utilisé l'éditeur d'ontologies *Protégé* permettant de faciliter la création et la gestion des ontologies dans un cadre graphique convivial pour l'utilisateur. Pour des raisons de visibilité, nous ne visons pas à représenter le codage complet de notre ontologie, mais nous présentons quelques exemples de descriptions de concepts fondamentaux dans notre ontologie.

La figure 25 illustre un segment de la représentation en OWL d'une pièce mécanique "$PartDocument$", en fonction de sa définition dans le schéma générique de notre ontologie (présenté dans la figure 23). Une pièce est définie comme une sous-classe d'un document "$Document$" du modèle de produit. Un document "$Document$" désigne le type du modèle, à savoir un assemblage, une pièce, ou un dessin 2D. Ainsi, nous définissons une relation de disjonction entre une pièce et les autres types de documents. Une pièce est définie par rapport à un système de coordonnées, et est liée à un solide décrivant la liste de ses composants. Une vue générique de notre ontologie implémentée est illustrée, avec le plugin Jambalaya de *Protégé*, dans l'annexe B de ce manuscrit.

```
<owl:Class rdf:ID="PartDocument">
      <rdfs:subClassOf rdf:resource="#Document"/>
      <rdfs:subClassOf>
         <owl:Restriction>
            <owl:onProperty rdf:resource="#_hasAxisSystem"/>
            <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
         </owl:Restriction>
      </rdfs:subClassOf>
      <rdfs:subClassOf>
         <owl:Restriction>
            <owl:onProperty rdf:resource="#_hasPartMainBody"/>
            <owl:someValuesFrom rdf:resource="#CDFOSolidBody"/>
         </owl:Restriction>
      </rdfs:subClassOf>
      <owl:disjointWith rdf:resource="#DrawingDocument"/>
      <owl:disjointWith rdf:resource="#ModelDoc"/>
      <owl:disjointWith rdf:resource="#ProductDocument"/>
</owl:Class>
```

Figure 25. Segment d'une représentation formelle de CDFO avec le langage OWL

4.4.2.1 Représentation de l'historique de construction

Un aspect fondamental de l'échange de l'intention de conception dans un modèle CAO est la capacité de représenter l'historique de construction, d'autant que l'ordre des features est hautement significatif dans un modèle procédural. Comme nous l'avons présenté dans le schéma générique de notre ontologie (figure 23), un corps volumique "$SolidBody$" est composé d'une liste ordonnée de composants,"$SolidBodyComponentList$", c'est-à-dire d'une séquence ordonnée d'esquisses et de features. La figure 26 présente une partie de notre ontologie illustrant cette représentation de liste.

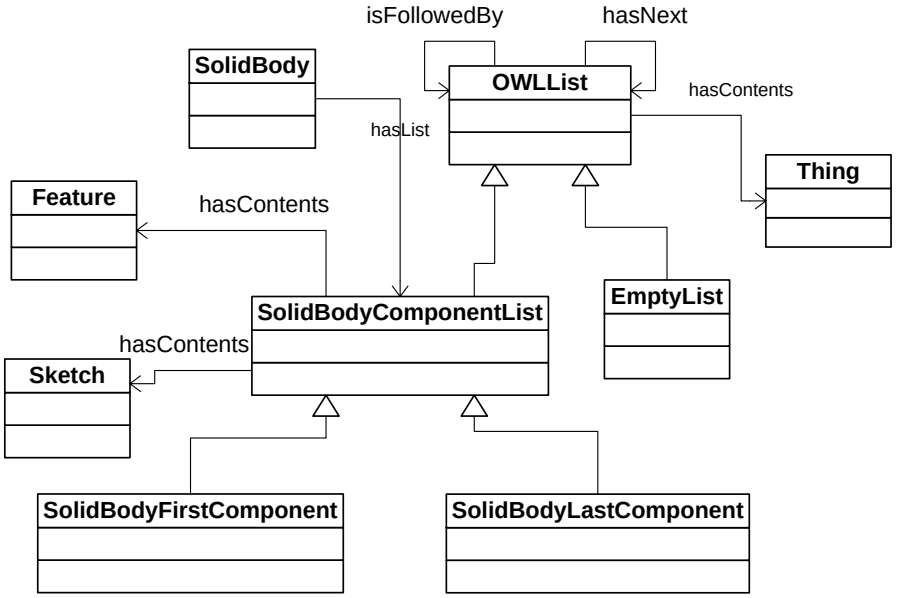


Figure 26. Schéma de représentation de l'historique de construction

Ainsi, la description du concept "$SolidBody$" dans notre ontologie OWL comprend une restriction existentielle sur la propriété "$hasList$", représentée de la façon suivante :

$$SolidBody \sqsubseteq Body \sqcap (\exists\, hasList\, (SolidBodyComponentList \sqcup EmptyList))$$

Cette description indique qu'un corps volumique est un corps qui doit avoir une liste de composants de solide ou une liste vide. D'ailleurs, la classe $SolidBodyComponentList$ désigne une liste de composants, définie comme une sous-classe d'une description de liste générique, appelée $OWLList$. Chaque instance de cette liste contient soit une esquisse $Sketch$, soit un feature.

Il n'existe pas de constructeurs définissant une description standard des séquences dans OWL. Cependant, dans (Drummond, et al., 2006) , les auteurs présentent une méthode d'implémentation des séquences dans OWL ainsi que les constructeurs nécessaires. Nous nous sommes basés sur cette méthode pour l'implémentation des séquences dans notre ontologie. Cette méthode d'implémentation définit la classe $OWLList$ ayant les caractéristiques suivantes (Figure 27):

- Une instance de "$OWLList$" est considérée comme un élément de la liste suivi seulement par une autre instance de "$OWLList$".
- $hasNext$: est une propriété fonctionnelle de $OWLList$ qui pointe sur l'élément direct qui suit dans la liste.
- $isFollowedBy$: est une propriété transitive. Elle est définie comme une super-propriété de la propriété $hasNext$ permettant d'inférer tous les successeurs indirects d'un élément de la liste.
- $hasContents$: est une propriété fonctionnelle « *Object Property* » définissant un pointeur sur le contenu d'un élément de la liste.
- La classe "$EmptyList$"est une sous-classe de "$OWLList$" ne contenant pas d'éléments. Elle est définie par la restriction suivante:

$$\begin{aligned} EmptyList \\ &\equiv OWLList \\ &\sqcap \left(\neg(\exists\ isFollowedBy\ owl{:}Thing)\right. \\ &\cup \neg\ (\exists\ hasContents\ owl{:}Thing)\left.\right) \end{aligned}$$

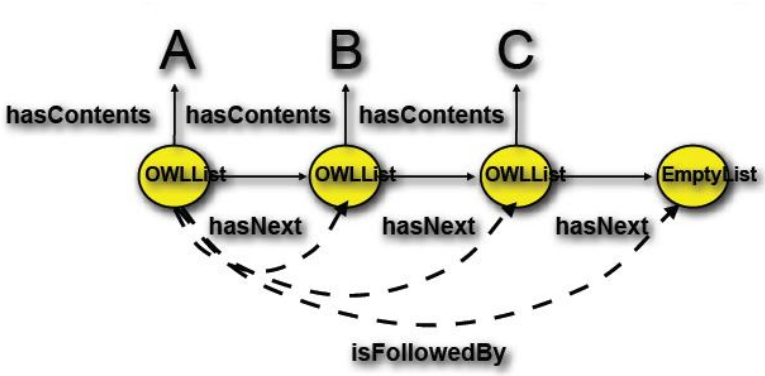


Figure 27. Schéma d'implémentation des séquences dans OWL (Drummond, et al., 2006)

Ainsi, la description de notre classe $SolidBodyComponentList$, définissant une liste de composants est représentée comme suit :

$$
\begin{aligned}
SolidBodyComponentList \\
&\equiv OWLList \\
&\sqcap \big((\exists\ hasNext\ EmptyList) \\
&\sqcup (\exists\ hasNext\ SolidBodyComponentList)\big) \\
&\sqcap (\exists\ hasContents\ (Sketch \sqcup Feature)) \\
&\sqcap (\forall\ hasContents\ (Sketch\ \sqcup Feature))
\end{aligned}
$$

De plus, la classe $SolidBodyComponentList$ a deux sous-classes définissant le premier et le dernier composant de la liste, respectivement $SolidBodyFirstComponent$ et $SolidBodyLastComponent$. Le dernier composant de la liste est un élément de la liste n'ayant pas de successeurs. Ceci est défini dans notre ontologie par la description suivante :

$$
SolidBodyLastComponent \equiv \\
SolidBodyComponentList \sqcap ((\exists\ hasNext\ EmptyList) \cup \neg(\exists\ hasNext\ owl{:}Thing))
$$

4.5 Axiomatisation et création des règles SWRL

La sémantique dans notre ontologie est encodée en utilisant des mécanismes de représentation et de raisonnement basés sur la logique descriptive. En effet, OWL DL est muni d'un ensemble riche de constructeurs permettant la définition des axiomes sur les concepts, les relations, ainsi que les instances. Par exemple, les axiomes de transitivité, de relation inverse, et de restriction de cardinalités peuvent être représentés respectivement avec $owl{:}TransitiveProperty$, $owl{:}inverseOf$, et $owl{:}Cardinality$. OWL fournit également des constructeurs de correspondance entre différentes entités. Par exemple, il est possible de définir des axiomes d'équivalence entre des concepts ou des propriétés en utilisant respectivement les constructeurs d'équivalence $owl{:}equivalentClass$ et $owl{:}equivaentProperty$. Des relations de subsomption entre des concepts ou des propriétés peuvent être également définies en utilisant respectivement $rdfs{:}subClassOf$ ou $owl{:}subPropertyOf$.

Dans le contexte de la logique descriptive, un axiome est considéré comme une hypothèse de raisonnement, fournissant de la connaissance dans un modèle de données de façon à permettre à la sémantique d'être interprétée par la machine. Cela comprend la définition des relations de subsomption ou d'équivalence entre des classes ou des propriétés,

ainsi que certaines caractéristiques des propriétés, par exemple la transitivité. Ce processus d'axiomatisation rend l'ontologie plus riche sémantiquement, et permet, en conséquence, de faire appel aux services d'inférences.

Ainsi, nous avons défini des axiomes dans notre ontologie CDFO pour décrire davantage la sémantique des concepts de base de notre ontologie. Les descriptions suivantes, décrites avec la syntaxe de la logique descriptive, représentent des exemples d'axiomes que nous avons définis dans notre ontologie.

- **Axiom1**: Dans la représentation de l'historique de construction, la propriété $hasPrevious$ a été ajoutée pour inférer les prédécesseurs d'un élément de la liste. Elle est définie comme étant la propriété inverse de la propriété $hasNext$.

$$hasPrevious \equiv (hasNext)^{-}$$

- **Axiom 2** : La propriété « *hasSubFeature* » est transitive.

$$Tr\ (hasSubFeature)$$

- **Axiom 3** : le concept $CDFO{:}\,SketchBasedShape$ est défini comme une sous-classe du concept $CDFO{:}\,Feature$ ayant au moins une propriété d'objets $CDFO{:}\,hasProfile$ avec le concept $CDFO{:}\,Sketch$.

$$CDFO{:}\,SketchBased \equiv CDFO{:}\,Feature \sqcap (\exists\ CDFO{:}\,hasProfile\ \ CDFO{:}\,Sketch)$$

Par ailleurs, le langage OWL a certaines limitations, dont la plupart concernent la définition des propriétés composites dans OWL (*cf.* section 3.2.2). Des efforts ont été effectués pour augmenter l'expressivité du langage OWL, notamment avec le développement du langage des règles du Web Sémantique SWRL. Nous avons utilisé ce langage pour la création des règles dans notre ontologie commune. Ces règles sont implémentées en utilisant le plugin de *Protégé*, à savoir SWRLTab. Ce dernier définit un environnement de développement pour gérer des règles en SWRL. Dans la suite, nous allons montrer des exemples sur la manière dont nous pouvons établir des relations complexes entre les concepts de notre ontologie en créant des règles SWRL.

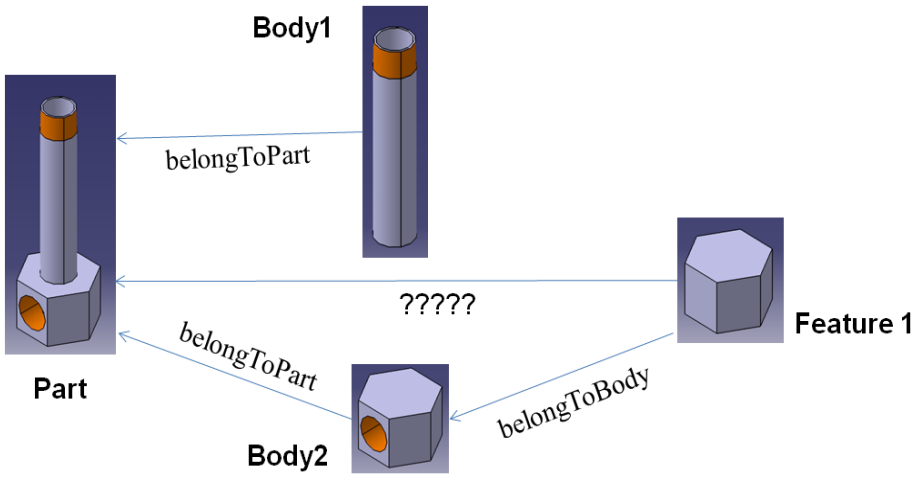


Figure 28. Exemple de définition des propriétés composites.

Considérons l'exemple illustré dans la figure 28. Dans cet exemple, une pièce "$Part$" est composée de deux corps solides "$Body1$" et "$Body2$". La propriété "$belongToPart$" est définie pour relier un corps à la pièce qui le contient. Le corps solide est à son tour composé de plusieurs features, prenons par exemple l'extrusion "$Feature1$". Une autre propriété "$BelongToBody$" est également définie permettant d'établir une relation d'appartenance du feature au corps solide. Une règle doit être créée afin de déduire une relation d'appartenance implicite entre le feature "$Feature1$" et la pièce"$Part$". D'une façon générique, cette règle indique que si un feature F appartient à un corps C, et si C appartient à son tour une pièce P, alors le feature F doit être lié à la pièce P.

Property	**Domaine**	**Co-domaine**
belongToPart	Body	Part
belongToBody	PartDesignFeature	Body

Tableau 1. Relations d'appartenance de type "Belong To"

Considérons la définition des propriétés dans le tableau 1. La propriété d'objet "$belongToPart$" est une propriété fonctionnelle définissant une relation de composition entre une pièce et un corps. La propriété d'objet "$BelongToBody$" est une propriété fonctionnelle définissant une relation d'appartenance entre les features de conception et leurs corps. Ainsi, nous avons créé les deux règles suivantes pour définir explicitement les relations entre une

pièce et les features qui sont intégrés dedans. Des instances de ces relations sont inférées grâce à l'utilisation d'un moteur d'inférence à base de règle, tel que JESS.

- $\big(SolidBody(?x) \wedge SolidBodyComponentList(?y) \wedge hasContents(?y,?z) \wedge hasSolidBodyComponentList(?x,?y) \rightarrow belongToBody(?z,?x)\big)$
- $\big(SolidBody(?x) \wedge SolidBodyComponentList(?y) \wedge hasContents(?t,?z) \wedge hasSolidBodyComponentList(?x,?y) \wedge SolidBodyComponentList\ (?t) \wedge isFollowedBy(?y,?t) \rightarrow belongToBody(?z,?x)\big)$

D'ailleurs, la création des relations composites permet éventuellement d'inférer des relations sémantiques entre les features, ce qui est nécessaire pour la navigation et l'interrogation des modèles CAO. Par exemple, pour extraire les caractéristiques d'un feature trou dans un modèle CAO, il est indispensable de détecter les relations de ce trou avec les autres features du modèle. Par exemple, si une instance de trou est suivie par un feature de translation appliqué à ce trou, de nouveaux paramètres comme les coordonnées du centre du trou devraient être calculés par rapport à la translation appliquée.

4.6 Synthèse

Dans ce chapitre, nous avons présenté le développement de notre ontologie commune des features de conception, CDFO « *Common Design Features Ontology* », qui sert d'intermédiaire entre les différentes applications de CAO. La construction de cette ontologie commune constitue une étape primordiale pour la mise en œuvre de notre approche décrite dans le chapitre précédent. Cette ontologie commune est construite d'une façon générique tout en ayant une vue globale sur les features de conception existants dans plusieurs systèmes commerciaux de CAO.

En effet, la construction d'une ontologie dans un domaine particulier est un processus long et itératif exigeant des phases de validation et de raffinement. De nouveaux concepts peuvent être ajoutés, modifiés ou supprimés. Ce processus itératif s'achève lorsque l'ontologie est jugée satisfaisante. Actuellement, il n'y a pas de méthodologie unique et correcte pour la construction d'ontologie, mais différentes approches ont été proposées. Nous avons suivi une méthodologie assez générique pour le développement de notre ontologie commune.

Dans un premier temps, nous avons identifié le domaine concerné, à savoir les systèmes CAO à base de features intégrant l'intention de conception. Nous avons étudié les features dans plusieurs systèmes CAO, avec une attention particulière aux features de conception des pièces mécaniques. Cette analyse a conduit à l'extraction des concepts fondamentaux à considérer dans notre ontologie. Nous avons ainsi représenté graphiquement ces concepts afin de mettre en relief la structure de notre ontologie incluant les caractéristiques des concepts et leurs relations principales. Cette représentation graphique nous a servi de base pour décrire formellement notre ontologie avec le langage OWL DL. Nous avons élaboré quelques exemples de représentation qui nous semblent utiles pour décrire les informations des modèles CAO. Nous avons également montré des exemples d'axiomes et de règles définis pour enrichir la sémantique représentée dans notre ontologie.

Enfin, la construction de cette ontologie ne constitue pas la dernière étape de notre travail : il faut maintenant élaborer des méthodes d'intégration entre l'ontologie développée et les ontologies d'application représentées avec OWL par la création des règles de correspondance.

Chapitre 5 Intégration sémantique d'ontologies

5.1 Introduction

La première étape de notre méthodologie, présentée dans le chapitre 3, repose sur une homogénéisation syntaxique des différents formats de modèles de représentation. Elle s'effectue par conversion vers un langage pivot, OWL DL. Ensuite, nous avons développé dans le chapitre 4, notre ontologie commune CDFO, représentée en OWL DL, servant d'intermédiaire d'échange entre les différentes ontologies d'application CAO. L'étape suivante de notre approche sera présentée dans ce chapitre. Elle consiste à réaliser l'intégration sémantique des ontologies résultantes, par l'élaboration des règles permettant d'établir des liens de correspondance (*mappings*) entre les entités des différentes ontologies. Notre objectif est de concevoir les correspondances entre les données contenues dans les différentes ontologies d'application et notre ontologie commune.

Nous présentons brièvement, dans la section 5.2, quelques définitions et méthodes actuelles d'intégration d'ontologies. Nous décrivons, dans la section 5.3, notre méthode d'intégration définie par la mise en correspondance des ontologies OWL. Elle consiste à créer des axiomes, en tirant parti de l'expressivité du langage OWL, et à définir des règles par l'utilisation du langage des règles du Web Sémantique, SWRL. Notre méthode repose principalement sur les capacités de raisonnement fournies par les moteurs d'inférence basés sur les logiques descriptives, LDs. Elle sera ensuite enrichie par le développement d'une méthode de mesure de similarité sémantique, présentée dans le chapitre suivant.

5.2 Méthodes connexes d'intégration

La distribution et l'hétérogénéité de l'information ont créé le besoin d'intégration pour améliorer l'accès et l'échange de données et assurer une recherche pertinente. Dans un domaine particulier, plusieurs ontologies peuvent être créées pour représenter des modèles de connaissances. Des entités de plusieurs ontologies, (par exemple concepts, propriétés, ou

instances) peuvent être mises en correspondance afin d'établir des liens entre ces modèles. Dans cette section nous présentons brièvement certaines méthodes actuelles d'intégration de différentes ontologies.

Une étude bibliographique sur les diverses approches de l'interopérabilité sémantique entre les ressources a été effectuée dans (Ferreira Da Silva, 2007). L'auteur a ainsi extrait de la littérature plusieurs termes d'intégration, provenant de différentes communautés. Ces termes relatifs à la mise en correspondance des entités, tels que l'alignement, la fusion, l'intégration, *etc*., sont définis comme suit (Ferreira Da Silva, 2007), (Pinto, et al., 1999) :

- **Alignement d'ontologies** : l'alignement est défini comme un ensemble de correspondances entre deux ou plusieurs ontologies (Bouquet, et al., 2005). Plus précisément, il établit une collection de relations binaires entre les vocabulaires des deux ontologies (Kalfoglou, et al., 2003). Il permet, en conséquence, de ramener deux ontologies à un accord mutuel pour les rendre compatibles et cohérentes (Abels, et al., 2005).
- **Fusion d'ontologies** : en général, la fusion désigne la mise en commun des sources de départ, qui probablement se chevauchent (Bouquet, et al., 2005), pour générer une ontologie cible unifiée (Pinto, et al., 2001). Ainsi, à la suite de la fusion il est difficile d'identifier les régions de l'ontologie cible qui ont été réutilisées à partir des ontologies sources (Pinto, et al., 2001). Pour (Bouquet, et al., 2005) la fusion d'ontologies est très liée à l'intégration.
- **Intégration d'ontologies :** selon (Kalfoglou, et al., 2003), l'intégration d'ontologies est la composition d'ontologies pour en construire d'autres, dont les vocabulaires respectifs ne sont pas forcément interprétés dans le même domaine de discours. (Pinto, et al., 2001) précisent que l'intégration est le processus de construction d'une ontologie en réutilisant (par combinaison, agrégation, spécialisation, etc.) d'autres ontologies disponibles. Ces différentes ontologies font partie de la nouvelle ontologie construite. En s'appuyant sur cette définition (Abels, et al., 2005) considèrent que l'alignement et la fusion d'ontologies sont des sous-types d'intégration d'ontologies.

D'ailleurs, la mise en correspondance s'effectue à différents niveaux : entre les entités d'une même ontologie, ou avec des entités d'autres types de ressources sémantiques. Dans ce contexte, deux catégories de mise en correspondance sont définies par (Wache, et al., 2001)

pour l'intégration de l'information : la correspondance entre les ontologies et l'information qu'elles décrivent, et la correspondance entre les différentes ontologies utilisées dans un système. Les différentes approches proposées pour ces deux catégories sont présentées comme suit :

1. **Connexion aux sources d'informations** : différentes approches générales sont utilisées pour établir une connexion entre les ontologies et les sources d'informations:
    a. *Ressemblance de la structure* : une approche directe est de produire une simple copie de structure de la source d'information et de l'encoder dans un langage qui rend possible le raisonnement automatique.
    b. *Définition des termes* : afin de clarifier la sémantique des termes dans un schéma de base de données, certaines approches utilisent les ontologies pour définir des termes supplémentaires de la base de données ou de son schéma.
    c. *Enrichissement de la structure* : Cette approche combine les deux approches précédentes. Un modèle logique est construit ressemblant à la structure de l'information source et contenant des définitions additionnelles des concepts.
    d. *Méta-annotation* : une nouvelle approche est l'utilisation des méta-annotations pour ajouter de la sémantique à la source d'information. Cette approche est devenue proéminente dans le besoin d'intégrer des informations présentes sur le web « *World Wide Web »*, où l'annotation est définie comme un moyen naturel pour l'ajout de la sémantique.
2. **Mapping inter-ontologies**: le problème de mise en correspondance (*mapping*) entre différentes ontologies est bien connu dans l'ingénierie de la connaissance. Des approches générales sont utilisées dans les systèmes d'intégration de l'information :
    a. *Les mappings définis* : une approche commune est de fournir la possibilité de définir des *mappings*. Elle est utilisée dans les systèmes où les translations entre différentes ontologies sont réalisées par des agents médiateurs spéciaux qui peuvent être adaptés pour établir la traduction entre différentes ontologies et même entre différents langages. Cette approche fournit une grande flexibilité, mais elle ne peut pas assurer la préservation de la sémantique : l'utilisateur est libre de définir des *mappings* arbitraires même s'ils n'ont pas de significations ou s'ils produisent des conflits.

b. *Relations lexiques* : Ces approches étendent le modèle de la logique de description par des relations inter-ontologies empruntées à la linguistique, par exemple « synonyme », « disjoint », « hyponyme », « hyponyme » *etc*.
c. *Connaissance de haut niveau* : afin d'éviter la perte de sémantique, un langage de la représentation formelle doit être exploité pour définir des « *mappings* » entre différentes ontologies. Une méthode directe consiste à relier toutes les ontologies utilisées à une ontologie située à un niveau plus élevé. Cette approche permet d'établir des connections entre des concepts de différentes ontologies en termes de superclasses communes. Toutefois, elle ne permet pas la correspondance directe.
d. *Correspondance sémantique* : pour surmonter l'ambiguïté causée par le « *mapping* » indirect, comme défini dans l'approche précédente, une solution est d'identifier des correspondances sémantiques entre les concepts des différentes ontologies. Ces approches comptent sur un vocabulaire commun pour la définition de ces concepts.

Le choix d'une approche d'intégration s'appuie sur différents critères. Certains sont définis dans (Wache, et al., 2001), tels que le rôle et l'architecture de l'ontologie, et les langages de représentations utilisées. Dans notre travail, nous procédons à l'intégration des ontologies par une méthode basée sur l'alignement d'ontologie, consistant à établir un ensemble de relations de correspondance entre différentes entités. Cette méthode nous semble la plus appropriée pour ne pas modifier les ontologies sources intervenant dans le processus d'intégration.

Dans la suite, le terme intégration désigne le processus d'alignement d'ontologie effectué par la mise en correspondance des entités de différentes ontologies. Nous allons ensuite proposer une méthode d'intégration, appropriée au langage OWL DL, basée sur une axiomatisation des ontologies d'application CAO, elles-mêmes (intra-ontologie), et également sur une axiomatisation entre ces différentes ontologies et notre ontologie commune CDFO (inter-ontologies).

5.3 Méthodologie d'intégration

Dans notre travail, nous définissons une méthode d'intégration basée sur l'alignement binaire des ontologies afin de préserver l'indépendance des ontologies intervenant dans le processus d'intégration. Ces ontologies sont supposées être homogènes au niveau syntaxique, après leur conversion en OWL, comme décrit dans la première phase de notre approche (*cf.* section 3.3.3). Par conséquent, l'étude menée dans ce chapitre se limite au traitement de l'intégration de l'information au niveau sémantique.

L'alignement sémantique d'ontologie consiste à établir des liens entre différentes entités, par le traitement de la sémantique formelle représentée explicitement dans une ontologie. Ceci permet donc de lever les ambiguïtés liées à l'interprétation du contenu de l'ontologie, par le bais des moteurs d'inférence. Ces derniers fournissent des capacités de raisonnement aidant à détecter automatiquement des correspondances sémantiques supplémentaires entre les entités. Ils permettent, en conséquence, de traduire les instances d'une ontologie, telles que le modèle CAO d'un produit, dans une autre ontologie. L'échange du modèle de produit est réalisé principalement par le transfert des valeurs des propriétés des instances afin de créer des instances équivalentes (comme étant des instances des concepts équivalents) dans l'autre ontologie.

L'objectif principal de la méthode d'alignement utilisée dans notre travail est de permettre l'interopérabilité sémantique des ontologies OWL DL, par l'identification des correspondances sémantiques entre les entités provenant des différentes ontologies (classes, propriétés ou instances). Cette méthode consiste, dans un premier temps, à définir explicitement des liens entre des entités des ontologies OWL DL, et ensuite à reconnaître automatiquement d'autres correspondances sémantiques entre les deux ontologies, à l'aide des services de raisonnement basés sur les logiques descriptives LDs.

5.3.1 Description de la méthodologie

Comme illustré dans la figure 29, la sémantique de chaque système CAO est décrite par sa propre ontologie OWL DL. Pour rendre ces ontologies interopérables, nous avons adopté la méthode d'échange indirect entre les ontologies d'application CAO et notre ontologie commune décrite dans le chapitre précédent. Cette ontologie commune est

construite sur un vocabulaire partagé global. Ce vocabulaire partagé comprend les termes d'un domaine particulier, en l'occurrence les systèmes CAO à base de features. Des concepts complexes du vocabulaire partagé peuvent être définis par des opérateurs logiques basés sur les LDs.

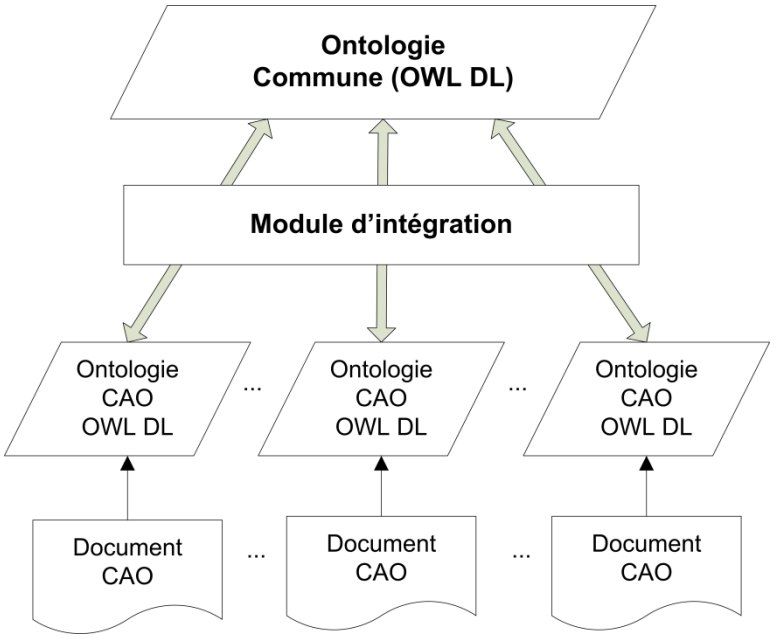


Figure 29. Schéma d'intégration d'ontologies OWL DL

En effet, l'échange d'un modèle de produit entre deux applications nécessite l'intégration mutuelle des données entre les ontologies d'application d'une part et notre ontologie commune d'autre part. Cette mise en correspondance des entités de deux ontologies est d'autant plus naturelle que l'ontologie commune que nous avons créée est générique. Cette ontologie a été conçue tout en gardant à l'esprit qu'elle sera utilisée pour l'élaboration des règles de correspondance, et qu'elle devra ainsi représenter des informations correspondantes à celles représentées dans les systèmes CAO à base de features.

Ainsi, de nouvelles ontologies représentant d'autres applications CAO peuvent être ajoutées dans un environnement collaboratif sans nécessiter la modification du vocabulaire partagé représenté dans l'ontologie commune. L'extensibilité de l'ontologie permet l'ajout d'autres concepts, sans que ces derniers soient en conflits avec les concepts déjà existants. Les conflits sont détectables dans une ontologie grâce à la vérification de la consistance de l'ontologie au fur et à mesure de sa construction. Cependant, la gestion des conflits ne sera pas considérée dans notre travail.

La mise en correspondance s'établit mutuellement entre une ontologie d'application et l'ontologie commune, représentées formellement avec OWL DL. Considérons l'échange d'un modèle CAO d'un système A vers un autre système B. Cet échange nécessite l'implémentation de deux phases d'intégration : d'abord du système A vers l'ontologie CDFO, et ensuite de CDFO vers le système B. Ce qui implique la mise en place d'un processus mutuel d'intégration d'ontologies respectivement entre l'ontologie du système A et l'ontologie commune, ensuite entre l'ontologie commune et l'ontologie du système B. Par conséquent, notre méthode d'intégration peut être réduite à la définition d'un processus d'intégration de deux ontologies d'une façon générique. Nous décrivons dans la suite le scénario défini pour le développement du processus d'intégration.

5.3.2 Scénario de la méthodologie

Rappelons que d'une façon générale, et dans les logiques de description en particulier, la représentation des connaissances s'articule autour de deux niveaux : $TBox$ et $ABox$. Le premier désigne le niveau terminologique, où sont introduites les définitions des concepts et des propriétés. Le second désigne le niveau factuel ou le niveau des assertions, où sont introduits les individus et les faits dans lesquels ces individus interviennent. À ce niveau sont représentées les instances qui désignent, dans notre travail, les modèles de produits en CAO[23].

Supposons l'échange d'un modèle de produit à base de features, appelé $ProduitX$, d'une ontologie source X vers une ontologie cible Y, sachant que X et Y peuvent désigner des ontologies d'application ou l'ontologie commune. Les ontologies d'application de X et Y sont appelées respectivement $OntoX$ et $OntoY$. $ProduitX$ représente une instance de l'ontologie $OntoX$, tandis que l'ontologie $OntoY$ est une ontologie non instanciée. Ainsi, l'ontologie source $OntoX$ est une base de connaissance composée des deux niveaux, terminologique et factuel puisqu'elle contient les instances représentant le modèle du produit $ProduitX$. D'autre part, l'ontologie cible $OntoY$ ne contient que le niveau terminologique $TBox$. Le scénario d'échange de cet exemple est illustré dans la figure 30.

[23] Notons qu'il ne faut pas confondre un modèle d'ontologie, qui est classé au niveau $TBox$, et un modèle d'un produit CAO défini comme une instance d'ontologie, représenté au niveau $ABox$

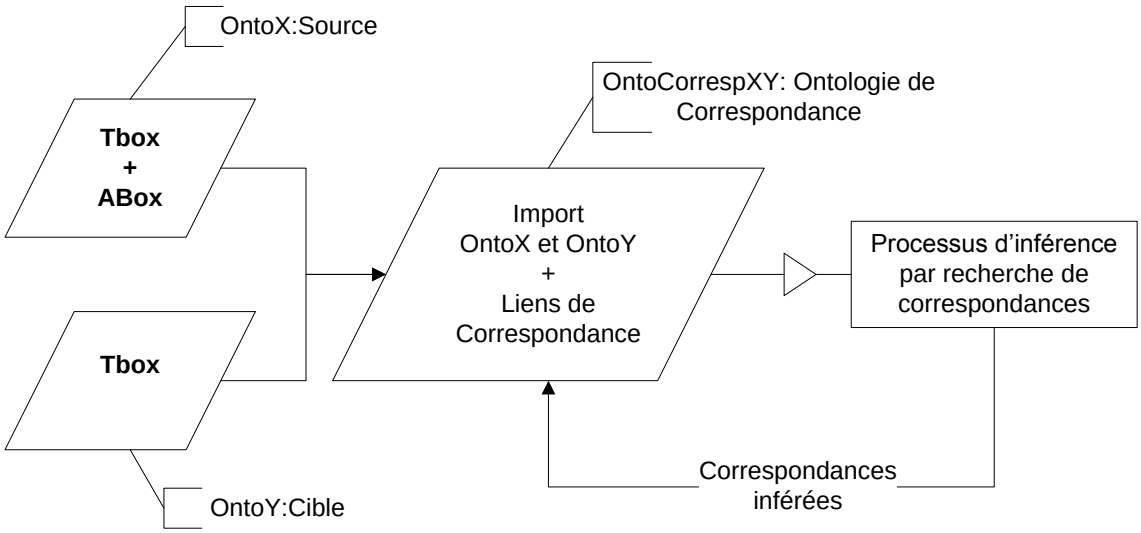


Figure 30. Scénario d'intégration

Notre méthode d'intégration consiste, dans un premier temps, à établir des liens sous forme d'axiomes et de règles de correspondance entre les entités des deux ontologies, et ensuite à les enregistrer dans une troisième ontologie appelée « *ontologie de correspondance* » (*mapping ontology*), appelée $OntoCorrespXY$ dans la figure 30, qui importe les deux ontologies concernées. Cette ontologie sert à représenter formellement les liens de correspondance entre les entités des deux ontologies, et sert également d'entrée pour le processus d'inférence.

D'autre part, notre méthode d'intégration repose sur les services d'inférence visant à détecter automatiquement des liens supplémentaires entre les entités des ontologies. Ce processus d'inférence prend en entrée l'ontologie de correspondance en OWL DL. Ensuite, il engendre un ensemble de correspondances inférées, qui doivent être validées par des experts du domaine avant d'être ajoutées dans l'ontologie de correspondance. Leur expertise est indispensable durant la phase d'apprentissage, permettant de vérifier que les liens découverts sont valides par rapport aux connaissances représentées dans les ontologies.

Ainsi, la mise en œuvre de notre méthode d'intégration nécessite de procéder à deux étapes : l'axiomatisation des ontologies intervenant dans le système d'échange des modèles CAO et le développement d'un algorithme de reconnaissance de correspondances sémantiques. Ces deux étapes sont décrites respectivement dans les deux sections suivantes.

5.4 Axiomatisation des ontologies

L'axiomatisation des ontologies est une étape préliminaire dans notre méthode d'intégration des ontologies. Elle s'effectue avant de procéder à la reconnaissance des correspondances sémantiques. Il s'agit principalement d'enrichir la sémantique associée aux données des ontologies par la définition des axiomes et la création des règles entre les différentes entités. L'axiomatisation peut être appliquée à une ontologie d'application CAO, que nous désignons par *intra-ontologie*, ou entre différentes ontologies, que nous désignons par *inter-ontologies*, aussi bien pour les concepts que les propriétés. L'axiomatisation intra-ontologie consiste en particulier à enrichir la sémantique formelle des entités de l'ontologie concernée, tandis que l'axiomatisation inter-ontologies s'applique aux ontologies de correspondance afin d'établir des liens de correspondance entre deux ontologies. Le processus d'axiomatisation tire parti de la richesse de l'expressivité du langage OWL DL pour la définition des axiomes à l'aide de ses constructeurs et du langage SWRL pour la création des règles. Ces deux types d'axiomatisation seront détaillés dans les deux sous-sections suivantes.

D'autre part, l'axiomatisation des ontologies s'effectue durant la phase d'apprentissage et nécessite l'intervention des experts du domaine capables de spécifier et de valider les liens entre les différentes entités. Par le terme « *entité* », nous désignons un concept, une propriété ou une instance dans une ontologie. En effet, une ontologie OWL DL, représentant une base de connaissance (*KB*), est formée d'un ensemble de concepts, de propriétés, et d'instances, où $KB = \{C, R, I\}$

- C désigne l'ensemble des concepts. La hiérarchie de ces concepts est définie par les relations de subsomption entre eux. Un concept C est subsumé par un concept D, noté par $C \sqsubseteq D$, signifie que C est plus spécifique que D, où D est plus générique que C. Un concept peut être primitif ou défini.
- R : désigne l'ensemble de relations permettant d'établir des relations sémantiques entre les concepts.
- I : L'ensemble des instances. Il représente les individus du monde réel.

5.4.1 Axiomatisation intra-ontologie

Cette phase correspond principalement aux correspondances définies entre les entités d'une même ontologie d'application CAO. L'objectif de cette phase est d'enrichir

sémantiquement l'ontologie par la description des entités en termes d'axiomes définis par des experts du domaine de l'application en question. Cet enrichissement affecte l'automatisation de l'intégration où des correspondances sémantiques peuvent être davantage inférées. Cette phase s'effectue essentiellement au niveau terminologique des ontologies, le $TBox$.

En effet, les ontologies d'application CAO, intervenant dans le processus d'intégration, résultent de la phase d'homogénéisation syntaxique, autrement dit, la conversion d'autres formats vers un langage pivot, en l'occurrence OWL DL (*cf.* section 3.3.3.1). Les formalismes de représentation des connaissances divergent aussi bien au niveau syntaxique qu'au niveau de leur expressivité. Ces différences d'expressivité signifient que certaines informations peuvent être exprimées formellement dans certains langages, mais pas dans d'autres, par exemple l'expression de la négation sur un concept.

Par conséquent, la migration des données d'un langage moins expressif, tel que XML, vers un langage plus expressif, tel que OWL, engendre des ontologies ayant une sémantique pauvre, où la richesse de l'expressivité du langage cible n'a pas été complètement exploitée. Ainsi, les descriptions des connaissances des ontologies d'application CAO, sont souvent limitées à des concepts primitifs, ou à des propriétés restreintes aux relations de subsomption. C'est le cas où le format d'origine est, par exemple, une taxonomie représentée avec le langage XML. Toutefois, l'expressivité du langage OWL DL est relativement riche, permettant la définition des concepts plus complexes à l'aide des opérateurs logiques des LDs, ainsi que des relations sémantiques ayant des caractéristiques spécifiques telles que la transitivité. A cet effet, nous envisageons dans cette étape d'enrichir la description des concepts des ontologies par des descriptions sémantiques plus complexes.

Dans la suite, nous considérons l'expressivité du langage OWL DL, afin de mettre en relief les primitives du langage qui nous intéressent dans la méthode d'intégration. Notre choix du langage OWL DL est justifié par le fait que ce langage permet de représenter une ontologie avec une expressivité maximale tout en gardant les capacités de raisonnement fournies par les moteurs d'inférence basés sur les logiques de description. L'expressivité du langage OWL DL correspond à la variante *SHOIN(D)* de la logique descriptive (Horrocks, et al., 2003). La syntaxe et la sémantique de cette variante sont présentées dans le tableau 2. Dans ce tableau, A et B représentent des concepts primitifs, C et D représentent des concepts complexes. Alors que R désigne un rôle atomique.

Lettre		Syntaxe	Sémantique	Description
$\mathcal{S}$	$\mathcal{AL}$	$\top$	Δ^I	Top *(owl :Thing)*
		$\bot$	$\emptyset$	Bottom *(owl :Noting)*
		A	$A^I \subseteq \Delta^I$	Concept primitif
		R	$R^I \subseteq \Delta^I \times \Delta^I$	Rôle atomique
		$\neg A$	$\Delta^I \backslash A^I$	Négation atomique
		$C \sqcap D$	$C^I \cap D^I$	Conjonction (intersection) *(owl :intersectionOf)*
		$(\forall R.C)$	$\{x \in \Delta^I \mid \forall y.(x,y) \in R^I \rightarrow y \in C^I\}$	Quantificateur universel complet (*owl :allValuesFrom*)
		$(\exists R)$	$\{x \in \Delta^I \mid \exists y.(x,y) \in R^I \wedge y \in \Delta^I\}$	Quantificateur existentiel limité (*owl :someValuesFrom*)
	$\mathcal{C}$	$\neg C$	$\Delta^I \backslash C^I$	Négation complète
		$Tr(R)$	$(R^I) = (R^I)^+$	Relation transitive *(owl :transitiveProperty)*
$\mathcal{H}$		$R_1 \sqsubseteq R_2$	$R_1^I \subseteq R_2^I$	Inclusion des rôles (*owl :subPropertyOf*)
$\mathcal{O}$		$\{o_1, \ldots\}$	$\{o_1^I, \ldots\}$	Enumération des classes *(owl :oneOf)*
$\mathcal{I}$		R^-	$(R^I)^-$	Propriété inverses *(owl :InverseOf)*
$\mathcal{N}$		$(\leq nR)$	$\{x \in \Delta^I \mid\mid \{y:(x,y) \in R^I\} \leq n\}$	Restriction de cardinalité maximale (*owl :maxCardinality*) où $n \in \mathbb{N}$
		$(\geq nR)$	$\{x \in \Delta^I \mid\mid \{y:(x,y) \in R^I\} \geq n\}$	Restriction de Cardinalité minimale (*owl :minCardinality*) où $n \in \mathbb{N}$

Tableau 2. Description de l'expressivité de la variante SHOIN

L'expressivité de la variante *SHOIN(D)* est décrite de la façon suivante :

- $\mathcal{S}$ désigne une abréviation du $\mathcal{AL}$ et $\mathcal{C}$, ainsi que la transitivité des propriétés. Sachant que $\mathcal{AL}$ (*Attributive Language*) représente le langage de base et $\mathcal{C}$ désigne une négation des concepts complexes. Le langage $\mathcal{AL}$ permet :
    - La négation des concepts primitifs ou atomiques $\neg A$
    - L'intersection des concepts $(C \sqcap \mathrm{D})$
    - Les restrictions universelles $(\forall R.C)$
    - Les quantificateurs existentiels limités $(\exists r)$, ceci est considéré comme un cas particulier de $(\exists r.C)$, où $C \equiv \top$.
- $\mathcal{H}$: permet la subsomption des rôles $R_1 \sqsubseteq R_2$
- $\mathcal{O}$: permet la description de concepts par l'énumération d'individus nommés (par le constructeur *owl:oneOf*)
- $\mathcal{I}$: permet les propriétés inverses.
- $\mathcal{N}$: permet des restrictions de cardinalités simples $(\geq n\ R)$ et $(\leq n\ R)$
- $\mathcal{D}$: permet l'utilisation des propriétés de type de données (*DataTypeProperty*), les valeurs de données.

Notons que le langage OWL DL permet également la définition de la disjonction entre deux concepts complexes $(C \sqcup D)$ et la quantification existentielle typée ou complète, $(\exists\, R.C)$. Ces deux descriptions sont disponibles dans OWL puisque ce dernier permet la négation des concepts définis, (représentée avec la lettre $\mathcal{C}$ de la variante). Elles sont représentées respectivement en fonction de la négation comme suit :

- $(C \sqcup D) \equiv \neg(\neg C \sqcap \neg D)$
- $(\exists\, R.C) \equiv \neg\, (\forall\, R.\neg C)$

L'axiomatisation concerne aussi bien les concepts que leurs propriétés. Des axiomes sur des propriétés peuvent être créés pour spécifier certaines caractéristiques des propriétés, par exemple la transitivité, la symétrie ou la fonctionnalité. D'autres axiomes peuvent être également créés pour définir le domaine et le co-domaine de la propriété. D'ailleurs, des constructeurs de correspondance sont définis dans ce langage permettant, par exemple, la création des axiomes d'équivalence ou de subsomption entre différentes entités. Ces constructeurs nous serviront de base pour l'axiomatisation, et en conséquence, pour la mise en correspondance des entités de différentes ontologies. Ces constructeurs sont présentés dans le tableau 3.

Type	Syntaxe	Sémantique	Description
Classe	$C \equiv D$	$C^I = D^I$	Equivalence des classes (*owl :equivalentClass)*
	$C \sqsubseteq D$	$C^I \subseteq D^I$	Subsomption des classes (*rdfs :subClassOf)*
	$C \sqcap D \equiv \perp$	$C^I \cap D^I = \emptyset$	Disjonction des classes *(owl:disjointWith)*
Propriété	$R_1 \equiv R_2$	$R_1^I = R_2^I$	Equivalence des propriétés (*owl :equivalentProperty)*
	$R_1 \sqsubseteq R_2$	$R_1^I \subseteq R_2^I$	Subsomption des propriétés (*rdfs :subPropertyOf)*
	$\geq 1\, R \sqsubseteq C$	$R^I \subseteq C^I \times \Delta^I$	Domaine des propriétés (*rdfs :domain)*
	$\top \sqsubseteq \forall R.C$	$R^I \subseteq \Delta^I \times C^I$	Co-domaine des propriétés *(rdfs :range)*
	$R \equiv ({}^{-}R_o)$	$R^I = (R_o^I)^{-}$	Propriété inverses *(owl :inverseOf)*
	$R \equiv ({}^{-}R)$	$R^I = (R^I)^{-}$	Propriété symétrique *(owl :SymmetricProperty)*
	$\top \sqsubseteq \leq 1R$	$R^I\ est\ fonctionnelle$	Propriété fonctionnelle *(owl :FunctionalProperty)*
	$\top \sqsubseteq \leq 1R^{-}$	$(R^I)^{-}\ est\ fonctionnelle$	Propriété inverse fonctionnelle *(owl :InverseFunctionalProperty)*
	$Tr(R)$	$(R^I) = (R^I)^{+}$	Propriété transitive *(owl :TransitiveProperty)*

Tableau 3. Syntaxe et sémantique des axiomes définis dans OWL DL

Notons que ce processus d'axiomatisation, *intra-ontologie,* a été déjà effectué pour notre ontologie commune durant son développement. Des exemples d'axiomes et de règles définis dans l'ontologie commune ont été présentés dans la section 4.5.

5.4.1.1 Exemple d'axiomatisation intra-ontologie

Nous présentons dans cette section un exemple d'axiomatisation *intra-ontologie* permettant la description de plusieurs types du feature trou, $CatHole$, dans une ontologie d'application CAO, en l'occurrence CatiaV5.

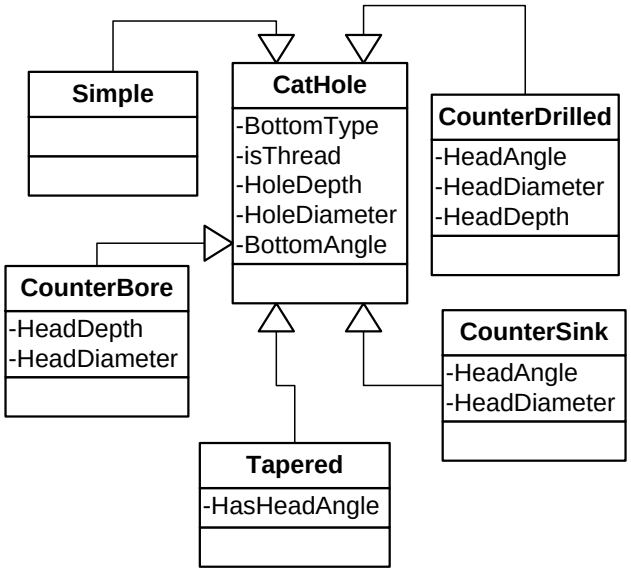


Figure 31. Classification du feature trou "Hole" dans CatiaV5

Simple *Tapered* *Counter Bore* *Counter Sink* *Counter Drilled*

Figure 32. Illustration des types de trous dans CatiaV5.

La figure 31 présente la classification des trous dans le logiciel CatiaV5 (voir l'illustration des trous dans la figure 32).

$CatHole$ désigne une classe « *abstraite* »[24] regroupant les propriétés communes du feature trou. Le trou $Simple$ est le trou de base défini par une profondeur et un diamètre. Le $BottomAngle$ représente la valeur de l'angle du bas de trou, s'il est défini en forme de "V".

D'autres trous complexes définissent des paramètres additionnels nécessaires à leur description. Par exemple le trou conique, $Tapered$, définit l'angle spécifique du trou. Notons que dans les paramètres additionnels, des trous complexes sont représentés par les propriétés suivantes : $hasHeadAngle$, $hasHeadDepth$, et $hasHeadDiameter$. Ces trois propriétés sont définies comme des sous-types de la propriété $hasHoleHeadParameters$.

Dans l'ontologie d'application CatiaV5, la description sémantique du trou conique $Tapered$ est définie comme suit :

$$\begin{aligned} Tapered \equiv\ & Hole \sqcap \left(\neg(\exists\ hasHeadDepth\ Length)\right) \\ & \sqcap (\neg(\exists\ hasHeadDiameter\ Length)) \\ & \sqcap (=1\ hasHoleHeadParameters\ Parameter) \\ & \sqcap (\exists\ hasHeadAngle\ Angle) \end{aligned}$$

Cela signifie qu'un trou conique est un trou, n'ayant qu'un seul paramètre additionnel et doit définir une valeur de l'angle. D'autres définitions sémantiques de trous peuvent être définies selon les valeurs de leurs propriétés, par exemple les trous plats sont des trous n'ayant pas de valeur d'angle de bas.

$$FlatHole \equiv Hole \sqcap \neg(\exists\ hasBottomAngle\ Angle)$$

5.4.2 Axiomatisation inter-ontologies

Pour améliorer l'interopérabilité sémantique entre les ontologies, il s'avère nécessaire d'établir des liens entre les sens explicités par les ontologies. Cette phase consiste donc à définir des relations de correspondance entre des entités provenant des différentes ontologies.

[24] OWL ne permet pas de définir directement une classe abstraite puisqu'il ne partage pas la même interprétation que celle du langage orienté-objet. Cependant, avec la notion « *Covering Axioms* », on peut spécifier qu'une classe est définie par l'union de ses différentes sous-classes.

Ces liens, créés sous forme d'axiomes ou de règles de correspondance, seront stockés dans les ontologies de correspondance *ad-hoc*. Ces correspondances s'effectuent, durant la phase d'apprentissage, au niveau terminologique des ontologies entre des classes ou des propriétés, dont le choix est effectué par des experts du domaine. Ces correspondances servent de base pour le processus de raisonnement, où de nouvelles correspondances sont automatiquement détectées entre les entités des deux ontologies. L'algorithme de raisonnement sera détaillé ultérieurement dans ce chapitre (cf. section 5.5.1).

Ainsi, d'une façon générique, chaque axiome, défini dans l'ontologie de correspondance, met en relation une entité d'une ontologie O_1 avec une entité d'une autre ontologie O_2. Ces axiomes initiaux représentent donc des points d'ancrage reliant les deux ontologies. Un axiome d'ancrage a la forme d'un triplet $< E_1, Corresp, E_2 >$, où $Corresp$ désigne une relation binaire de correspondance entre deux entités. E_1 et E_2 représentent respectivement les entités « antécédente » et « conséquente » de l'axiome tel que $E_1 \in O_1$ et $E_2 \in O_2$. Par exemple, si un axiome définit une relation de subsomption ($\sqsubseteq$) entre deux concepts, alors $< C_1, \sqsubseteq, C_2 >$ signifie que le concept C_1 est subsumé par C_2.

Comme nous l'avons constaté dans le tableau 3 (*cf.* section 5.4.1), différents types de correspondance entre deux entités E_1 et E_2 peuvent être établis, en fonction du type de l'entité, notamment les concepts, les propriétés d'objets ou les propriétés de type de données. Les types de correspondance utilisés dans notre méthode d'intégration sont les suivants :

- *L'équivalence* $(E_1, \equiv, E_2)$: peut être définie entre deux concepts (*equivalentClass*), ou deux propriétés (*equivalentProperty*). Deux concepts sont considérés équivalents si toutes les instances de E_1 sont des instances de E_2, et réciproquement.
- *La subsomption* $(E_1, \sqsubseteq, E_2)$: peut être également définie entre deux concepts (*subClassOf*) ou entre deux propriétés (*subPropertyOf*). Un concept E_1est subsumé par un autre concept E_2 (respectivement E_2 subsume E_1), noté par $E_1 \sqsubseteq E_2$ (respectivement $E_2 \sqsupseteq E_1$), si et seulement si toutes les instances de E_1 sont également des instances de E_2, mais E_2 a des instances qui ne sont pas incluses dans E_1. Le concept E_2 est appelé le subsumant et E_1 le subsumé.
- *La disjonction* $(E_1, \not\equiv, E_2)$: définie uniquement entre deux concepts (*disjointWith*) pour désigner que les deux concepts ne peuvent pas avoir des instances en commun.

Notons que la mise en correspondance des propriétés constitue une étape enrichissante de l'intégration d'ontologies puisque leur implication dans le processus de reconnaissance permet de générer des nouvelles correspondances sémantiques entre les concepts. Par exemple, si une relation Rel_1, définie dans une ontologie O_1, a pour domaine et co-domaine respectivement les deux concepts X_1 et Y_1. Et si Rel_2, est définie comme relation équivalente à Rel_1 dans une autre ontologie O_2, définissant une relation entre X_2 et Y_2 , alors le moteur d'inférence déduit des relations de subsomption entre X_1 et X_2 d'une part, et entre Y_1 et Y_2 d'autre part. Une erreur est générée si, par exemple, X_1 et X_2 sont déclarés des concepts disjoints.

L'algorithme 1, présente le processus de définition des axiomes entre deux entités E_1 et E_2 définies respectivement dans deux ontologies O_1 et O_2 :

AjoutAxioms (OntCorresp, TypeAxiom, E_1, E_2) // ajouter axiome corresp entre $E_1 \in O_1$ et $E_2 \in O_2$
1. //entrée : TypeAxiom, E_1et E_2
2. //Sortie : OntCorresp // Ontologie de correspondance modifiée par l'ajout d'axiome
3. ChargerOntologie (OntCorresp) //Charger les 3 ontologies concernées en mémoire
4. Axiom ax= OntCorresp ->CréerAxiome (TypeAxiom, E_1, E_2) //méthode de factory de l'ontologie
5. OntCorresp ->ajouterAxiome (ax) // Méthode d'ajout d'axiome dans une ontologie
6. Sauvegarder (OntologyCorresp)

Algorithme 1. Algorithme d'ajout des axiomes de correspondance entre deux ontologies

5.4.2.1 Exemple d'axiomatisation inter-ontologies

Nous présentons dans cette section quelques exemples d'axiomatisation inter-ontologies dans le but de créer des correspondances pour le transfert des instances de features de conception entre différents systèmes. Considérons un modèle de pièce conçu dans CatiaV5, illustré dans la figure 33. Ce modèle est composé d'une liste de features de conception, tels que des extrusions (*Pad.1* et *Pad.2*), des poches (*Pocket.1*), des congés (*EdgeFillet.1*), *etc*. Nous supposons convertir ce modèle du CatiaV5 vers le logiciel SolidWorks.

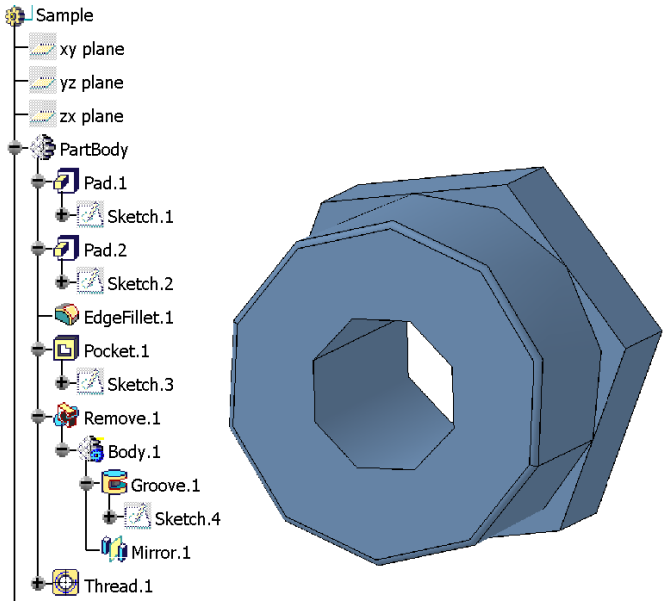


Figure 33. Exemple d'un modèle procédural de pièce en CatiaV5

En effet, le processus de conversion se réalise par l'intermédiaire de l'ontologie commune *CDFO*. Ainsi, les axiomes de correspondance doivent être définis entre les ontologies *CatiaV5* et *CDFO* d'une part et entre *CDFO* et *SolidWorks* d'autre part. L'ontologie commune CDFO est supposée être extensible pour être en mesure de définir les concepts des applications intégrées. La figure 34 illustre deux segments de deux taxonomies spécifiques aux systèmes *CatiaV5* et *SolidWorks* représentant certains concepts existants dans notre exemple (*Extrusion, poche et congé*).

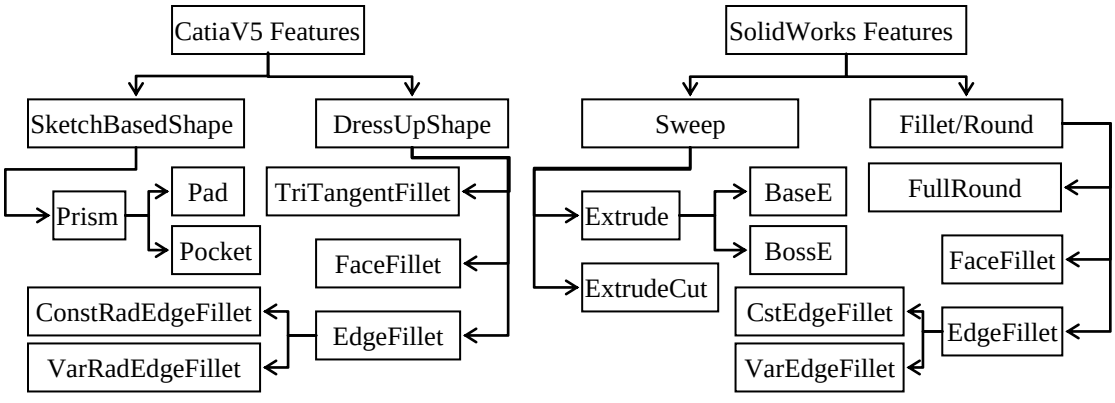


Figure 34. Segments de taxonomies dans CatiaV5 et SolidWorks

Il faut donc définir des axiomes et des règles de correspondance permettant, par exemple, de spécifier une équivalence entre des termes ayant des syntaxes différentes mais la même sémantique. C'est le cas des features de congés d'arêtes, $Catia{:}\,ConstRadEdgeFillet$,

et $SW: CstEdgeFillet$, qui sont considérés par des experts comme des concepts équivalents. La figure 35 illustre les correspondances entre les features de CatiaV5 et de SolidWorks en passant par notre ontologie commune CDFO.

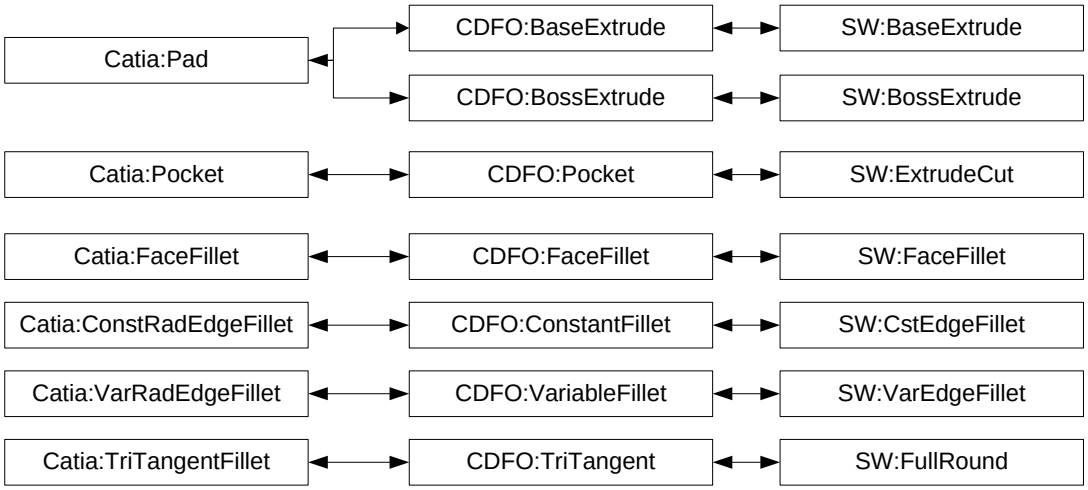


Figure 35. Correspondances entre des concepts de CatiaV5, CDFO et SolidWorks

Toutefois, des cas plus complexes peuvent se présenter, où des correspondances plus complexes doivent être créées entre des concepts en considérant le contexte dans lequel ces concepts sont définis. C'est le cas de l'équivalence sémantique « *conditionnelle* » entre le feature $Catia: Pad$ d'une part et les deux features $SW: BaseExtrude$ et $SW: BossExtrude$ d'autre part. En effet, l'équivalence est déterminée en fonction de la propriété d'objets $hasPrevious$. Cette dernière permet de spécifier l'ordre du feature dans l'arbre de spécification du modèle de produit. Cette condition est représentée dans les deux axiomes suivants.

Axiome 1 : Le concept $CDFO: BaseExtrude$ est équivalent à $Catia: Pad$, si tous ses prédécesseurs dans l'arbre sont des esquisses, $Catia: Sketch$, autrement dit, aucun de ses parents ne doit être un feature. C'est l'exemple de l'instance *Pad.1* (figure 33) définie comme le premier feature dans l'arbre de spécifications ayant comme seul prédécesseur l'esquisse du feature.

$$CDFO: BaseExtrude \equiv Catia: Pad \sqcap \forall\, Catia: hasPrevious\ Catia: Sketch$$

Axiome 2 : Selon le même principe de l'axiome 2, le concept $CDFO: BossExtrude$ est équivalent à $Catia: Pad$, si au moins un de ses parents est une instance du feature $Catia: Feature$. C'est le cas de l'instance *Pad.2* précédée par le feature *Pad.1* (figure 33).

$$CDFO{:}\,BossExtrude \equiv Catia{:}\,Pad \sqcap \exists\, Catia{:}\,hasPrevious\; Catia{:}\,Feature$$

Notons que dans notre ontologie commune, nous avons présenté les deux spécifications du feature d'extrusion défini dans SolidWorks, puisque notre ontologie commune est extensible et destinée à représenter tous les concepts des systèmes intégrés.

5.5 Reconnaissance de correspondance

L'axiomatisation des ontologies présentée dans la section précédente s'effectue manuellement par des experts du domaine durant la phase d'apprentissage des ontologies. Le but est d'établir des relations de correspondance, définies comme des axiomes d'ancrage entre les entités d'une même ontologie ou de différentes ontologies. Toutefois, les ontologies OWL DL offrent un avantage fondamental grâce aux capacités de raisonnement fournies par les moteurs d'inférence. Dans cette section, nous mettons en relief le processus de raisonnement permettant la déduction automatique des correspondances supplémentaires pour renforcer l'intégration sémantique des données.

Un processus de raisonnement repose effectivement sur des opérations d'inférence fournies par un raisonneur. Le choix du raisonneur dépend particulièrement de la complétude « *completeness* » et de la correction « *soundness* » des algorithmes sur lesquels s'appuie le raisonneur. Notons qu'un système déductif est *correct*, si les inférences produites sont en accord avec la sémantique associée au système, autrement dit, ce qui est vrai sur le plan syntaxique l'est également sur le plan sémantique. D'ailleurs, un système déductif est complet si toutes les formules avérées sur le plan sémantique, peuvent être démontrées sur le plan syntaxique (Napoli, 2005). Le raisonneur Pellet (Parsia, et al., 2004), utilisé dans notre méthode d'intégration, s'appuie sur un algorithme décidable et complet, de manière qu'il puisse déduire toutes les correspondances « *inférables* » dans une base de connaissance.

Pour exécuter la reconnaissance des correspondances sémantiques, le processus de raisonnement reçoit en entrée l'ontologie de correspondance, important les deux ontologies à intégrer et définissant un ensemble d'axiomes et/ou de règles de correspondance entre des entités provenant des deux ontologies. Cette ontologie est chargée dans le moteur d'inférence afin de détecter automatiquement d'autres correspondances entre les entités. En effet, les opérations principales liées au raisonnement peuvent être menées aux tests de subsomption

entre deux entités, ou de satisfiabilité d'un concept. Le moteur d'inférence, utilisé dans notre méthode, applique la méthode des tableaux sémantiques présentée dans (Baader, et al., 2007). Le résultat de ce processus est un ensemble de triplets $< E_1, Corresp, E_2 >$ définissant des correspondances entre deux entités E_1 et E_2 des deux ontologies O_1 et O_2 respectivement. Les correspondances inférées peuvent être des relations d'équivalence ou de subsomption. Une fois les correspondances inférées sont validées, elles peuvent être ajoutées à l'ontologie de correspondance comme étant des axiomes d'ancrage pour des opérations supplémentaires de raisonnement.

Les inférences produites peuvent être effectuées à deux niveaux : le niveau de raisonnement terminologique et le niveau de raisonnement factuel. Un algorithme spécifique à chaque niveau sera développé et détaillé dans les sous sections suivantes. Au niveau terminologique, la recherche des correspondances s'effectue pour les définitions des concepts et des propriétés. Au niveau factuel, les inférences permettent de détecter les types des instances représentant un modèle de produit dans l'ontologie cible. En cas d'absence d'équivalence, une méthode de recherche de similarité serait effectuée, comme détaillée dans le chapitre suivant.

5.5.1 Algorithme au niveau terminologique

L'objectif du raisonnement terminologique de notre méthode est de déduire les correspondances entre les définitions des concepts et des propriétés de deux ontologies. A ce niveau, les instances, représentant des modèles de produit, ne sont pas considérées. En effet, les opérations principales de raisonnement terminologique sont basées sur un test de subsomption ou de satisfiabilité. Le raisonneur effectue ainsi un test de subsomption mutuel entre deux entités E_1 et E_2. Si une relation de subsomption est inférée dans un seul sens, alors la correspondance est du type « *Subsomption* ». En revanche, si la subsomption est inférée dans les deux sens, ceci mène à une relation d' « *Equivalence* » entre les deux entités.

L'algorithme de reconnaissance des correspondances est illustré dans l'algorithme 2. Cet algorithme définit la procédure de récupération de la liste de correspondances inférées entre les entités $E_1 \in O_1$ et $E_2 \in O_2$. L'algorithme est construit d'une façon générique, où l'entité traitée peut être un concept, une propriété d'objet ou une propriété de type de données. Il s'agit donc de traiter toutes les entités de l'ontologie O_1, et de trouver toutes les entités correspondantes dans O_2. Comme le figurent les lignes 8, 12, et 20 de notre algorithme, le

raisonneur dispose de méthodes permettant d'inférer les entités équivalentes, les sous et les super entités. Cette fonctionnalité dépend de la définition de l'entité dans l'ontologie.

Dans une ontologie, les concepts peuvent être primitifs (⊑) ou définis (≡). Les concepts primitifs représentent des définitions atomiques des classes et servent de base à la construction des concepts définis. Cette distinction a un impact sur le comportement des moteurs d'inférence vis-à-vis des concepts d'une ontologie. Effectivement, les raisonneurs ne peuvent pas effectuer des inférences sur les concepts primitifs.

GetInferenceCorresp (OntologieCorresp) //Récupérer liste de correspondances inférées
1. //entrée : OntCorresp
2. //Sortie : ListeTriplets $(E_1, Corresp, E_2)$
3. ChargerOntologie (OntCorresp) //Charger les 3 ontologies concernées en mémoire
4. ChargerRaisonneur (OntCorresp) // Créer le raisonneur et charger les ontologies dedans
5 . **Pour** chaque entité E_1 dans O_1 **faire** // traitement de tous les concepts de 1ère ontologie
6. Entité [] L // Liste de entités correspondantes inférées
7.
8. L = raisonneur.GetEquivalent (E_1) //Méthode du raisonneur, retourne entités équivalentes
9. **Pour** chaque E_2 de L **faire**
10. Si $E_2 \in O_2$ ➔ ListeTriplets = ListeTriplets + $(E_1, Equivalent, E_2)$ //Ajouter $(E_1 \equiv E_2)$
11. **Fin** pour

12. L = raisonneur.GetSubEntities (E_1) //Méthode du raisonneur, retourne sous entités de E1
13. **Pour** chaque E_2 de L **faire**
14. **Si** $C_2 \in O_2$ **alors**
15. **Si** $(E_1 \sqsubseteq E_2)$ ListeTriplets=ListeTriplets+$(E_1, Equivalent, E_2)$ //Subsomption mutuelle
16. **Sinon** ListeTriplets=ListeTriplets +$(E_1, SuperEntity, E_2)$ //Subsomption unique
17. **Fin Si**
18. **Fin Si**
19. **Fin** pour

20. L = raisonneur.GetSuperEntities (E_1) //Méthode du raisonneur, retourne sous entités de E1
21. **Pour** chaque E_2 de L **faire**
22. **Si** $C_2 \in O_2$ **alors**
23. **Si** $(E_2 \sqsubseteq E_1)$ ListeTriplets=ListeTriplets+$(E_1, Equivalent, E_2)$ //Subsomption mutuelle
24. **Sinon** ListeTriplets=ListeTriplets +$(E_1, SubEntity, E_2)$ //Subsomption unique
25. **Fin Si**
26. **Fin Si**
27. **Fin** pour
28. **Fin** pour
28. **retourne** ListeTriplets ;

Algorithme 2. Reconnaissance des correspondances au niveau terminologique

Par conséquent, nous supposons que les concepts primitifs de deux ontologies soient liés explicitement par des axiomes d'ancrage créés dans l'ontologie de correspondance. Ces axiomes permettent de spécifier, par exemple, que deux termes syntaxiquement différents,

mais ayant la même sémantique, sont équivalents. Les correspondances des concepts définis sont inférées en fonction des correspondances des leurs concepts primitifs.

Considérons l'exemple suivant : Considérons les trois concepts $A, B\ et\ C$ d'une ontologie O_1, tel que A et B désignent deux concepts primitifs et C un concept défini par l'intersection de A et B. Considérons également les trois concepts $A', B' et\ C'$ d'une ontologie O_2, tel que A' et B' désignent deux concepts primitifs et C' un concept défini par l'intersection de A' et B'. Ceci est représenté par : $(C \equiv A \cap B)$ tel que $A, B, C \in O_1$ et $(C' \equiv A' \cap B')$ tel que $A', B', C' \in O_2$. Si les relations d'équivalence entre les concepts primitifs sont définies par : $(A \equiv A')$ et $(B \equiv B')$ alors une correspondance d'équivalence doit être inférée entre C et C', $(C \equiv C')$.

5.5.1.1 Exemples d'inférence terminologique

Exemple 1 : Nous présentons dans cette section un exemple sur le raisonnement terminologique effectué entre des entités de deux ontologies d'application CatiaV5 et SolidWorks. Nous nous intéressons plus spécifiquement aux features trous définis dans ces deux systèmes. La figure 36 illustre la classification de ces features dans les deux systèmes. En effet, le feature trou, $Hole$, est déclaré dans les deux ontologies d'application comme étant un concept primitif. Les sous-classses des trous représentent des concepts complexes, définis en termes de spécifications attribuées à chaque type de trou. (Voir l'exemple de description du trou conique $Tapered$ de CatiaV5 présenté dans la section 5.4.1.1).

Dans cet exemple, nous avons défini un axiome d'équivalence entre les deux concepts primitifs $Hole$ des deux systèmes. Cet axiome d'ancrage est représenté dans la figure 36 par un trait continu ayant deux flèches. Il permet aux moteurs d'inférence de détecter des correspondances supplémentaires. Le résultat obtenu consiste en un ensemble de correspondances inférées représentant des relations de subsomption entre différentes entités des deux ontologies. Par exemple, l'ensemble des trous de l'ontologie CatiaV5 a été inféré comme sous-classe du trou de SolidWorks. De plus, le trou dans SolidWorks est inféré comme étant un feature à base d'esquisse dans l'application CatiaV5. Les correspondances inférées sont représentées par des traits discontinus dans la figure 36.

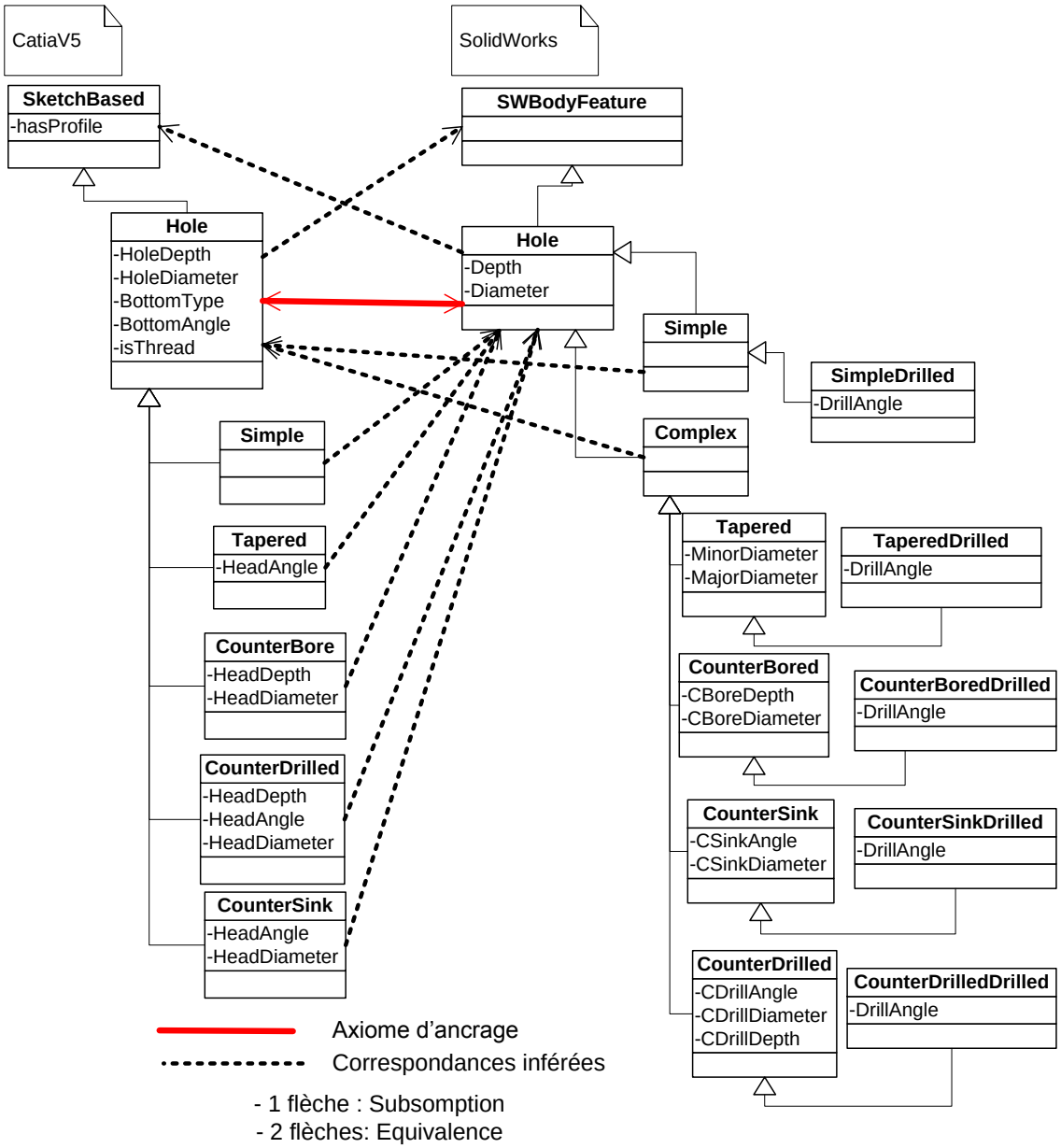


Figure 36. Raisonnement sur les features « Hole » dans CatiaV5 et SolidWorks

Exemple 2 : Nous nous intéressons dans cet exemple à la définition d'un axiome de correspondance entre deux propriétés des deux ontologies de CatiaV5 et SolidWorks. Considérons l'exemple des propriétés des trous définissant l'angle de bas du trou dans ces deux systèmes : $Catia{:}\,hasBottomAngle$ et $SW{:}\,hasDrillAngle$. Les descriptions de ces deux propriétés dans les deux ontologies sont représentées respectivement par :

$$Catia{:}\,hasBottomAngle\ (Catia{:}\,Hole, Angle)$$

$$SW{:}\,hasDrillAngle\ (SW{:}\,ComplexHole, Angle)$$

Ces deux axiomes définissent le domaine et le co-domaine de chaque propriété. En effet, la création d'un axiome d'équivalence entre ces deux propriétés, $Catia{:}\,hasBottomAngle \equiv SW{:}\,hasDrillAngle$, engendre la reconnaissance d'un ensemble de correspondances supplémentaires entre des concepts des deux ontologies. La liste des correspondances inférées est la suivante :

$$SW{:}\,SimpleDrilled \sqsubseteq Catia{:}\,Hole$$

$$SW{:}\,TaperedDrilled \sqsubseteq Catia{:}\,Hole$$

$$SW{:}\,CounterBoreDrilled \sqsubseteq Catia{:}\,Hole$$

$$SW{:}\,CounterSinkDrilled \sqsubseteq Catia{:}\,Hole$$

$$SW{:}\,CounterDrilledDrilled \sqsubseteq Catia{:}\,Hole$$

5.5.2 Algorithme d'échange au niveau factuel

Un modèle de produit CAO se représente par des instances créées dans l'ontologie d'application correspondante. Cette phase consiste à développer un algorithme de raisonnement au niveau factuel, dont l'objectif est de pouvoir transférer des modèles de produit de l'ontologie source O_1 vers l'ontologie cible O_2, par inférence des types d'instances dans l'ontologie cible. Ainsi, pour chaque instance i, appartenant à un concept ou à une propriété $E_1, (E_1 \in O_1)$, l'algorithme développé vise à détecter son type correspondant E_2 dans l'ontologie O_2. Dans le cas où aucune correspondance n'a été détectée, une recherche de similarité sera effectuée, dont la procédure sera présentée dans le chapitre suivant. Cet algorithme est illustré dans l'algorithme 3.

Notons qu'au niveau factuel figurent essentiellement les instances des concepts et des propriétés. Ainsi, deux types d'assertions se présentent :

- $C(a)$: où C est un concept, et a est une extension du concept, *i.e.* un individu.

- $r(a,b) tel\ que\ r \in R$: où r est une instance de la propriété R, a est un individu, et b est un individu ou une valeur de données, en fonction du type de R, c'est-à-dire s'il s'agit respectivement d'une propriété d'objets ou une propriété de type de données. L'assertion $r(a,b)$ désigne une relation r entre a et b.

```
TranslateModelIndividuals (OntologieCorresp)  //Transférer les instances par inférence de type
1. //entrée : OntCorresp
2. //Sortie : ListeTriplets (i1, Corresp, E2) // Liste des instances i1 et leur types corresp E2 dans O2
3. ChargerOntologie (OntCorresp)            //Charger les 3 ontologies concernées en mémoire
4. ChargerRaisonneur (OntCorresp)           // Créer le raisonneur et charger les ontologies dedans
5. Property [] LP ;                         // Liste des attributs ou de relations attribué à un individu
6. Model3D model3D = O1->AvoirModele3D( )  //Récupérer l'instance du modèle 3D de O1
7. AvoirTypeinfere (model3D, Corresp, E2) //Avoir le type inféré dans O2
8.  LP = model3D->AvoirPropriétés () ;  //Recheche des attributs et relations du modèle
9.  Pour chaque pté dans LP faire
10.       AvoirTypeinfere (pté, Corresp, E2) //Inférer le type de chaque attribut dans O2
11. Fin Pour
12. Part [] pièces = model3D->AvoirListePièces ( ) //Avoir les pièces associées à un modèle
13. Pour chaque pièce dans pièces faire
14.       LP = pièce ->AvoirPropriétés ( ) ;  //Avoir les propriétés de la pièce
15.       Pour chaque pté dans LP faire
16.            AvoirTypeinfere (pté, Corresp, E2) //Inférer le type de propriété dans O2
17.       Fin Pour
18.       Body [] volumes = pièce->AvoirListeVolumes ( ) //Avoir liste de volumes d'une pièce
19.       Pour chaque volume dans volumes faire
20.            LP = volume->AvoirPropriétés ( ) ;  //Propriétés d'un volume
21.            Pour chaque pté dans LP faire
22.                 AvoirTypeinfere (pté, Corresp, E2) //Inférer le type dans O2
23.            Fin Pour
24.            Feature [] features = volume ->AvoirListeFeatures ( ) // liste de features d'un volume
25.            Pour chaque feature dans features faire
26.                 LP = feature->AvoirPropriétés ( ) ;  //Propriétés d'un feature
27.                 Pour chaque pté dans LP faire
28.                      AvoirTypeinfere (pté, Corresp, E2) //Inférer le type dans O2
29.                 Fin Pour
30.                 … // Récupérer la géométrie, etc….
31.            Fin Pour //feature
32.       Fin Pour  //Volume
33.  Fin Pour //pièce
```

Algorithme 3. Algorithme de conversion des instances entre deux ontologies

La procédure de notre algorithme est la suivante. Un modèle du produit ($model3D$) est présenté comme une instance dans l'ontologie d'application dans laquelle ce modèle a été créé. Nous supposons que le modèle 3D en question est un modèle à base de features, c'est-à-dire qu'il est structuré dans un arbre de spécification, représentant l'historique de construction du modèle, les features inclus, ainsi que les paramètres et les contraintes associés.

Concrètement, notre algorithme de conversion entame la recherche de l'instance représentant la racine de l'arbre de spécifications, à savoir l'instance $Model3D(model3D)$ (*ligne 6*). Cette instance est caractérisée par un ensemble de propriétés $r(model3D, b)$ associées à cette instance. Pour chaque propriété, notre algorithme procède à la recherche du

type E_2 correspondant dans l'ontologie destinataire O_2 *(lignes 10,16,22 et28)*. L'algorithme de recherche des types correspondants est illustré dans l'algorithme 4. Un modèle de produit peut être défini comme étant une pièce ou un assemblage. Un assemblage est composé à son tour d'une liste de sous-assemblages ou de pièces. Pour des raisons de visibilité, nous avons simplifié la présentation de l'algorithme en considérant le traitement des pièces uniquement (*ligne 12*).

Une pièce constitue un élément de base dans un modèle de produit. Pour chaque instance de pièce, on cherche l'ensemble de ses propriétés, y compris la liste de ses composants. Chaque instance de ces propriétés fait ensuite l'objet d'une procédure ayant pour objectif la détection du type auquel elle appartient dans l'ontologie cible. Les composants peuvent être des volumes solides ou surfaciques (*ligne 18*). Un volume est à son tour composé d'une liste de features (*ligne 24*). Ensuite, pour chaque instance de feature, nous cherchons ses propriétés, leurs types dans O_2, ainsi que le type correspondant du feature lui-même. L'algorithme cherche également à traiter les informations géométriques et topologiques associées au feature, que nous ne présentons pas dans l'algorithme.

Enfin, pour chaque instance récupérée, que ça soit une instance de classe $C_1(a) \in O_1$ ou de propriété $r(a,b) \in R_1$ dans O_1, notre algorithme procède à la recherche des types correspondants dans l'ontologie cible, i.e. les concepts $C_2(a) \in O_2$ ou les propriétés $r(a,b) \in R_2$ dans O_2. Si aucune correspondance n'a été détectée, notre algorithme effectue une recherche des types similaires dans l'ontologie O_2.

AvoirTypeinfere (i_1, Corresp, E_2) //Chercher pour $i_1 \in O_1$ le type correspondant $E_2 \in O_2$
1. //entrée : OntCorresp
2. //Sortie : ListeTriplets ($i_1, Corresp, E_2$) // Liste des instances i_1 et leur types corresp E_2 dans O_2
3. ChargerOntologie (OntCorresp) //Charger les 3 ontologies concernées en mémoire
4. ChargerRaisonneur (OntCorresp) // Créer le raisonneur et charger les ontologies dedans
5. Entite [] L ; // Liste de entités correspondantes inférées
6. Bool direct = true // pour inférences des types directes des instances
7. L = raisonneur.GetTypes (i_1, direct) //Méthode raisonneur, retourne entités types d'une instance
8. **Pour** chaque E_2 de L **faire**
9. Si $E_2 \in O_2$ ➔ ListeTriplets = ListeTriplets + ($i_1, appartenance, E_2$) //Ajouter ($E_1 \equiv E_2$)
10. **Fin** pour

Algorithme 4. Algorithme de conversion des instances entre deux ontologies

5.5.3 Raisonnement par règles SWRL

Nous avons montré dans le chapitre 3 certaines restrictions liées à l'expressivité du langage OWL DL, notamment sur la composition des propriétés (*cf.* section 3.2.2). Il nous semble donc nécessaire d'intégrer dans notre méthode de mise en correspondance des règles du Web Sémantique afin de combler ces lacunes. Dans ce cadre, nous avons utilisé le langage des règles, SWRL, pour la création des règles de correspondance ayant la forme, $A \rightarrow B$, c'est-à-dire si la condition A est satisfaite, alors la conséquence B doit l'être également. Nous nous intéressons, dans cette section, à la création des règles dans l'ontologie de correspondance, menant à établir des correspondances conditionnelles entre les entités de deux ontologies. Nous avons donc utilisé un moteur d'inférence basé sur les règles, en l'occurrence JESS (Friedman-Hill, 2007). Dans la suite, nous présentons un exemple qui prouve la nécessité de la création des règles pour le transfert d'un modèle de produit ainsi que les inférences obtenues.

5.5.3.1 Cas d'utilisation

Nous présentons dans cette section des exemples de création d'axiomes et de règles de correspondance. Nous considérons la classification du feature trou, $Hole$, dans les deux logiciels *CatiaV5* et *SolidWorks*. Cette classification a été présentée dans la figure 36. Nous allons procéder dans la suite à la définition des axiomes et des règles permettant la mise en correspondance des trous, notamment les trous complexes des deux ontologies. Cette mise en correspondance est illustrée dans la figure 37.

D'une part, les trous dans *CatiaV5* sont définis comme des sous-classes du concept $Hole$, dont ils héritent les paramètres de base : la profondeur et le diamètre. Les trous complexes de *CatiaV5* sont définis en fonction de trois paramètres additionnels : $HeadAngle$, $HeadDiameter$, et $HeadDepth$. La représentation sémantique d'un trou complexe est donc spécifiée en fonction d'une combinaison de ces trois paramètres. Par exemple, le trou $CounterDrill$ intègre dans sa définition l'ensemble de ces trois paramètres. Les autres trous complexes définissent une combinaison de deux paramètres additionnels, ou uniquement d'un seul comme le trou conique $Tapered$.

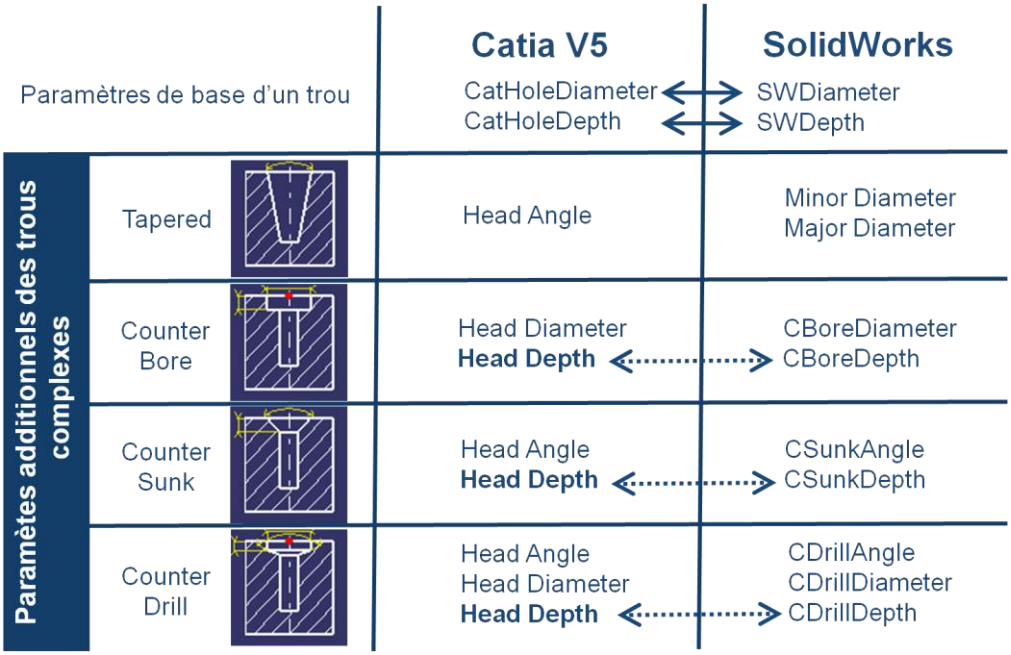


Figure 37. Mise en correspondance des paramètres des trous de CatiaV5 et SolidWorks

D'autre part, les trous complexes de *SolidWorks* sont définis en fonction d'une liste de paramètres additionnels spécifiques à chaque type de trou. Par exemple, le trou $CounterDrill$ est défini en fonction de ses trois paramètres spécifiques : $CounterDrillAngle$, $CounterDrillDepth$, et $CounterDrillDiameter$. En effet, un paramètre additionnel d'un trou complexe dans *SolidWorks* est spécifique au type de trou.

Par conséquent, les équivalences entre les paramètres des trous de *CatiaV5* et ceux de *SolidWorks*, ne peuvent pas être définies d'une manière générique. C'est l'exemple du paramètre $HeadDepth$ du CatiaV5 qui peut être équivalent à $CBoreDepth$, $CSunkDepth$, ou $CDrillDepth$ en fonction du type de trou défini. Ceci est représenté dans la figure 37 par les traits discontinus. Des règles « *contextuelles* » peuvent être élaborées, considérant le type du trou, afin d'établir des correspondances entre les paramètres des trous dans *CatiaV5* et *SolidWorks*.

Nous définissons dans la suite des exemples d'axiomes et de règles de correspondance définis pour mettre en place les relations entre les paramètres des trous des deux systèmes. Les deux axiomes suivants décrivent des équivalences directes entre des propriétés des trous de CatiaV5 et SolidWorks (représentés par les traits continus dans la figure 37).

Axiome 1 : La propriété $hasHoleDiameter$ dans *CatiaV5* est définie comme étant équivalente à la propriété $hasDiameter$ dans *SolidWorks :*

$$Catia{:}\ hasHoleDiameter \equiv SldWorks{:}\ hasDiameter$$

Axiome 2: La propriété $hasHoleDepth$ dans *CatiaV5* est équivalente à la propriété $hasDepth$ dans *SolidWorks*.

$$Catia{:}\ hasHoleDepth \equiv SldWorks{:}\ hasDepth$$

Cependant, pour les équivalences « *contextuelles* », autrement dit dépendantes des types des trous, des règles de correspondance sémantique plus complexes ont été créées en utilisant SWRL. Ces règles consistent à définir des correspondances entre les paramètres des trous complexes des deux systèmes *CatiaV5* et *SolidWorks*. Ces règles permettent de prendre en considération le contexte dans lequel ces paramètres ont été utilisés. Ces règles sont décrites comme suit :

Règle 1 :

$$Catia{:}\ Counterbored(?x) \wedge Catia{:}\ Length(?y) \wedge Catia{:}\ hasHeadDepth(?x, ?y) \rightarrow SldWorks{:}\ hasCounterBoreDepth(?x, ?y)$$

Règle 2 :

$$Catia{:}\ Counterbored(?x) \wedge Catia{:}\ Length(?y) \wedge Catia{:}\ hasHeadDiameter(?x, ?y) \rightarrow SldWorks{:}\ hasCounterBoreDiameter(?x, ?y)$$

Règle 3 :

$$Catia{:}\ Counterdrilled(?x) \wedge Catia{:}\ Angle(?y) \wedge Catia{:}\ hasHeadAngle(?x, ?y) \rightarrow SldWorks{:}\ hasCounterDrillAngle(?x, ?y)$$

Règle 4 :

$$Catia{:}\ Counterdrilled(?x) \wedge Catia{:}\ Length(?y) \wedge Catia{:}\ hasHeadDepth(?x, ?y) \rightarrow SldWorks{:}\ hasCounterDrillDepth(?x, ?y)$$

Règle 5 :

$$Catia{:}\,Counterdrilled(?\,x) \wedge Catia{:}\,Length(?\,y) \wedge Catia{:}\,hasHeadDiameter(?\,x,?\,y) \rightarrow SldWorks{:}\,hasCounterDrillDiameter(?\,x,?\,y)$$

Règle 6 :

$$Catia{:}\,Countersunk(?\,x) \wedge Catia{:}\,Angle(?\,y) \wedge Catia{:}\,hasHeadAngle(?\,x,?\,y) \rightarrow SldWorks{:}\,hasCountersinkAngle(?\,x,?\,y)$$

Règle 7 :

$$Catia{:}\,Countersunk(?\,x) \wedge Catia{:}\,Length(?\,y) \wedge Catia{:}\,hasHeadDiameter(?\,x,?\,y) \rightarrow SldWorks{:}\,hasCountersinkDiameter(?\,x,?\,y)$$

La 1ère règle indique que : on considère deux variables d'instances x et y définies dans *CatiaV5*. Si x est une instance du trou *CounterBored*, ayant une relation de profondeur *hasHeadDepth* avec y, où y est une longueur, alors la relation équivalente dans *Solidworks* est *hasCounterBoreDepth.* C'est-à-dire l'instance x est liée à y dans *SolidWorks* par l'intermédiaire de la relation *hasCounterBoreDepth*. Les autres règles définissent les équivalences contextuelles de la même façon.

Notons que ces règles définissent des équivalences directionnelles, c'est-à-dire de *CatiaV5* vers *Solidworks*. Pour augmenter l'efficacité de l'interopérabilité sémantique, ces règles de correspondance doivent être bidirectionnelles entre les ontologies d'application et l'ontologie commune CDFO. Par conséquent, des règles de correspondance supplémentaires doivent être également définies pour assurer le transfert de *SolidWorks* vers *CatiaV5*. Ces règles ont été implémentées et testées dans l'éditeur des ontologies *Protégé*, et des instances des trous créés dans *CatiaV5* ont été automatiquement inférées comme étant des instances des trous correspondants dans l'ontologie de SolidWorks.

Exemple : Supposons l'instance $CounterBoredHole\_1$ appartenant au feature "$Catia{:}\,CounterBore$", représentée par $Catia{:}\,CounterBore\ (CounterBoredHole\_1)$. Cette instance est caractérisée par les deux propriétés suivantes :

$$Catia{:}\,hasHeadDiameter\ (CounterBoredHole\_1, Diameter\_1)$$

$$Catia{:}\,hasHeadDepth\ (CounterBoredHole\_1, Depth\_1)$$

où $Diameter\_1$ et $Depth\_1$ représentent deux instances de définition des valeurs de dimension. Ainsi, l'appel au moteur d'inférence JESS a engendré la déduction des deux instanciations suivantes :

$$SW{:}\,hasCounterBoreDiameter\ (Catia{:}\,CounterBoreHole\_1, Diameter\_1)$$

$$SW{:}\,hasCounterBoreDepth\ (Catia{:}\,CounterBoreHole\_1, Depth\_1)$$

Ces deux instanciations inférées permettent de déduire que l'instance $Catia{:}\,CounterBoreHole\_1$ appartient aux concepts définis comme « *domaine* » des deux propriétés d'objet $SW{:}\,hasCounterBoreDepth$ et $SW{:}\,hasCounterBoreDiameter$, c'est-à-dire elle appartient au concept $SW{:}\,ComplexHole$.

5.6 Synthèse

Nous avons présenté dans ce chapitre notre méthode d'intégration des ontologies des systèmes CAO afin d'échanger des modèles de produit tout en gardant la sémantique des données. Cette méthode se base sur un alignement binaire des ontologies effectué par la mise en correspondance des entités des deux ontologies. L'avantage de notre méthode réside dans l'exploitation des technologies du Web Sémantique, telles que le langage OWL DL et le langage des règles sémantiques SWRL.

Notre méthode repose essentiellement sur deux grandes étapes : l'axiomatisation et le raisonnement. La première consiste à enrichir la sémantique des données contenues dans les ontologies afin d'établir des liens sémantiques entre les entités d'une même ontologie (intra-ontologie) ou entre des entités de différentes ontologies (inter-ontologies). Cette axiomatisation s'applique aux concepts ainsi qu'aux propriétés d'objets ou de type de données. Dans l'axiomatisation inter-ontologies, une troisième ontologie est créée afin de stocker les correspondances sémantiques des deux ontologies en question. Cette ontologie de correspondance sert d'entrée pour le processus de raisonnement.

La deuxième étape de notre méthode consiste à exploiter le raisonnement déductif des moteurs d'inférence basés sur les logiques de description. Ceci permet de reconnaître des correspondances supplémentaires entre les entités provenant de deux ontologies différentes.

Cette reconnaissance de correspondance sémantique repose sur les descriptions formelles des concepts contenus dans les ontologies.

Le raisonnement est effectué au niveau terminologique et factuel. Le premier vise à détecter les correspondances sémantiques des définitions des concepts et des propriétés. Le deuxième permet de transférer les instances d'une ontologie source vers une ontologie cible, par l'intermédiaire des inférences des types correspondants. Dans le cas d'absence de correspondance, notre algorithme de raisonnement lance une méthode de recherche des similarités. Cette méthode sera présentée dans le chapitre suivant. Plusieurs exemples ont été fournis dans ce chapitre afin de clarifier l'application de nos algorithmes dans notre méthode d'intégration.

Chapitre 6 Mesure de similarité sémantique

6.1 Introduction

Nous avons présenté dans le chapitre 5 notre méthodologie d'alignement d'ontologies développées dans le cadre d'échange des modèles CAO à base de features. Cette méthode se base sur la création des axiomes et des règles de correspondance afin d'établir des liens sémantiques entre les ontologies concernées. Ensuite, les services de raisonnement basés sur les logiques de description permettent d'inférer des correspondances supplémentaires entre les entités des différentes ontologies en s'appuyant sur les définitions formelles de ces entités.

Ces méthodes d'alignement sémantique basées sur l'utilisation d'un moteur d'inférence sont des méthodes exactes. Elles se limitent particulièrement à détecter les relations de correspondance exacte entre les entités, telles que les relations d'équivalence ou de subsomption. Pourtant, il est parfois nécessaire de concevoir des méthodes d'alignement approximatif qui tentent de calculer des mesures de similarité entre les différentes entités afin de fournir aux utilisateurs des outils d'aide à l'intégration. La mesure de similarité entre deux concepts est particulièrement importante lorsqu'il s'agit d'évaluer le poids et la direction du lien entre les deux concepts, plutôt que d'identifier la simple présence ou l'absence de ce lien.

En effet, durant la phase d'échange d'un modèle de produit entre deux ontologies, il arrive que certaines instances d'une ontologie ne puissent pas être transférées vers une autre ontologie. Ceci revient au fait que les concepts utilisés dans un système n'ont pas d'équivalence exacte dans l'autre système. Certaines raisons menant à l'incapacité de détection des liens exacts sont présentées dans (Patil, 2005). Par exemple, des concepts d'un même domaine peuvent avoir des structures différentes, adaptées aux besoins des applications qui les intègrent. Ainsi, des attributs peuvent être utilisés en plus ou en moins pour définir un même concept.

Nous visons dans ce chapitre à étendre notre méthode d'alignement automatique, basée sur la reconnaissance des correspondances sémantiques, par le développement d'une

méthode de mesure de similarité pour les entités dépourvues de relations d'équivalence dans les autres ontologies. Il consiste donc à fournir des moyens permettant une correspondance sémantique approximative entre les entités en question. Ceci permet d'enrichir notre méthode d'alignement pour servir de technique de base pour des scénarios interactifs, par exemple pour l'utilisation des résultats obtenus comme une suggestion à l'utilisateur.

Dans ce contexte, nous allons définir les différents facteurs intervenant dans notre méthode de calcul. Il est d'abord supposé que les ontologies utilisées dans notre méthode d'alignement soient représentées avec le langage OWL DL. Ainsi, notre méthode de mesure de similarité sera donc appropriée aux logiques de description basées sur la variante correspondante au langage OWL DL, soit $SHOIN(D)$.

6.2 Méthodes actuelles de similarité

La notion de similarité a été empruntée à la philosophie et a été établie afin de déterminer pourquoi et comment les entités sont regroupées en catégories, et pourquoi certaines catégories sont comparables les unes aux autres, tandis que d'autres ne le sont pas (Medin, et al., 1993). Dans le domaine d'intégration des connaissances, différentes approches d'alignement d'ontologies sont basées sur une mesure de similarité entre les différentes entités. Ainsi, l'alignement de deux ontologies consiste à trouver des correspondances entre les entités qui sont sémantiquement similaires (Ehrig, et al., 2004). D'une manière formelle, l'alignement est défini par la fonction map comme suit (Zghal, et al., 2007) :

$$map : O \rightarrow O' \; tel \; que \; map\,(e_1) = \; e_1' \; si \; Sim(e_1, e_1') > t,$$

où O et O' sont deux ontologies à aligner, t désigne un seuil minimal de similarité appartenant à l'intervalle [0,1], c'est-à-dire le niveau minimum accepté pour que deux entités soient considérées similaires, $e_1 \in O$ et $e_1' \in O'$. e_1 et e_1' représentent les entités des deux ontologies. La fonction de similarité $Sim : O \times O \rightarrow \mathbb{R}$ est une fonction d'une paire d'entités vers une valeur réelle définissant la similarité des deux entités. Cette fonction est caractérisée par :

$$\forall\, x, y \in O, Sim\,(x, y) \geq 0 \qquad (positivité)$$

$$\forall\, x \in O, \forall\, y, z \in O, Sim\,(x, x) \geq Sim\,(y, z) \qquad (maximalité)$$

Dans la littérature, différents modèles de mesure de similarité ont été définis (Goldstone, et al., 2005) (Thibau, 1997). Parmi les travaux effectués sur la quantification de la similarité, nous avons pu distinguer deux grands modèles : les modèles géométriques (*Geometric Models*), appelés aussi modèles multidimensionnels (Nosofsky, 1992) et les modèles des attributs (*Feature Model)* appelés aussi « *Contrast Model* » (Tversky, 1977). Les modèles dimensionnels permettent de décrire des entités sous forme d'un ensemble limité de dimensions contenues. Une dimension est un axe dans un espace. Ainsi, une entité se définit par ses coordonnées dans l'espace des axes (*dimensions*) et reçoit une seule valeur sur chacune des dimensions qui le définissent. La proximité entre les objets dans cet espace reflète leur similarité.

D'autre part, (Tversky, 1977) a proposé un modèle de similarité qui repose sur la notion d'attribut (*The Contrast Model*). Il conteste les approches strictement dimensionnelles qui, selon lui, sont applicables aux seuls domaines que l'on peut décrire à l'aide d'un nombre restreint de dimensions. Dans les modèles des attributs, les concepts sont définis sous forme d'une liste d'attributs. La mesure de similarité utilisée se base sur un calcul impliquant les attributs communs et les attributs distinctifs présents dans les représentations des entités comparées. Dans le modèle de *Tversky*, la similarité est fondée sur une fonction linéaire des attributs communs et distinctifs définie par :

$$Sim\,(a,b) = \theta f(A \cap B) - \alpha f(A - B) - \beta f(B - A)$$

où a est le sujet de la comparaison et b est le référent. f est une fonction mesurant l'appariement de deux objets décrits sous la forme d'un ensemble d'attributs. A est l'ensemble d'attributs du concept a et B est l'ensemble d'attributs du concept b.

- $A \cap B$ représente les attributs communs, qui sont à la fois dans A et dans B
- $A - B$ représente les attributs distinctifs de A, c'est-à-dire qui sont en A mais pas en B
- $B - A$ représente les attributs distinctifs de B, c'est-à-dire qui sont en B mais pas en A
- $\theta, \alpha, et\ \beta \geq 0$ représentent des coefficients donnés aux attributs communs et distinctifs. Ils reflètent l'importance de chaque paramètre de la fonction.

La fonction de *Tversky* est une fonction asymétrique si $(\alpha \neq \beta)$, autrement dit, $Sim\,(a,b) \neq Sim\,(b,a)$. Selon *Tversky,* dans une tâche d'estimation de la similarité, on porte attention au sujet de la comparaison, soit l'objet a. Ainsi, on donne plus de poids aux

attributs de l'objet comparé (a) qu'aux attributs du référent (b), d'où $(\alpha > \beta)$. L'hypothèse de l'attention implique que la fonction de *Tversky* est une fonction directionnelle asymétrique.

Le calcul de similarité s'est avéré comme un moyen efficace pour améliorer les méthodes d'extraction de connaissance et celles d'intégration des sources de données hétérogènes. La mesure de similarité entre deux entités hétérogènes consiste en particulier à effectuer des comparaisons entre les composants de ces entités. Ces composants reflètent des hétérogénéités à différents niveaux : syntaxique, structurel et sémantique. Par conséquent, les différentes mesures de similarité définies dans la littérature ont été proposées pour pallier ces problèmes d'hétérogénéité. Une classification des différentes mesures de similarité utilisées dans le processus d'alignement est présentée dans (Rahm, et al., 2001) comme suit :

1) **La méthode terminologique** : Elle consiste à comparer des chaines de caractères appliquées aux noms, labels, et commentaires des entités en question. Elle est décomposée en une approche purement syntaxique et une autre lexicale. L'approche syntaxique effectue la correspondance à travers les mesures de similarité des chaînes de caractères (*e.g.*, EditDistance), tandis que l'approche lexicale effectue la correspondance à travers les relations lexicales (*e.g.*, synonymie, hyponymie, etc.).
2) **La méthode de comparaison des structures internes :** Elle compare les structures internes des entités telles que leurs attributs, ou les propriétés de type de données (*DatatypeProperty)* en parlant du langage OWL, (*e.g.*, cardinalité d'attributs, etc.).
3) **La méthode de comparaison des structures externes** : Elle compare les entités avec d'autres entités auxquelles elles sont liées. Si deux entités de deux ontologies sont similaires, leurs voisinages peuvent l'être également. Elle est décomposée en méthodes de comparaison des entités au sein de leurs taxonomies, c'est à dire en se basant sur la position des entités dans leurs hiérarchies (relations de subsomption), et en méthodes de comparaison des structures externes, telles que les propriétés associées, en l'occurrence les propriétés d'objet dans OWL (*ObjectProperty*). Par exemple, supposons qu'une relation R relie une classe A à une classe B dans une ontologie, et qu'une relation R' relie une classe A' à une classe B' dans une autre ontologie. Si on a une similarité entre les relations R et R' d'une part, et une similarité entre les classes B et B' d'autre part, alors on peut inférer qu'une similarité peut exister entre les deux classes A et A'.

4) **La méthode de comparaison des instances :** compare les extensions des entités, c'est-à-dire l'ensemble des autres entités qui lui sont attachées (généralement les instances des classes).
5) **La méthode sémantique** : compare les interprétations (ou plus exactement les modèles) des entités.

En se basant sur ces différents critères de comparaison d'entités, plusieurs approches d'alignement par mesure de similarité ont été proposées dans la littérature. Une étude bibliographique détaillée sur certaines approches est effectuée dans (Euzenat, et al., 2004). Nous présentons dans la suite quelques méthodes qui nous semblent intéressantes à étudier dans notre travail, et dont les langages de représentation sont appropriés au langage de nos ontologies, OWL DL.

(Maedche, et al., 2002) définissent une méthode de comparaison des concepts et des propriétés des ontologies à deux niveaux : syntaxique et sémantique. Cette méthode calcule la similarité entre deux taxonomies en effectuant une comparaison structurelle pour chaque classe, c'est à dire les labels de leurs hiérarchies (superclasses et sous-classes). Le calcul de similarité des propriétés est effectué en fonction des similarités des concepts définis dans leur domaine et co-domaine. La description d'un concept n'est pas considérée dans la fonction de calcul de cette méthode. Ceci ne représente pas d'inconvénients si les ontologies concernées sont limitées à la représentation des taxonomies de concepts primitifs. Néanmoins, cette méthode se révèle moins appropriée lorsqu'il s'agit d'une ontologie plus expressive, tel que le langage OWL DL, où des concepts plus complexes peuvent être définis en fonction de concepts primitifs et de restrictions sur des propriétés. Une expressivité plus riche d'un langage de représentation d'ontologie nécessite le développement des processus d'intégration plus complexes.

Les méthodes d'alignement OLA (Euzenat, et al., 2004-b) et EDOLA (Zghal, et al., 2007 - b) visent à aligner des ontologies représentées avec le langage OWL Lite. (Euzenat, et al., 2004-b) ont défini une méthode de calcul de similarité entre les entités d'ontologies OWL Lite en fonction de deux facteurs : la catégorie de l'entité (classe, instance, propriété, *etc.*) et l'ensemble de caractéristiques liées à cette catégorie (par exemple les superclasses, les propriétés, et les instances). Cette méthode présente l'avantage de la prise en considération des spécifications des ontologies en format OWL Lite, notamment les relations d'héritage entre des classes ou des propriétés, les restrictions des classes, et les caractéristiques des

propriétés. Ainsi, elle considère la plupart des caractéristiques du langage OWL Lite dans le processus de calcul de similarité. Elle permet de traiter la structure interne d'une classe définie en termes de propriétés et de contraintes, aussi bien que la structure externe définie en termes de relations sémantiques avec d'autres classes et de relations de subsomption. Cependant, cette méthode ne considère pas tous les constructeurs de OWL Lite, tels que la disjonction des instances ou des classes. En outre, des tests supplémentaires sont nécessaires pour définir les poids des calculs de similarité.

(Zghal, et al., 2007) définissent une méthode d'alignement d'ontologies en OWL DL, appelée SODA (*Structural Ontology OWL-DL Alignment*), basée sur le calcul de similarité. Cette méthode définit deux modèles de calcul de similarité : locale et globale. Elle combine les mesures de similarité locale (terminologique et structurelle) pour l'évaluation de la similarité globale. Elle permet de générer un alignement exploitant l'aspect structurel du voisinage des entités à apparier.

Sim-DL (Janowicz, et al., 2007) est une méthode de mesure de similarité sémantique des ontologies représentées avec OWL DL. Elle est appliquée à l'extraction de l'information dans le domaine de la géographie. Cette méthode est adaptée à la variante $ALCNR$ de la logique de description. Elle est implémentée avec l'éditeur des ontologies *Protégé*. Le scénario de cette approche repose sur les cinq étapes suivantes :

1) Sélection des entités à apparier
2) Transformation des concepts en une forme normale, telle que la forme normale de disjonction (*DNF : Disjunctive Normal Form*)
3) Définition d'une matrice d'alignement entre les descriptions des concepts. Elle s'applique à tous les composants de descriptions entre deux concepts.
4) Application des fonctions locales de similarité spécifiques aux constructeurs OWL DL pour chaque paire de concepts.
5) Définition d'une fonction globale de similarité. Elle représente une fonction d'agrégation normalisée en fonction des similarités locales, et pondérée avec des poids attribués à chaque fonction.

Ce scénario forme un squelette générique pour différentes méthodes de calcul de similarité (d'Amato, et al., 2006), (Janowicz, 2006). Certaines étapes de ce scénario seront utilisées dans le développement de notre méthode de calcul de similarité, présentée dans la section suivante (*cf.* section 6.3). L'avantage de cette approche réside dans la généralité du

scénario suivi, ainsi que dans la prise en compte des propriétés dans une ontologie pour la définition des fonctions de similarité. Cependant, l'inconvénient de cette approche réside dans le fait que les fonctions de similarité s'appliquent uniquement aux descriptions des entités à apparier, où les similarités des concepts avoisinants (super et sous classes et les concepts liés avec des propriétés d'objets) n'ont pas été considérées.

(Patil, 2005) définit une méthode de calcul de similarité entre les ontologies en OWL DL pour l'échange de la sémantique des données d'un modèle de produit. Cette méthode définit une fonction globale d'agrégation en fonction de similarités locales calculées pour la description d'un concept et pour le contexte dans lequel il a été défini. Cette méthode procède par la conversion des descriptions des concepts en une forme normale, en l'occurrence la forme normale de disjonction DNF. Les fonctions locales de similarité sont basées sur les modèles des attributs de *Tversky*, où une fonction linéaire et asymétrique de comparaison des attributs de concepts a été utilisée. L'inconvénient de cette approche, de notre point de vue, est que les fonctions de similarité définies ne prennent pas en considération la similarité entre les attributs à comparer. Autrement dit, la classification des attributs des concepts à comparer se base strictement sur des équivalences exactes : les attributs équivalents sont ajoutés à l'ensemble des attributs communs, sinon à l'ensemble des attributs distinctifs. De plus, aucune similarité entre des propriétés n'a été définie dans cette méthode.

Après l'étude des différentes approches de mesure de similarité dans cette section, nous allons définir dans la section suivante une méthode de calcul de similarité sémantique appropriée à notre prototype d'échange des modèles de produit. En effet, le langage de représentation dans un domaine de connaissance joue un rôle primordial dans la définition des fonctions de calcul de similarité. La plupart de ces méthodes d'alignement considèrent seulement des sous-ensembles de définitions dans une ontologie. Notre objectif est de définir une méthode de calcul de similarité sémantique qui prend en considération l'ensemble des connaissances représentées avec les ontologies OWL DL, tel que la description d'une entité et son contexte. De plus, notre méthode s'applique non seulement aux concepts mais aussi aux propriétés définies dans une ontologie. Nous nous intéressons donc dans la suite à définir une méthode correspondant à l'expressivité du langage OWL DL, basé sur la logique descriptive et plus précisément sur sa variante $SHOIN(D)$.

6.3 Définition de notre méthode

Dans notre méthode d'intégration présentée dans le chapitre précédent, la conversion d'un modèle CAO de produit se base sur la détection des entités correspondantes, comme les concepts ou les propriétés dans l'ontologie cible. Dans le cas où les types des instances n'ont pas de correspondants, nous procédons à la recherche des similarités des entités concernées. L'algorithme de recherche de similarité est donc appelé lorsque le moteur d'inférence n'a pas pu déduire d'entités équivalentes. Ainsi, la recherche de similarité constitue une extension de notre méthode d'intégration visant à trouver pour chaque instance le type correspondant, c'est-à-dire celui ayant une similarité maximale. Notre méthode de mesure de similarité est conçue de façon qui permet d'exploiter toutes les connaissances représentées dans l'ontologie OWL DL, décrivant les classes, les propriétés, et les instances.

Avec les langages de représentation de connaissance, les entités sont décrites en fonction de l'expressivité du langage. Elles peuvent être définies comme étant des ensembles de caractéristiques, des dimensions dans un espace multidimensionnel, ou des spécifications de restrictions formelles tout en utilisant des opérateurs logiques. Les ontologies utilisées dans notre approche d'échange sont encodées avec le langage OWL DL. Ainsi nous nous limitons dans notre travail aux descriptions fournies par les constructeurs du langage OWL.

OWL DL se base sur la variante expressive $SHOIN(D)$ permettant l'intersection, l'union, la quantification universelle complète, la restriction sur des propriétés, la négation complète (négation des concepts définis), et les restrictions de cardinalités. Ceci permet la description des concepts complexes basés sur des concepts primitifs et des propriétés (prédicats binaires). Le tableau 3, présenté dans le chapitre précédent (*cf.* section 5.4.1) illustre l'expressivité syntaxique et sémantique de la variante $SHOIN(D)$.

Nous nous intéressons dans notre méthode aux aspects structurels et sémantiques liés aux entités plutôt qu'aux aspects terminologiques. Nous écartons donc de notre méthode la définition de fonctions de similarité basées sur des critères syntaxiques liés aux entités. Par exemple, la comparaison des noms des concepts n'est pas considérée par dans notre méthode de mesure de similarité. Néanmoins, l'intégration de cette comparaison terminologique peut être envisagée si nécessaire, et des algorithmes de comparaisons syntaxiques et lexicales

existant dans la littérature peuvent être facilement ajoutés dans notre méthode (Monge, et al., 1996), (Euzenat, et al., 2004), (Fellbaum, 1998).

D'ailleurs, les méthodes sémantiques de recherche de similarité ne sont pas très autonomes et nécessitent souvent une phase préalable de traitement « *preprocessing* » (Euzenat, et al., 2004). Cette phase consiste principalement à définir des valeurs de similarité entre des entités, servant de valeurs d'« *ancrage* » avant l'appel des fonctions de mesure de similarité des autres entités. Ces valeurs peuvent être le résultat de calcul des méthodes syntaxiques ou lexicales, ou prédéfinies par l'utilisateur par des axiomes d'équivalence ou de disjonction. Nous représentons les valeurs de similarité entre des entités de deux ontologies dans un vecteur de triplets V_{sim}. Chaque triplet est défini par :

$$< E_1, E_2, val > tel\ que\ \ E_1 \in O_1,\ E_2 \in O_2, val\ \in \mathbb{R}\ \ avec\ 0 \leq val \leq 1$$

où E_1 et E_2 sont deux entités appartenant respectivement aux ontologies O_1 et O_2, val désigne la valeur de similarité assignée à la paire d'entités. Notons qu'une équivalence définie ou inférée entre deux entités est représentée dans un triplet dont la valeur de similarité est maximale, c'est-à-dire val = 1.

Considérons les deux ontologies O_1 et O_2. Pour une entité $E \in O_1$et de catégorie X (classe ou propriété), la correspondance par similarité sémantique revient à trouver l'entité $F \in O_2$ de la même catégorie X, tel que :

$$Sim\ (E, F) = \max\big(Sim(E, F_i)\big), \forall\ F_i \in\ O_2$$

où F_i est la $i^{ème}$ entité de catégorie X dans l'ontologie O_2, et $1 \leq i \leq n$, avec n le nombre d'entités de catégorie X dans O_2. Une fois la similarité entre E et F est calculée, le triplet représentant cette similarité sera ajouté dans le vecteur de similarité V_{sim}. Les catégories des entités représentent les types d'entités à apparier, tels qu'une classe, une propriété d'objet, une propriété de type de données, *etc*. Ces différentes catégories et leurs fonctions de similarité seront détaillées dans la section suivante (*cf.* section 6.4).

En effet, notre méthode exploite la structure des ontologies pour calculer les similarités entre deux entités. Elle associe à chaque catégorie d'entités une fonction d'agrégation, qui prend en considération non seulement la description des entités à apparier, mais aussi leurs entités avoisinantes liées par des relations de subsomption ou des propriétés

d'objets. Elle se base effectivement sur un modèle de calcul de similarité à deux niveaux : local et global. Ces deux niveaux sont définis comme suit :

1) Fonction globale de similarité relative à chaque catégorie : Elle représente la valeur finale de similarité entre une paire d'entités. Elle est décrite par une fonction d'agrégation pondérée des valeurs des similarités locales calculées pour chaque catégorie d'entités. Ce niveau de calcul de similarité sera détaillé dans la section 6.4.
2) Fonction locale de similarité : la fonction locale de similarité entre deux entités de deux ontologies est calculée entre les composants correspondant aux entités en question. Ces composants sont des parties de connaissances contenues dans les définitions de l'entité par les primitives du langage OWL. Cette fonction sera détaillée dans la section 6.5.

6.4 Définition de la fonction globale

La fonction globale de mesure de similarité entre deux entités désigne la valeur finale calculée en fonction de connaissances liées à ces deux entités. Ceci nécessite la prise en compte de l'expressivité du langage OWL DL et de l'ensemble de ses caractéristiques entrant dans la description des entités. Ainsi, la fonction globale de similarité est définie en fonction de méthodes basées sur les structures interne et externe des entités à apparier.

Les méthodes basées sur la structure interne des entités utilisent des descriptions telles que les co-domaines des propriétés (propriété d'objets ou de types de données), les cardinalités, et certaines caractéristiques des propriétés (transitivité, symétrie, *etc.*) (Euzenat, et al., 2004). Concernant la structure externe, la mesure de similarité entre deux entités repose principalement sur la position des entités dans leur hiérarchie. Autrement dit, si deux entités de deux ontologies sont similaires, leurs voisins peuvent avoir une similarité entre eux.

La définition de notre fonction globale est basée sur celle définie dans la méthode OLA (Euzenat, et al., 2004-b) pour le langage OWL Lite. Par conséquent, nous adaptons cette fonction dans notre méthode pour prendre en considération l'extension de la version DL par rapport à la version Lite du langage OWL. En effet, la mesure de similarité globale dépend de deux facteurs :

a) La catégorie des entités à apparier, telle que définie dans OWL DL : une classe ou concept (C), une propriété d'objet (R), une propriété de type de données (P), un

type de données (D), une instance de classe (IC), une instance de propriété (IR), ou une valeur de données (V).

b) L'ensemble de caractéristiques associées à chaque catégorie d'entités, par exemple sa description, sa hiérarchie, ses propriétés, etc.

Comme présentées dans OLA, ces catégories peuvent être liées entre elles avec des relations, notées $F(x) = \{x;\ \exists y;\ < x, y > \in F\}$. Ces liaisons sont classées sous différentes catégories :

1) *Spécialisation* (S) : (ou subsomption) entre deux classes ou deux propriétés. Dans une hiérarchie, une sous-entité est une « *spécialisation* » d'une super-entité, et réciproquement la super-entité est une « *généralisation* » de la sous-entité.
2) *Attribution* (A) : désigne une relation d'attribution entre une classe et une propriété, ou également entre une instance de classe et une instance de propriété. Elle peut être spécifiée en deux attributions : attribution (A_r) s'il s'agit d'une propriété d'objet, ou attribution (A_p) si la propriété est de type valeur de données.
3) *Instanciation* (I) : (ou type) désigne une relation d'appartenance entre une instance de classe et une classe, entre une instance de propriété et une propriété, ou entre une valeur de données et un type de données.
4) *Equivalence (E)* : peut être définie entre deux classes ou deux propriétés.
5) *Exclusion (X)* : désigne une relation de disjonction entre deux classes. Autrement dit, elle sert à dénoter que l'intersection des deux classes doit être un ensemble vide.
6) Domaine *(Do) :* spécifie les relations entre une propriété et les concepts définis comme domaine de la propriété.
7) Co-domaine *(Co) :* spécifie les relations entre une propriété et les concepts définis comme co-domaine de la propriété.
8) Propriété Inverse *(InvP) :* définit une relation de propriété inverse entre deux propriétés de type d'objets. Par exemple, dans une liste définie la propriété « *hasPrevious* » est l'inverse de la propriété « *hasNext* » ;
9) Des fonctions sont également définies pour déterminer certaines caractéristiques des propriétés, telles que la transitivité ou la symétrie d'une propriété. Sachant que leur valeur de retour est 0 ou 1, ces fonctions sont définies comme suit :
    a. $Func(R)$, $Func(P)$: déterminent respectivement si une propriété d'objet ou une propriété de type de données est fonctionnelle ou non.

b. $Tr\,(R)$, $Sym\,(R)$, $Inv(R)$: déterminent respectivement si une propriété d'objets R est une propriété transitive, symétrique, ou inverse fonctionnelle.

Comme définie dans (Euzenat, et al., 2004-b), chaque catégorie d'entités est distinguée par une liste de caractéristiques et de relations qui lui sont spécifiques. En se basant sur cette distinction, la définition de la fonction globale de similarité doit être donc relative à chaque catégorie, en fonction de ses caractéristiques et de ses liaisons. Ainsi, la fonction globale de calcul de similarité $Sim_X\,(C, D)$, associée à une catégorie *X* entre deux entités *C* et *D*, dépend proportionnellement des valeurs de similarité des paires d'entités (C_i, D_i) , tel que C_i et D_j sont des entités liées à *C* et *D* respectivement via des relations (x). Par exemple C_i peut être une sous-classe de *C* définie avec une relation de spécialisation entre *C* et C_i.

Le couple (C, D) à apparier est appelé le couple d'« ancrage » de la similarité, et les couples (C_i, D_i) liés respectivement au couple (C, D) sont des couples contributeurs, puisqu'ils affectent directement la valeur de similarité du couple d'ancrage (Euzenat, et al., 2004-b). Un couple contributeur dans une fonction peut être un couple d'ancrage dans une autre fonction. Cette relation de contribution peut être propagée transitivement à travers les couples d'entités évalués selon des ordres $n\ tel\ que\ n > 0$. Un paramètre de profondeur n sera défini dans notre algorithme pour déterminer l'ordre de propagation désiré pour le calcul de similarité entre un couple d'entités. Par défaut, les entités directes sont uniquement concernées par le calcul de similarité, c'est-à-dire $n = 1$.

Nous avons utilisé l'équation de la fonction globale de similarité définie dans OLA (Euzenat, et al., 2004-b). Cette fonction globale est obtenue par l'agrégation des valeurs de similarité des couples contributeurs, par une somme pondérée de ces valeurs, permettant de contrôler la contribution de chaque élément. Concrètement, la fonction globale de similarité, $\mathcal{Sim}_X$: $X^2 \rightarrow [0,1]$, relative à une catégorie *X* est définie comme suit (Euzenat, et al., 2004-b) :

$$Sim_X(x, x') = \sum_{F \in N(x)} \omega_F^X MSim_Y(F(x), F(x'))$$

où $N(X)$ représente l'ensemble de liens définis pour chaque catégorie *X*. Une attention particulière est affectée à chaque type de liaison des entités, $F(x)$. Pour considérer l'importance de cette attention, un paramètre de pondération, ω_F^X , est défini reflétant le poids attentionnel porté par les sujets à la liaison définie, où $\omega_F^X \geq 0$, et $\sum \omega_F^X = 1$.

Le tableau 4 présente les caractéristiques de chaque catégorie que nous considérons dans la définition de la fonction globale de similarité. En effet, nous avons défini ces caractéristiques, appropriées au langage OWL DL, pour deux catégories : concepts et propriétés. Dans ce tableau, les concepts sont représentés par la catégorie classe, et les propriétés par les deux catégories propriétés d'objet et propriété de type de données.

Catégorie (X)	**Relation F(X)**	**Conditions**	**Fonction de Similarité $\mathbf{Sim_Y(F(x), F(x'))}$**
Classe $Sim_C(C, D)$	Equivalent	$(C, C') \in E$ et $(D, D') \in E$	$Sim_C(C', D')$
	SubClassOf	$(C, C') \in S$ et $(D, D') \in S$	$Sim_C(C', D')$
	DisjointClass	$(C, C') \in X$ et $(D, D') \in X$	$Sim_C(C', D')$
	ObjectProperty	$R_1 \in R, (C, R_1) \in A_r$ et $R_1' \in R, (D, R_1') \in A_r$	$MSim_R(R_1, R_1')$
	DatatypeProperty	$P_1 \in P, (C, P_1) \in A_p$ et $P_1' \in P, (D, P_1') \in A_p$	$MSim_P(P_1, P_1')$
Propriété d'objets $Sim_R(R_1, R_1')$	EquivalentPoperty	$(R_1, R_2) \in E$ et $(R_1', R_2') \in E$	$MSim_R(R_2, R_2')$
	SubPropertyOf	$(R_1, R_2) \in S$ et $(R_1', R_2') \in S$	$MSim_R(R_2, R_2')$
	Domain	$(R_1, C_1) \in Do$ et $(R_1', C_1') \in Do$	$Sim_C(C_1, C_1')$
	Range	$(R_1, C_1) \in Co$ et $(R_1', C_1') \in Co$	$Sim_C(C_1, C_1')$
	InverseProperty	$(R_1, R_2) \in InvP$ et $(R_1', R_2') \in InvP$	$MSim_R(R_2, R_2')$
	Functional	$Func(R_1)$ $Func(R_1')$	1 si $Func(R_1) = Func(R_1')$; 0 sinon
	InverseFunctional	$Inv(R_1)$ $Inv(R_1')$	1 si $Inv(R_1) = Inv(R_1')$; 0 sinon
	Symmetry	$Sym(R_1)$ $Sym(R_1')$	1 si $Sym(R_1) = Sym(R_1')$; 0 sinon
	Transitivity	$Tr(R_1)$ $Tr(R_1')$	1 si $Tr(R_1) = Tr(R_1')$; 0 sinon
Propriété de type de données $Sim_P(P_1, P_1')$	EquivalentPoperty	$(P_1, P_2) \in E$ et $(P_1', P_2') \in E$	$MSim_P(P_2, P_2')$
	SubPropertyOf	$(P_1, P_2) \in S$ et $(P_1', P_2') \in S$	$MSim_P(P_2, P_2')$
	Domain	$(P_1, C_1) \in Do$ et $(P_1', C_1') \in Do$	$Sim_C(C_1, C_1')$
	Range	$(P_1, C_1) \in Co$ et $(P_1', C_1') \in Co$	$Sim_D(C_1, C_1')$
	Functional	$Func(P_1)$ $Func(P_1')$	1 si $Func(P_1) = Func(P_1')$; 0 sinon

Tableau 4. Similarité des catégories en fonction de leurs composants

Dans notre méthode, nous nous intéressons à la comparaison des entités au niveau terminologique $TBox$, plutôt qu'au niveau factuel $ABox$. En effet, pour transférer les instances d'une ontologie à une autre, il suffit de détecter les types des instances, soit les concepts et les propriétés, correspondants dans l'ontologie cible. Comme présenté dans le tableau 4, la fonction globale de mesure de similarité des concepts *C* (catégorie Classe) est définie en fonction des caractéristiques liées à cette catégorie comme suit :

- *Equivalent* : établit une liaison d'équivalence entre *C* et un ensemble de classes équivalentes à *C*. Les relations d'équivalence peuvent être définies en termes d'équivalences explicites avec d'autres classes, ou de classes complexes anonymes représentées par des restrictions étant des conditions nécessaires et suffisantes. L'ensemble de ces classes anonymes et explicites sont groupées avec l'opérateur d'intersection pour former la description de la classe équivalente (C'). Par exemple si un concept C est défini tel que : $C \equiv (A \cup B)$ et ayant la restriction $\forall R.D$ définie comme étant une condition nécessaire et suffisante de C, alors la description de son équivalence représentée par C' est définie par $C' \equiv (A \cup B) \cap \forall R.D$
- *SubClassOf* : définit une liaison de subsomption entre C avec l'ensemble de ses super-classes. Les relations de subsomption peuvent être définies explicitement, ou moyennant des classes complexes anonymes représentées par des restrictions définies sous des conditions nécessaires.
- *DisjointClass* : définit une liaison de disjonction entre C et un ensemble de concepts.
- *ObjectProperty* : définit l'ensemble de propriétés d'objet ayant C comme domaine de propriété. Ainsi, la similarité entre deux propriétés d'objet peut être exploitée pour affecter la similarité des concepts définis comme domaines ou co-domaines des propriétés. Par exemple, si une relation R relie une classe A à une classe B dans une ontologie, et une relation R' relie une classe A' à une classe B' dans une autre ontologie, si R est similaire à R' d'une part, et si B et B' sont similaires, alors on peut inférer qu'une similarité peut exister entre les deux classes A et A'.
- *DatatypeProperty* : définit l'ensemble de propriétés de type de données ayant C comme domaine de propriété.

La combinaison de toutes ces connaissances construit la définition de l'entité. Ainsi, la fonction globale de similarité entre deux concepts est définie par :

$$Sim_C(C,D) = \omega_E^C Sim\left(E(C),E(D)\right) + \omega_S^C Sim\left(S(C),S(D)\right) + \omega_{A_R}^R MSim\left(A_R(C),A_R(D)\right) + \omega_{A_p}^P MSim\left(A_P(C),A_P(D)\right)$$

Où $\omega_F^X \geq 0\ et\ \sum \omega_F^X = 1$

Cette équation indique que la fonction globale de mesure de similarité entre les deux entités C et D, de la catégorie classe, est calculée en fonction de similarité de leurs entités équivalentes $((E(C),E(D))$, de leurs entités avoisinantes $\left(S(C),S(D)\right)$, de leurs propriétés d'objet $\left(A_R(C),A_R(D)\right)$, et de leurs propriétés de types de données $\left(A_P(C),A_P(D)\right)$. Pour les similarités de propriétés, $MSim$ indique que le calcul s'effectue entre deux ensembles d'éléments. Nous avons donc utilisé la fonction de mesure de similarité, « *Match-Based Similarity* », définie par (Touzani, 2005) :

$$MSim(E,E') = \frac{\sum_{(i,i')\in Paires(E,E')} Sim\ (i,i')}{Max(|E|,|E'|)}$$

où (E) et (E') représentent deux ensembles d'éléments de même catégorie. Cette fonction requiert que les similarités entre les paires d'éléments (i,i') de (E,E') soient calculées. Ensuite, une phase de sélection des paires est effectuée permettant de choisir pour chaque élément i de (E) l'élément i' de (E') ayant le maximum de similarité.

Considérons l'exemple de calcul de similarité entre les deux concepts $(Revolve)$ et $(Hole)$ représentant deux features de conception. La fonction globale de mesure de similarité entre les deux concepts est donnée par :

$$Sim_C(Revolve,Hole) = \omega_E^C Sim\left(E(Revlove),E(Hole)\right) + \omega_S^C Sim\left(S(Revolve),S(Hole)\right) + \omega_{A_R}^R MSim\left(A_R(Revolve),A_R(Hole)\right) + \omega_{A_p}^P MSim\left(A_P(Revolve),A_P(Hole)\right)$$

Nous visons dans cet exemple à mettre en relief le calcul de similarité entre deux ensembles d'éléments, notamment les ensembles de propriétés des deux concepts. Ainsi, nous supposons que :

- Les valeurs de similarité des concepts équivalents et avoisinants soient nulles, c'est-à-dire $Sim\left(E(Revlove), E(Hole)\right) = 0$ et $Sim\left(S(Revolve), S(Hole)\right) = 0$.
- Les poids soient équitablement répartis entre les différents composants, c'est-à-dire $\omega_E^C = \omega_S^C = \omega_{A_R}^R = \omega_{A_p}^P = 0.25$.
- L'ensemble de propriétés de chaque concept, $(Revolve)$ et $(Hole)$, soit donné respectivement par :

$$(E) = A\ (Revolve) = \{hasDiameter, hasSketch\}$$

$$(E') = A(Hole) = \{hasRadius, hasProfile, hasName\}$$

Rappelons que A désigne la fonction d'attribution d'un concept à une propriété, composée de (A_r) pour les propriétés d'objets et (A_p) pour les propriétés de type de données.

- Les calculs de similarité entre les paires d'éléments de (E) et (E') aient donné les résultats suivants :

$Sim(hasDiameter, hasRadius) = 0.72$; $Sim(hasDiameter, hasProfile) = 0.20$

$Sim(hasDiameter, hasName) = 0.11$; $Sim(hasSketch, hasRadius) = 0.18$

$Sim(hasSketch, hasProfile) = 0.67$; $Sim(hasSketch, hasName) = 0.09$

Alors les entités sélectionnées, représentant les meilleures similarités pour chaque élément, seront les suivantes :

$$Paires(E, E') = \{(hasDiameter, hasRadius), (hasSketch, hasProfile)\}$$

Et

$$MSim(E, E') = \frac{0.72 + 0.67}{3} = 0.46$$

$$Sim_C(Revolve, Hole) = \omega_A^R MSim\left(A(Revolve), A(Hole)\right) = 0.5 \times 0.46 = 0.23$$

Notons que les fonctions de similarité des caractéristiques pourraient amener à des récursivités lors des appels des fonctions de similarité. Pour éviter ce problème, nous définissons dans notre algorithme un paramètre permettant de définir la profondeur souhaitée lors du processus de calcul de similarité entre deux entités. Par exemple, les couples contributeurs directs d'un couple d'ancrage sont définis avec une profondeur 0, les contributeurs des contributeurs sont définis au niveau de profondeur 1, et ainsi de suite. De plus, nous définissons un système de blocage de calcul de similarité par la création d'un vecteur d'entités incluses dans le processus de calcul. Ainsi, le traitement des entités déjà existantes dans ce vecteur seront écartées par une affectation de valeur 0 à leur fonction de similarité. Pour calculer les similarités des descriptions d'entités, nous allons définir dans la section suivante un algorithme de calcul de similarité locale entre deux descriptions.

6.5 Similarité locale

Dans notre méthode, la mesure de similarité locale revient à calculer la similarité entre deux descriptions définissant des classes anonymes complexes. Le scénario suivi dans notre méthode est celui utilisé dans la plupart des méthodes de mesure de similarité entre deux descriptions, comme dans (Patil, 2005), (Janowicz, et al., 2007). Ce scénario est particulièrement basé sur les étapes suivantes :

1) *Normalisation* : Cette étape consiste à transformer la description en une forme normale, telle que la forme normale de conjonction (*CNF : Conjunctive Normal Form*) ou de disjonction (*DNF : Disjunctive Normal Form*).
2) *Application Matricielle* : Cette étape consiste à appliquer une matrice d'alignement entre les différentes parties de normalisation résultant de la première étape.
3) *Fonction spécifique aux constructeurs logiques* : Cette étape consiste à comparer les différents éléments de la matrice précédente et à définir une fonction locale de calcul de similarité spécifique à chaque type de constructeur, $(\cap, \cup, \forall, \exists, \leq, \geq)$, trouvé dans les éléments à comparer.

Ces étapes sont détaillées dans les sous-sections suivantes.

6.5.1 Normalisation des descriptions

Avant de procéder à la comparaison des descriptions de deux entités *C* et *D*, ces descriptions doivent être normalisées afin de réduire l'influence syntaxique. La normalisation consiste à transformer la description vers une forme canonique, notamment la forme normale de disjonction (DNF). La procédure de normalisation se réalise par l'application de certaines règles de conversion. Elle dépend particulièrement du langage de représentation utilisé, et sa complexité dépend de l'expressivité du langage.

Pour les langages basés sur les logiques de description, un algorithme de subsomption, présenté dans (Baader, et al., 2003), permet de convertir la description d'un concept en une forme normale de disjonction DNF. Pratiquement, une implémentation a été réalisée dans le cadre du projet SIM-DL pour le calcul de similarité dans le domaine de la géographie (Janowicz, et al., 2007). Les développeurs de cette application nous ont donné accès à l'API de cette implémentation et étaient collaboratifs pour l'envoi des sources nécessaires pour la normalisation des descriptions OWL DL.

La procédure de normalisation est la suivante : une description de concept est d'abord convertie en sa forme normale négative (FNN), *i.e.* les seules négations dans une description sont de la forme $(\neg A)$, où A désigne un concept primitif. Les expressions de concepts peuvent être converties en FNN en appliquant les règles de normalisation suivantes, y compris les *règles de Morgan* :

$$\neg \top \rightarrow \bot$$
$$\neg \bot \rightarrow \top$$
$$\neg \neg C \rightarrow C$$
$$\neg(C \cup D) \rightarrow \neg C \cap \neg D$$
$$\neg(C \cap D) \rightarrow \neg C \cup \neg D$$
$$\neg(\forall R.C) \rightarrow (\exists R.\neg C)$$
$$\neg(\exists R.C) \rightarrow (\forall R.\neg C)$$
$$\neg(\leq n\ R) \rightarrow \geq (n+1)\ R$$
$$\neg(\geq n\ R) \rightarrow \leq (n-1)\ R\ si\ n > 1$$
$$\neg(\geq n\ R) \rightarrow \forall R.\bot\ si\ n = 1$$

où C et D désignent deux concepts complexes, R une relation, et n un entier.

Ensuite, le résultat obtenu est converti en sa forme normale de disjonction (DNF) en utilisant les règles de distribution suivantes :

$$A \cap (C \cup D) \leftrightarrow (A \cap C) \cup (A \cap D)$$

$$(C \cup D) \cap A \leftrightarrow (C \cap A) \cup (D \cap A)$$

En logique de description, l'application des règles ci-dessus mène à la normalisation des expressions des concepts en DNF. Ainsi, une description d'une entité C est normalisée en DNF si et seulement si elle est représentée sous une des formes suivantes :

$$C \equiv \top, C \equiv \bot, ou\ C \equiv C_1 \cup C_2 \cup C_3 \cup \ldots \cup C_n$$

où chaque $C_i\ (i{:}\ 1 \ldots n)$ est une disjonction de C formée par une conjonction d'éléments décrite sous la forme suivante :

$$C_i \equiv E_1 \cap E_2 \cap E_3 \cap E_4 \cap E_5 \cap E_6$$

où $E_i, 1 \leq i \leq 6$, est un élément de conjonction qui correspond à l'un des six composants suivants :

$$E_1 \equiv (\cap_0^I\ A_i)$$

$$E_2 \equiv \left(\cap_0^J\ R.D_j\right)$$

$$E_3 \equiv (\cap_0^K\ \forall R.D_k)$$

$$E_4 \equiv (\cap_0^L\ \exists R.D_l)$$

$$E_5 \equiv (\cap_0^M \geq n\ R.D_m)$$

$$E_6 \equiv (\cap_0^N \leq p\ R.D_n)$$

E_1 désigne l'ensemble d'intersection des primitives A_i, ou de leur négation. R représente une propriété binaire. E_3 et E_4 représentent respectivement des restrictions définies par des quantificateurs universels et existentiels. E_5 et E_6 représentent des restrictions sur des cardinalités minimales et maximales.

D'ailleurs, pour réduire davantage l'impact de l'aspect syntaxique de représentation sur la mesure de similarité, des règles de réécriture, présentées dans (Brandt, et al., 2002), peuvent être appliquées. Ces règles de réécriture permettent de mettre en correspondance des expressions équivalentes, comme par exemple $(\forall\, R.\bot)$ et $(\leq 0\ R.\top)$.

Une fois les descriptions sont normalisées, nous procédons au calcul de similarité des différentes parties de disjonction, formant une représentation matricielle détaillée dans la section suivante.

6.5.2 Définition des matrices d'alignement

Le processus de normalisation effectué dans la phase précédente engendre un ensemble de disjonction pour chaque description. Ainsi, les descriptions de C et D, notées C_{des} et D_{des}, sont respectivement représentées comme suit :

$$C_{des} \equiv\ C_1 \cup\ C_2\ \cup \ldots \cup C_m$$

$$D_{des} \equiv\ D_1 \cup\ D_2\ \cup \ldots \cup D_n$$

où chaque disjonction de C_{des} et D_{des} est représentée respectivement par C_{dis} et D_{dis} .

La plupart des méthodes de mesure de similarité supposent que la fonction de calcul de similarité est une fonction binaire, s'appliquant à une paire de concepts. Ainsi, une matrice d'alignement des paires entre les différentes disjonctions des concepts est établie. Pour la comparaison de C_{des} et D_{des}, une matrice d'alignement consiste à créer des tuples $Sim\left(C_i, D_j\right)$ pour toutes les combinaisons possibles du produit cartésien $C_{des} \times D_{des}$ (Markman, 2001).

$$M_{Des} = \begin{bmatrix} C_1, D_1 & C_1, D_2 & \cdots & C_1, D_{n-1} & C_1, D_n \\ C_2, D_1 & C_2, D_2 & \cdots & C_2, D_{n-1} & C_2, D_n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ C_{m-1}, D_1 & C_{m-1}, D_2 & \cdots & C_{m-1}, D_{n-1} & C_{m-1}, D_n \\ C_m, D_1 & C_m, D_2 & \cdots & C_m, D_{n-1} & C_m, D_n \end{bmatrix}$$

Matrice de paires à comparer.

Chaque élément de la matrice $\left(C_i, D_j\right)$ représente une paire de disjonction, C_{dis} et D_{dis}. Comme nous l'avons indiqué dans la section précédente (*cf.* section 6.5.1), chaque

disjonction est formée d'un ensemble de conjonction d'éléments E_i. Le calcul de similarité entre deux éléments (E_i, E_i') peut être effectué seulement si les éléments ont le même type de constructeur. Par exemple, une restriction de cardinalité,($\leq$), ne peut pas être comparée à une restriction de quantification universelle $\forall$. Ainsi, nous présentons dans la section suivante (*cf.* section 6.5.3) une fonction de similarité relative au type de constructeur sera décrite dans la section suivante

Après le calcul de similarité de toutes les paires $Sim\left(C_i, D_j\right)$, nous choisissons pour chaque C_i la disjonction D_j dont la valeur de similarité est la plus élevée. Ces paires sont sélectionnées pour déterminer ultérieurement la valeur de similarité entre C_{des} et D_{des}. Ainsi, l'ensemble de paires sélectionnées est représenté comme suit :

$$S = \{(C_1, D_l), (C_2, D_m), \ldots, (C_n, D_z)\}$$

Notons que chaque disjonction C_i et D_j ne doit être présente qu'une seule fois dans l'ensemble S. La fonction de similarité entre les descriptions C_{des} et D_{des}, définie comme une valeur moyenne de l'ensemble de similarité des paires de disjonction choisies, S, est donnée par la fonction suivante :

$$Sim_{des}(C, D) = \frac{\sum_{(C_i, D_j) \in S} Sim_{dis}(C_i, D_j)}{Max(m, n)}$$

où $Sim_{des}(C, D)$ représente la valeur de similarité au niveau de description entre la paire (C, D), et $Sim_{dis}(C_i, D_j)$ désigne la valeur de similarité entre les paires sélectionnées C_i, D_j, dont la fonction de calcul sera présentée dans la section suivante. m et n représentent respectivement le nombre de disjonction des descriptions de C et D.

Supposons par exemple que les descriptions de C et D soient définies par :

$$C_{des} \equiv C_1 \cup C_2 \cup C_3$$

$$D_{des} \equiv D_1 \cup D_2 \cup D_3 \cup D_4$$

Supposons que le calcul de similarité $Sim_{dis}(C_i, D_j)$ entre les paires (C_i, D_j) soit représenté dans la matrice suivante :

$$M_{Des} = \begin{bmatrix} C_1, D_1 = 0.64 & C_1, D_2 = 0.52 & C_1, D_3 = 0.31 & C_1, D_4 = 0.12 \\ C_2, D_1 = 0.49 & C_2, D_2 = 0.76 & C_2, D_3 = 0.67 & C_2, D_4 = 0.28 \\ C_3, D_1 = 0.15 & C_3, D_2 = 0.36 & C_3, D_3 = 0.64 & C_3, D_4 = 0.31 \end{bmatrix}$$

Alors, pour chaque disjonction C_i , nous sélectionnons la disjonction D_j ayant la plus grande valeur de similarité. Ceci revient à sélectionner pour chaque ligne de matrice la valeur la plus élevée. Ainsi, l'ensemble des paires sélectionnées, S, est représenté par :

$$S = \{(C_1, D_1), (C_2, D_2), (C_3, D_3)\}$$

Ensuite, la valeur de similarité est calculée par la fonction suivante :

$$Sim_{des}(C, D) = \frac{\sum_{(C_i, D_j) \in S} Sim_{dis}(C_i, D_j)}{Max(m, n)} = \frac{0.64 + 0.76 + 0.6}{\max(3,4)} = \frac{2}{4} = 0.5$$

6.5.3 Définition de fonction locale de disjonction

Après la conversion des descriptions de concepts en une forme normale de disjonction, DNF, chaque composant de disjonction C_i du concept C est formé par une conjonction d'éléments E_i, définie sous la forme suivante :

$$C_i \equiv (\cap_0^I \ A_i) \cap \left(\cap_0^J \ R.D_j\right) \cap (\cap_0^K \ \forall R.D_k) \cap (\cap_0^L \ \exists R.D_l) \cap (\cap_0^M \geq n \ R.D_m)$$

$$\cap (\cap_0^N \leq p \ R.D_n)$$

C_i est donc formé d'intersection des primitives A_i, des relations R, des concepts définis par des opérateurs de quantification existentielle $(\exists)$ et universelle $(\forall)$, ou des restrictions de cardinalité.

L'objectif de cette section est de définir une fonction locale de similarité entre une paire de disjonction représentée par : $Sim_{dis}(C_i, D_j)$. Comme nous l'avons indiqué dans la section 6.5.1, six types de constructeurs peuvent être définis comme éléments de conjonction. Ceci nécessite la définition de six fonctions locales de similarité relatives à chaque constructeur. La fonction de similarité $Sim_{dis}(C_i, D_j)$, déterminant la valeur de similarité au niveau de paires de disjonction, est définie par une somme normalisée des similarités

obtenues des fonctions de similarité spécifiques à chaque constructeur, c'est-à-dire en comparant les différents éléments E_i et E_i' ayant le même constructeur.

$$Sim_{dis}(C_i, D_j) = \frac{\sum Sim_{prim} + \sum Sim_{prop} + \sum Sim_{exist} + \sum Sim_{uni} + \sum Sim_{min} + \sum Sim_{max}}{Nb_{elements}}$$

Nous représentons les fonctions de similarité appropriées à chaque type de constructeur, Sim_{prim} pour l'intersection des primitives, Sim_{exist} pour les quantifications existentielles, Sim_{uni} pour les quantifications universelles, Sim_{min} et Sim_{max} pour les restrictions de cardinalités minimales et maximales respectivement. Chaque fonction relative est considérée comme une fonction normalisée, où la valeur de retour de chaque fonction est comprise entre 0 et 1. Le facteur de normalisation $\mathrm{Nb}_{\mathrm{elements}}$ est défini par la somme totale de cardinalités des éléments comparés dans les différentes fonctions de similarité relatives.

Ces fonctions relatives, utilisées dans notre développement de calcul de similarité, sont basées sur le modèle de comparaison des attributs, *The Contrast Model*, proposée par *Tversky*. Ce modèle, présenté dans la section 6.2, définit une approche formelle de calcul de similarité comme une combinaison linéaire des mesures des attributs communs et distinctifs des concepts à apparier. Par conséquent, la valeur de la similarité sémantique augmente proportionnellement avec le nombre d'attributs communs, et inversement proportionnel aux attributs distinctifs entre deux éléments E_i et E_i'. La fonction est décrite comme suit (*cf.* section1.1) (Tversky, 1977):

$$Sim\,(a,b) = \theta f(A \cap B) - \alpha f(A - B) - \beta f(B - A)$$

f est une fonction mesurant l'appariement de deux entités décrites sous la forme d'un ensemble d'attributs. f représente, en l'occurrence, la cardinalité de l'ensemble qui constitue ses arguments, c'est-à-dire le nombre des attributs communs ou distinctifs. $\theta, \alpha, et\ \beta \geq 0$ représentent des coefficients donnés aux attributs communs et distinctifs. Ils reflètent l'importance de chaque paramètre de la fonction.

Afin de normaliser cette fonction de similarité, nous appliquons l'équation suivante :

$$Sim\,(a,b) = \frac{\theta f(A \cap B)}{\theta f(A \cap B) + \alpha f(A - B) + \beta f(B - A)}$$

où f est une fonction de cardinalité ≥ 0 et $\theta, \alpha, et\ \beta$ représentent des coefficients ≥ 0

Dans certaines approches de mesure de similarité basées sur les modèles d'attributs, telles que celles utilisées par (Patil, 2005), la comparaison des attributs est réduite à la détection des équivalences. Ainsi, si deux attributs comparés sont équivalents, alors ils font partie de l'ensemble des attributs communs, sinon ils sont des attributs distinctifs. Cependant, dans notre méthode, nous ne reposons pas sur des équivalences parfaites entre les attributs pour les spécifier dans l'ensemble des attributs communs ou distinctifs, mais nous attribuons un seuil d'acceptation qui affecte cette distinction. Ainsi, deux attributs sont considérés communs si leur mesure de similarité est plus grande qu'un un seuil d'acceptation t défini dans notre algorithme. Si la similarité dépasse ce seuil, ces attributs sont considérés communs, sinon ils sont distinctifs.

Supposons que $N_{E_i \cap E_i'}$ désigne la cardinalité des attributs communs entre les deux éléments E_i et E_i' , $N_{E_i - E_i'}$ désigne la cardinalité des attributs distinctifs de E_i, et $N_{E_i' - E_i}$ désigne la cardinalité des attributs distinctifs de E_i'. La fonction de calcul de similarité locale entre deux éléments E_i et E_i', relative au type de constructeur, est décrite dans les algorithmes suivants:

1. Sim_{prim} est appelée si les éléments E_i et E_i' sont de la forme de conjonction $E_1 \equiv (\cap_0^I\ A_i)$.

CalculSimPrim ($E_i, E_i', N_{E_i \cap E_i'}, N_{E_i - E_i'}, N_{E_i' - E_i}$) // Calculer les trois cardinalités N
Pour chaque concept primitif X dans E_i **faire**
booléen PasCommun = Vrai
Pour chaque concept primitif Y dans E_i' et Si PasCommun **faire**
Si (Sim (X,Y) > t) alors PasCommun = Faux
Fin pour
Si (PasCommun) alors $N_{E_i - E_i'}$ ++
Sinon $N_{E_i \cap E_i'}$ ++
$N_{E_i' - E_i} = N_{E_i' - E_i} + |E_i'|$ - $N_{E_i \cap E_i'}$ - $N_{E_i - E_i'}$
Fin pour

Dans cet algorithme, nous cherchons pour chaque primitive X de E_i s'il existe une primitive Y de E_i' similaire (plus grand que le seuil d'acceptation). Notons que pour les primitives, les calculs de similarité sont extraits du vecteur d'ancrage V_{sim}

définissant des similarités d'ancrage calculées après les inférences effectuées par le raisonneur d'alignement ou introduites par l'utilisateur lui-même.

2. Sim_{prop} est appelée si les éléments E_i et E_i' sont de la forme de conjonction $E_2 \equiv \left(\cap_0^J \ R.D_j\right)$

CalculSimProp (E_i, E_i', $N_{E_i \cap E_i'}$, $N_{E_i - E_i'}$, $N_{E_i' - E_i}$) // Calculer les trois cardinalités N
Pour chaque role R.C dans E_i **faire**
booléen PasCommun = Vrai
Pour chaque role R'.D dans E_i' et Si PasCommun **faire**
Si (Sim (R,R') > t) alors
calculSim(C,D)
Si Sim (C,D) > t alors PasCommun = Faux
Fin pour
Si (PasCommun) alors $N_{E_i - E_i'}$ ++
Sinon $N_{E_i \cap E_i'}$ ++
$N_{E_i' - E_i} = N_{E_i' - E_i} + |E_i'| - N_{E_i \cap E_i'} - N_{E_i - E_i'}$
Fin pour

Dans cet algorithme, nous cherchons pour chaque propriété $R.C$ de E_i s'il existe une propriété similaire $R'.D$ de E_i', et tel que les co-domaines C et D sont également similaires.

3. Sim_{uni} est appelée si les éléments E_i et E_i' sont de la forme de conjonction $E_3 \equiv (\cap_0^K \ \forall R.D_k)$

CalculSimUniversal (E_i, E_i', $N_{E_i \cap E_i'}$, $N_{E_i - E_i'}$, $N_{E_i' - E_i}$)// Calculer les 3 cardinalités N
Pour chaque role ∀ R.C dans E_i **faire**
booléen PasCommun = Vrai
Pour chaque role ∀R'.D dans E_i' et Si PasCommun **faire**
Si (Sim (R,R') > t) alors
calculSim(C,D)
Si Sim (C,D) > t alors PasCommun = Faux
Fin pour
Si (PasCommun) alors $N_{E_i - E_i'}$ ++
Sinon $N_{E_i \cap E_i'}$ ++
$N_{E_i' - E_i} = N_{E_i' - E_i} + |E_i'| - N_{E_i \cap E_i'} - N_{E_i - E_i'}$
Fin pour

Dans cet algorithme, nous cherchons pour chaque propriété $\forall R.C$ de E_i s'il existe une propriété similaire $\forall R'.D$ de E_i', et tel que les co-domaines C et D sont également similaires.

4. Sim_{exist} est appelée si les éléments E_i et E'_i sont de la forme de conjonction $E_4 \equiv (\cap_0^L \ \exists \ R.D_l)$

CalculSimExistentiel (E_i, E'_i, $N_{E_i \cap E'_i}$, $N_{E_i - E'_i}$, $N_{E'_i - E_i}$)//Calculer les trois cardinalités N
Pour chaque role ∃ R.C dans E_i **faire**
booléen PasCommun = Vrai
Pour chaque role ∃R'.D dans E'_i et Si PasCommun **faire**
Si (Sim (R,R') > t) alors
calculSim(C,D)
Si Sim (C,D) > t alors PasCommun = Faux
Fin pour
Si (PasCommun) alors $N_{E_i - E'_i}$ ++
Sinon $N_{E_i \cap E'_i}$ ++
$N_{E'_i - E_i} = N_{E'_i - E_i} + |E'_i| - N_{E_i \cap E'_i} - N_{E_i - E'_i}$
Fin pour

Dans cet algorithme, nous cherchons pour chaque propriété $\exists R.C$ de E_i s'il existe une propriété similaire $\exists R'.D$ de E'_i, et tel que les co-domaines C et D sont également similaires.

5. Sim_{min} est appelée si les éléments E_i et E'_i sont de la forme de conjonction $E_5 \equiv (\cap_0^M \geq n \ R.D_m)$

CalculSimMin (E_i, E'_i, $N_{E_i \cap E'_i}$, $N_{E_i - E'_i}$, $N_{E'_i - E_i}$) // Calculer les trois cardinalités N
Pour chaque role $\geq n_1 R$ dans E_i **faire**
booléen PasCommun = Vrai
Pour chaque role $\geq n_2 \ R'$ dans E'_i et Si PasCommun **faire**
Si (Sim (R,R') > t) alors
Si $n_1 \geq n_2$ alors PasCommun = Faux
Fin pour
Si (PasCommun) alors $N_{E_i - E'_i}$ ++
Sinon $N_{E_i \cap E'_i}$ ++
$N_{E'_i - E_i} = N_{E'_i - E_i} + |E'_i| - N_{E_i \cap E'_i} - N_{E_i - E'_i}$
Fin pour

Dans cet algorithme, nous cherchons pour chaque propriété ayant une cardinalité minimale $(\geq n_1 \ R)$ de E_i s'il existe une propriété similaire $(\geq n_2 \ R)$ de E'_i, tel que $n_1 \geq n_2$.

6. Sim_{max} est appelée si les éléments E_i et E_i' sont de la forme de conjonction $E_6 \equiv (\cap_0^N \leq p\ R.D_n)$

CalculSimMax (E_i, E_i', $N_{E_i \cap E_i'}$, $N_{E_i - E_i'}$, $N_{E_i' - E_i}$) // Calculer les trois cardinalités N
Pour chaque role $\leq n_1 R$ dans E_i **faire**
booléen PasCommun = Vrai
Pour chaque role $\leq n_2\ R'$ dans E_i' et Si PasCommun **faire**
Si (Sim (R,R') > t) alors
Si $n_1 \leq n_2$ alors PasCommun = Faux
Fin pour
Si (PasCommun) alors $N_{E_i - E_i'}$ ++
Sinon $N_{E_i \cap E_i'}$ ++
$N_{E_i' - E_i} = N_{E_i' - E_i} + |E_i'|$ - $N_{E_i \cap E_i'}$ - $N_{E_i - E_i'}$
Fin pour

Dans cet algorithme, nous cherchons pour chaque propriété ayant une cardinalité maximale ($\leq n_1\ R$) de E_i s'il existe une propriété similaire ($\leq n_2\ R$) de E_i', et tel que $n_1 \leq n_2$.

6.6 Synthèse

Nous proposons dans ce chapitre une méthode de recherche de similarité sémantique appropriée au langage OWL DL, autrement dit, basée sur la variante $SHOIN(D)$ de la logique de description. Cette méthode vise à enrichir notre méthode d'intégration des ontologies dans le domaine de la CAO à base de features, afin de pouvoir quantifier les correspondances sémantiques non équivalentes en utilisant les services d'un moteur d'inférence. Notre méthode de calcul de similarité tire parti de l'expressivité riche du langage OWL DL pour mesurer la similarité entre les entités, tout en considérant les connaissances incluses dans la description des entités.

Notre méthode se base sur certains modèles de calcul de similarité développés dans la littérature, tels que le modèle des attributs permettant de calculer la similarité en fonction des attributs communs et distinctifs de deux entités. Notre méthode définit une fonction globale de calcul de similarité qui dépend de la catégorie des entités en question ainsi que de l'ensemble de caractéristiques attribuées à l'entité. Cette fonction globale est basée sur celle définie par l'approche OLA, spécifique au langage OWL Lite. Ainsi, nous avons adapté cette fonction pour l'approprier au langage OWL DL. Ensuite des fonctions locales sont définies

entre les descriptions de concepts, en passant par deux étapes génériques définies dans la plupart des méthodes de calcul de similarité. La première consiste à normaliser les descriptions sous une forme normale, en l'occurrence la forme normale de disjonction DNF. La deuxième consiste à procéder à la comparaison des éléments regroupés selon les constructeurs qui les définissent.

L'avantage de l'intégration de notre méthode de mesure de similarité dans notre approche d'échange des modèles CAO réside dans la capacité de fournir aux utilisateurs des outils d'aide à l'échange de la sémantique des données, permettant d'aboutir à un échange plus complet où des concepts similaires sont suggérés à l'utilisateur en cas de difficulté de conversion. Cependant, cette méthode de similarité doit être testée davantage afin d'obtenir des résultats adaptés aux besoins des utilisateurs. En effet, ces résultats servent de base pour définir certains paramètres, tels que les poids attribués aux différentes fonctions, le seuil d'acceptation, la profondeur des entités recherchées, *etc*. Ces paramètres permettront de tester notre méthode dans des conditions plus réalistes.

Chapitre 7 Mise en œuvre

7.1 Introduction

Nous avons présenté jusqu'ici notre méthode d'échange de données sémantiques des modèles CAO à base de features entre différentes applications. Cette méthode, décrite dans le 3ème chapitre, se base sur le développement d'une ontologie commune, nommée CDFO, servant d'intermédiaire entre les différentes ontologies d'applications CAO. Le développement de cette ontologie en OWL DL est détaillé dans le 4ème chapitre. Ensuite, nous avons défini, dans le 5ème chapitre, notre méthode d'intégration des ontologies, fondée sur les capacités de raisonnement des moteurs d'inférence basés sur les logiques de description. Cette méthode a été enrichie par une méthode de calcul de similarité sémantique entre les entités de différentes ontologies, présentée dans le 6ème chapitre.

Dans ce chapitre, nous allons décrire la mise en place de notre méthode d'échange par l'implémentation d'un prototype d'intégration entre différentes ontologies OWL DL. Nous allons donc présenter, dans un premier temps, les outils nécessaires utilisés lors de la mise en œuvre du prototype, notamment les outils de développement des ontologies ainsi que les outils d'implémentation de notre interface d'échange. Nous allons présenter ensuite l'interface de notre prototype avec des illustrations sur les différentes fonctionnalités disponibles. Enfin, nous présentons un exemple d'intégration entre deux ontologies d'applications CAO, CatiaV5 et SolidWorks.

7.2 Outils de développement du prototype d'échange

L'objectif de cette section est d'introduire les outils utilisés lors de l'implémentation de notre prototype d'échange des modèles de produits. Dans la première section, nous présentons les outils liés au développement des ontologies, notamment l'éditeur d'ontologie *Protégé*, son plug-in permettant l'édition des règles du Web Sémantique, *SWRLTab*, ainsi que les raisonneurs utilisés, Pellet et Jess (*Java Expert System Shell*). Dans la deuxième section,

nous nous intéressons aux outils nécessaires pour l'implémentation de notre interface d'échange, tels que le langage Java, et l'API OWL.

7.2.1 Outils de développement des ontologies

7.2.1.1 Protégé

Une étape fondamentale de notre méthode d'échange des modèles de produit consiste à développer des ontologies d'applications, telles que CatiaV5 et SolidWorks, et une ontologie commune de features de conception, CDFO, représentées avec le langage OWL DL. Pour l'implémentation de ces ontologies et pour la création des règles, nous avons choisi d'utiliser l'environnement de développement d'ontologie *Protégé*, et plus particulièrement son plug-in OWL (Holger, et al., 2004).

Le choix de l'éditeur *Protégé* repose sur plusieurs raisons. En effet, *Protégé* est un éditeur qui permet de construire une ontologie pour un domaine donné, de définir des formulaires d'entrée de données, et d'acquérir des données à l'aide de ces formulaires sous forme d'instances de cette ontologie. Il a été créé à l'Université de Stanford[25], et est largement utilisé par la communauté académique ainsi qu'industrielle dans le domaine du Web Sémantique et la représentation des connaissances. *Protégé* est également une librairie Java, dont le code est *open source. Protégé* est étendu pour créer de véritables applications à base de connaissances en utilisant un moteur d'inférence pour raisonner et déduire de nouveaux faits par l'application de règles d'inférence aux instances de l'ontologie et à l'ontologie elle-même (méta-raisonnement). Cet éditeur est muni d'une interface simple, conviviale et facilite l'utilisation des ontologies. Il fournit également un ensemble riche de constructions d'OWL et d'opérateurs de combinaison de ces opérateurs, par exemple l'intersection, l'union, et la négation.

Protégé ne consiste pas en une application simple de modélisation, mais plutôt une architecture ouverte qui supporte une grande variété de *plug-ins* pour la modélisation des connaissances. La figure 38 illustre une représentation graphique d'une ontologie à l'aide du *plug-in Jambalaya*. Ce dernier est un outil permettant la visualisation des ontologies OWL à l'aide de nœuds et d'arcs.

[25] http://www.stanford.edu/

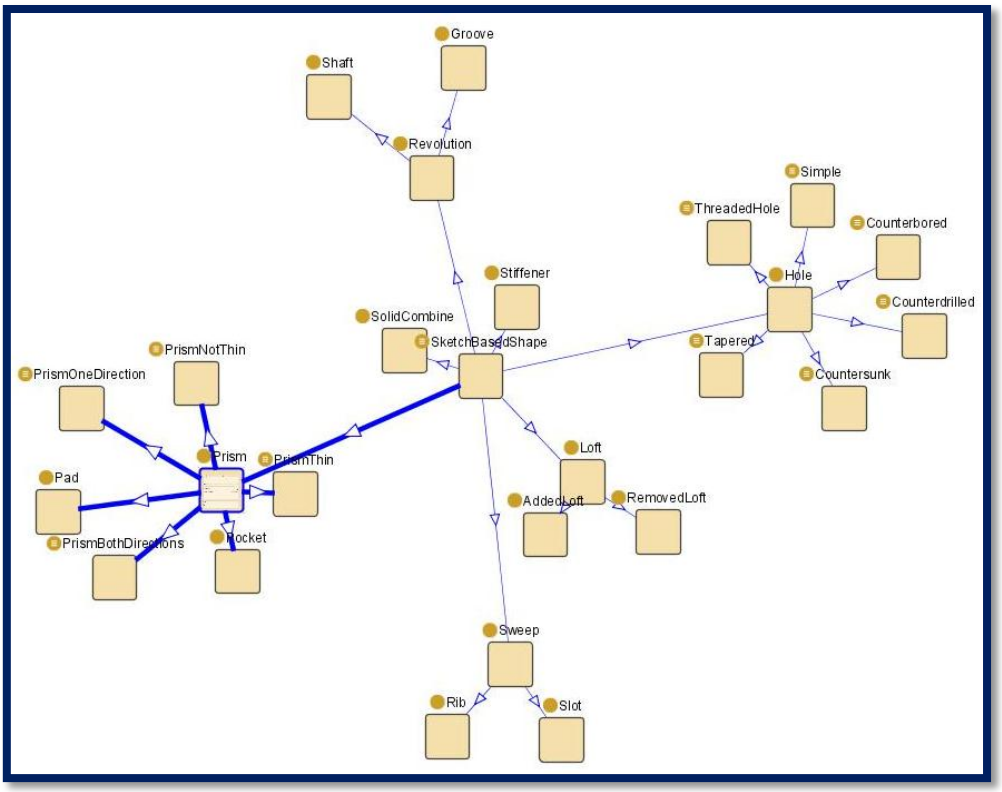


Figure 38. Représentation graphique de segment d'ontologie CDFO avec Jambalaya

Ce logiciel nous a été également utile pour la création et l'édition des règles du Web Sémantique, *SWRL,* moyennant son plug-in *SWRLTab.* La création de ces règles constitue une étape primordiale dans notre méthode d'échange afin de rendre nos ontologies plus riches sémantiquement et de faciliter l'automatisation effectuée par les moteurs d'inférence.

7.2.1.2 Moteurs d'inférence

Notre méthode d'intégration des ontologies repose principalement sur les services de raisonnement fournis par les moteurs d'inférence basés sur les logiques descriptives. En effet, il est possible de faire appel à un raisonneur, comme Pellet, directement dans les dernières versions de *Protégé* (*e.g.*la version 3.4). Le moteur d'inférence Pellet est utilisé pour le raisonnement des faits supplémentaires à partir de la base de connaissance définie dans une ontologie.

Pellet (*version 1.5.2*) est un moteur développé et soutenu commercialement par Clark & Parsia LLC[26]. Son code est implémenté en Java et disponible en source libre. Pellet

[26] http://www.clarkparsia.com/

supporte l'expressivité totale du langage OWL DL, et permet ainsi d'effectuer des inférences sur des ontologies représentées en OWL DL, y compris le raisonnement sur les nominaux (classes énumérées). Plus récemment, il a été étendu pour considérer toutes les fonctionnalités du langage OWL1.1, à l'exception des types de données d'arité n (Sirin, et al., 2007). Il implante une procédure de décision fondée sur l'algorithme des tableaux pour des $TBox$ génériques (subsomption, satisfiabilité, classification) et pour des $ABox$ (récupération et réponse aux requêtes conjonctives). Il supporte les API OWL, DIG et Jena (Wilkinson, et al., 2003).

Cependant, Pellet ne permet pas les inférences appliquées aux règles définies avec le langage SWRL. Nous avons donc utilisé le moteur d'inférence JESS (*Java Expert System Shell*) (Friedman-Hill, 2007). JESS propose un environnement pour la création et l'édition de systèmes à base de règles. Il représente un moteur de règles qui permet de construire des systèmes capables de raisonner en utilisant de la connaissance fournie sous forme de règles déclaratives. Il est utilisé dans notre prototype pour les inférences effectuées sur les règles du Web Sémantique SWRL définies dans nos ontologies. JESS est disponible pour les communautés de recherche sous des licences académiques gratuites. Notons qu'une phase préalable de raisonnement consiste à convertir la syntaxe OWL vers une syntaxe spécifique à JESS, et une conversion réciproque, en OWL, sera appliquée après l'exécution du raisonnement de JESS. Le plug-in de JESS dans *Protégé* est illustré dans la figure 39.

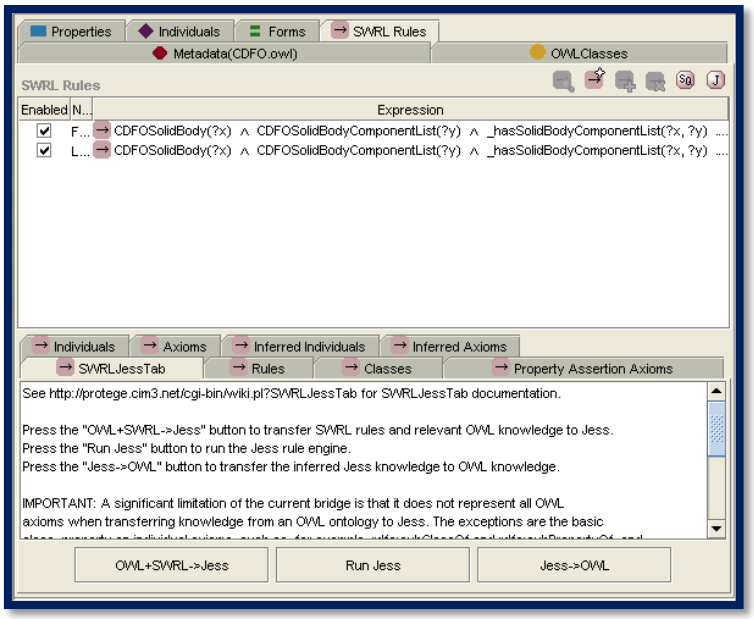


Figure 39. Le Plug-in Jess dans *Protégé*

7.2.2 Outils d'implémentation de notre application

7.2.2.1 Java

Pour l'implémentation de notre application d'intégration des ontologies OWL, nous avons utilisé le langage de programmation orienté objet *Java*. En effet, grâce à sa portabilité et à son indépendance de plate-forme, Java permet l'exécution des applications indépendamment des architectures matérielles ou des systèmes d'exploitation sur lesquels ces applications sont utilisées. De plus, beaucoup de librairies et d'APIs de développement d'ontologie sont basées sur le langage Java, ce qui rend ce dernier approprié pour les applications utilisant des ontologies.

Concernant l'environnement de développement, nous avons utilisé l'IDE Eclipse. Ce dernier permet potentiellement de créer des projets de développement mettant en œuvre différents langages de programmation, y compris Java. La libraire SWT, *Standard Widget Toolkit*, développée par IBM et la fondation Eclipse[27], est également utilisée pour créer l'interface graphique d'utilisateur, GUI (*Graphic User Interface*). SWT est une bibliothèque graphique libre pour Java, formée de bibliothèques de composants graphiques (texte, label, bouton, panel), et de différents utilitaires nécessaires pour développer une interface graphique en Java. L'avantage d'utilisation de SWT est qu'il permet à Java d'utiliser les composants natifs du système d'exploitation sur lequel l'application s'exécute. Par conséquent, l'application Java devient en général plus fluide et plus rapide.

7.2.2.2 Interfaces de programmation pour OWL

Les interfaces de programmation (API) pour les langages du Web Sémantique constituent des librairies permettant de créer, gérer, et manipuler des ontologies représentées avec ces langages. Différentes APIs ont été développées pour considérer le langage OWL, telles que Jena (McBride, 2002), OWL API (Bechhofer, et al., 2003), *etc.* L'utilisation de ces APIs permet aux développeurs de s'affranchir de la programmation liée à l'analyse syntaxique (*parsing*) et à l'écriture (*sérialisation*) des syntaxes spécifiques. Elle facilite la tâche et permet également aux développeurs de se concentrer sur des opérations sur les ontologies à un niveau supérieur d'abstraction (Ferreira Da Silva, 2007).

[27] http://www.eclipse.org

Jena, développé par *Hewlett-Packard Development Company*, HP, est une API fondée sur le langage RDF. Autrement dit, son modèle interne de représentation se conforme à une représentation en graphes RDF. Ainsi, les formalismes d'ontologies qui peuvent être manipulés par cette API sont ceux qui sont construits au-dessus de RDF, spécifiquement il s'agit des langages RDFS, DAML+OIL, et OWL y compris ses trois sous langages. L'inconvénient d'utilisation de Jena pour le langage OWL réside dans le fait que RDF constitue un niveau d'abstraction inférieur à celui du langage OWL.

L'API OWL est l'interface de programmation fondamentale utilisée dans notre développement permettant au noyau de l'application de manipuler directement les ontologies OWL et de communiquer avec les moteurs d'inférences. Contrairement à l'API Jena, l'interface OWL API fournit un modèle interne de représentation reposant sur la syntaxe abstraite du langage OWL.Le code de cette API, développée en Java, est ouvert et disponible en licence LGPL. Cette API fournit des fonctionnalités de création, de stockage, de manipulation, et de raisonnement des ontologies OWL. Ainsi, une ontologie contient des axiomes et des faits, qui à leur tour fournissent l'information concernant les classes, les propriétés et les individus dans l'ontologie. Les annotations sont aussi représentées comme des axiomes. Cette API permet également l'interfaçage pour des moteurs d'inférences. Cette dernière fonctionnalité est particulièrement intéressante pour retrouver, par exemple, les individus liés à travers une propriété donnée, les valeurs des propriétés pour un individu, etc.

7.3 Interface de notre application

La dernière étape de notre travail consiste à concevoir l'interface de notre prototype afin qu'il puisse être utilisé pour l'échange des données sémantiques d'un produit. Nous décrivons, dans cette section, l'interface graphique de notre application servant à l'intégration de deux ontologies CAO, X et Y, basée sur la méthode d'intégration présentée dans le chapitre 5. L'objectif de cette application est de fournir une interface permettant à l'utilisateur de réaliser l'alignement de deux ontologies par l'établissement des correspondances entre les entités des deux ontologies concernées. Cette mise en correspondance permettra de réaliser l'échange des données sémantiques, y compris les features, d'un modèle de produit, représenté par des instances dans l'ontologie X, vers l'ontologie destinataire Y. Notons que les deux ontologies X et Y représentent potentiellement des ontologies d'applications CAO ou notre ontologie commune, CDFO. L'ontologie source X est instanciée par un modèle de

produit, tel que une pièce ou un assemblage créé avec le logiciel de *X*, tandis que l'ontologie destinataire *Y* représente un système destinataire de CAO non instancié. L'interface graphique de notre application, illustrée dans la figure 40, est composée de deux sections principales :

Section de contrôle (A) : Elle permet à l'utilisateur d'interagir avec les différentes fonctionnalités principales de notre application. Cette section est composée à son tour d'un ensemble de groupes de contrôles, respectivement pour l'ontologie source, l'ontologie destinataire, l'ontologie de correspondances et l'inférence. Ces groupes seront détaillés dans les sous sections suivantes.

Section d'affichage (B) : C'est dans cette section que les résultats sont affichés, en fonction des manipulations effectuées par l'utilisateur dans la section de contrôle.

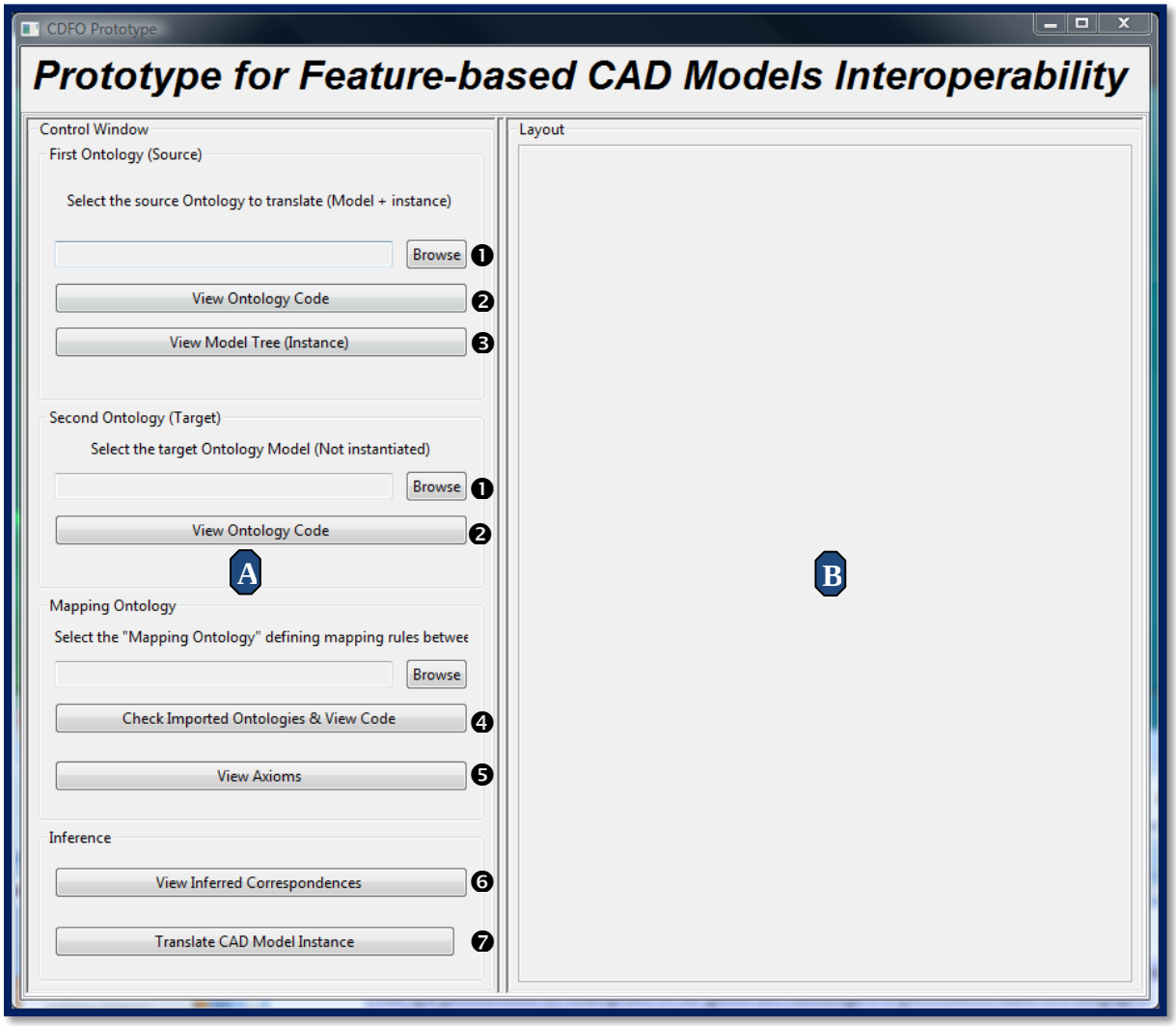


Figure 40. Schéma de L'interface graphique de notre prototype

Comme illustrée dans la figure 40, notre interface possède les fonctionnalités suivantes :

- **Chargement des ontologies ❶** : L'interface doit tout d'abord avoir les fonctionnalités pour charger les ontologies concernées, notamment l'ontologie source (*Source Ontology*), l'ontologie destinataire (*Target Ontology*), et l'ontologie de correspondance (*Mapping Ontology*). Une boîte de dialogue est ensuite ouverte afin de spécifier les endroits ou les adresses URI des ontologies OWL à charger. Notons que l'ontologie source est une ontologie instanciée qui représente les données d'un modèle de produit. Tandis que l'ontologie destinataire est supposée définie au niveau terminologique, autrement dit ne contient pas des instances.
- **Visualisation de l'ontologie ❷**: Notre interface offre également à l'utilisateur la possibilité de visualiser le code OWL qui correspond à chacune des ontologies chargées dans l'application. La figure 41 montre que l'appui sur le bouton '*View Ontology Code*' provoque l'affichage du code OWL de l'ontologie dans la fenêtre '*Layout'* de notre interface.

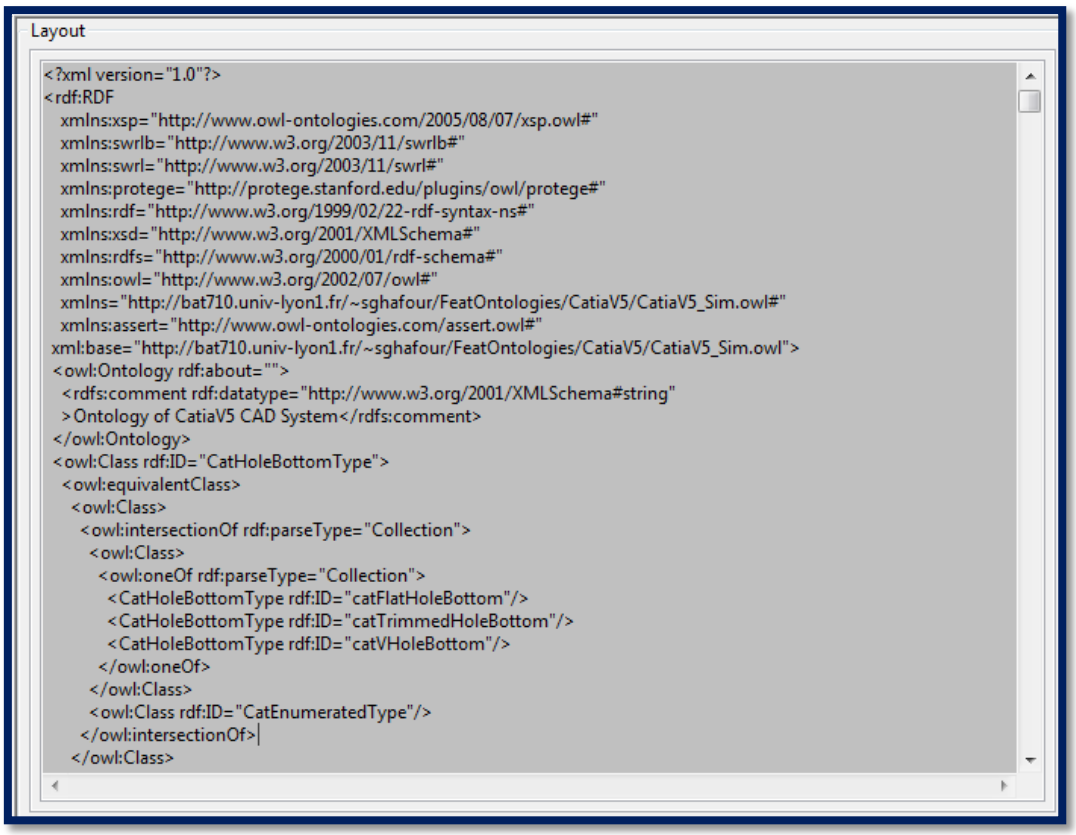


Figure 41. Affichage du code de l'ontologie

- **Visualisation de l'arbre de spécifications ❸** : Cette fonctionnalité n'est disponible que pour l'ontologie source instanciée par un modèle de produit. Elle permet d'afficher l'arbre de spécifications associé à ce modèle, incluant l'historique de construction et l'ensemble des features de conception utilisé lors de la création du modèle. La figure 42 montre l'arbre obtenu pour un modèle composé d'une pièce créée dans CatiaV5.

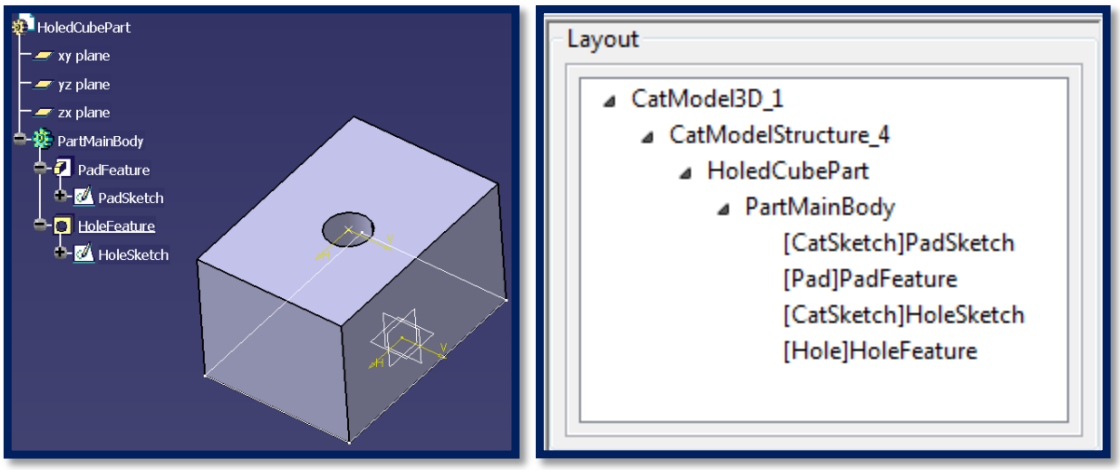


Figure 42. (a) modèle dans CatiaV5. (b) l'arbre de spécifications dans notre application

- **Vérification de l'ontologie de correspondance ❹:** Une ontologie de correspondance est supposée importer les deux ontologies en question. Ce bouton vérifie la cohérence entre les ontologies chargées dans l'application et celles importées dans l'ontologie de correspondance avant d'afficher le code de l'ontologie de correspondance. Différents types d'erreurs peuvent être affichés. La figure 43 montre l'exemple d'une erreur détectée lorsque l'ontologie source chargée dans l'application est différente de l'ontologie source importée par l'ontologie de correspondance.

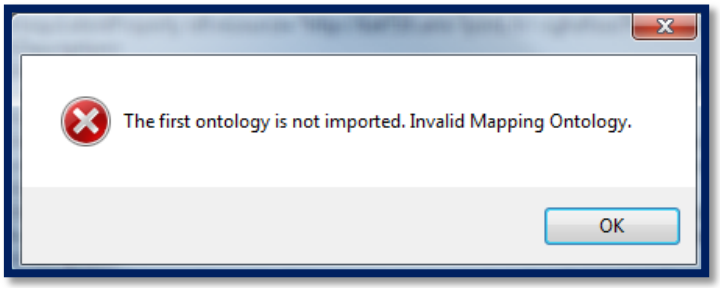


Figure 43. Exemple d'erreur d'import de l'ontologie source

- **Gestion des axiomes de correspondance ❺:** Cette fonctionnalité, associée à l'ontologie de correspondance, permet de visualiser et de gérer l'ensemble des axiomes définis dans une ontologie de correspondance. Les axiomes entre les entités de différentes ontologies sont affichés dans un tableau de trois colonnes : le type d'axiome, l'entité dans l'ontologie source, et l'entité dans l'ontologie destinataire. Ce tableau est illustré dans la figure 44 :

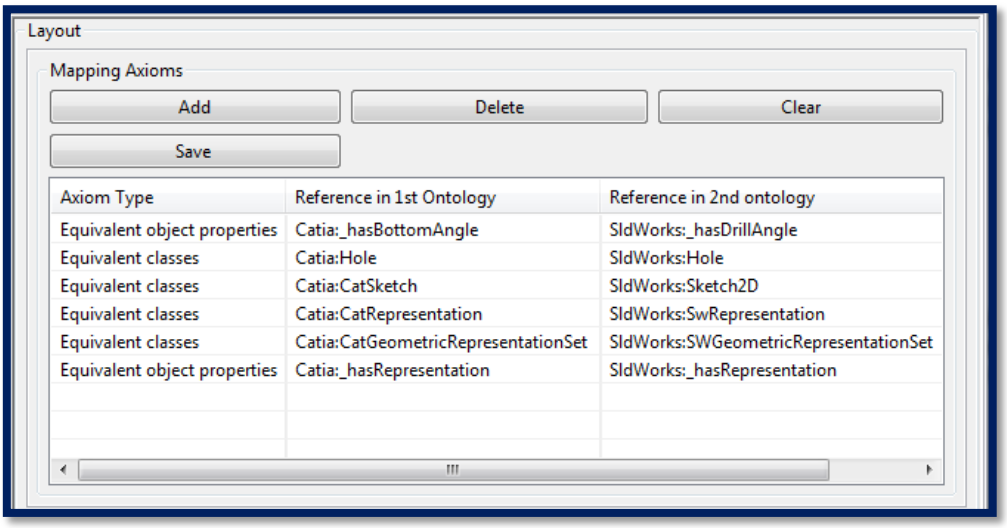


Figure 44. Tableau d'axiomes définis dans l'ontologie de correspondance

Notons que différents boutons ont été définis pour la gestion des axiomes, notamment pour l'ajout (*Add*), la suppression d'un axiome (*Delete*), la suppression de l'ensemble des axiomes (*Clear*), et la sauvegarde de l'ontologie de correspondance (*Save*) après une modification éventuelle des axiomes. Par exemple, pour l'ajout des axiomes, une nouvelle boîte de dialogue, illustrée dans la figure 45, s'affiche afin de permettre à l'utilisateur de choisir le type de l'axiome désiré, la référence de l'entité dans l'ontologie source, ainsi que la référence de l'entité dans l'ontologie destinataire. Les « *comboBox* » sont définis relativement au type d'axiome choisi. Autrement dit, si le type d'axiome est une équivalence entre deux classes, les *comboBox* des références affichent uniquement les classes appartenant respectivement aux deux ontologies.

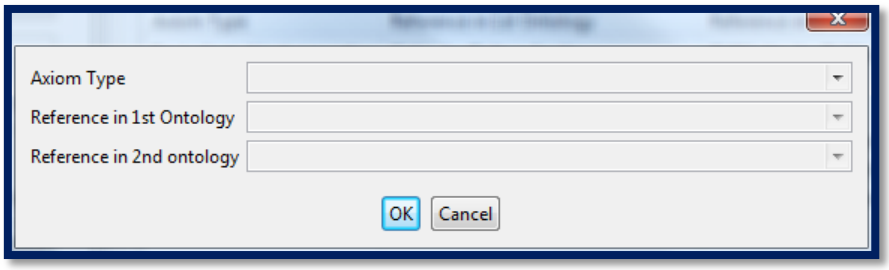


Figure 45. Boîte de dialogue permettant l'ajout d'un axiome de correspondance

- **Inférence au niveau terminologique ❻:** Cette fonctionnalité permet d'obtenir les axiomes de correspondances inférés entre les entités des deux ontologies. Elle est principalement basée sur les services d'inférences fournis par le raisonneur, Pellet, appelé dans notre prototype. La liste des inférences obtenues est affichée dans un tableau, comme illustré dans la figure 46. Cette liste dépend spécifiquement des axiomes définis avec la fonctionnalité précédente, où l'ajout d'un nouvel axiome dans l'ontologie de correspondance engendre une nouvelle liste de correspondances inférées.

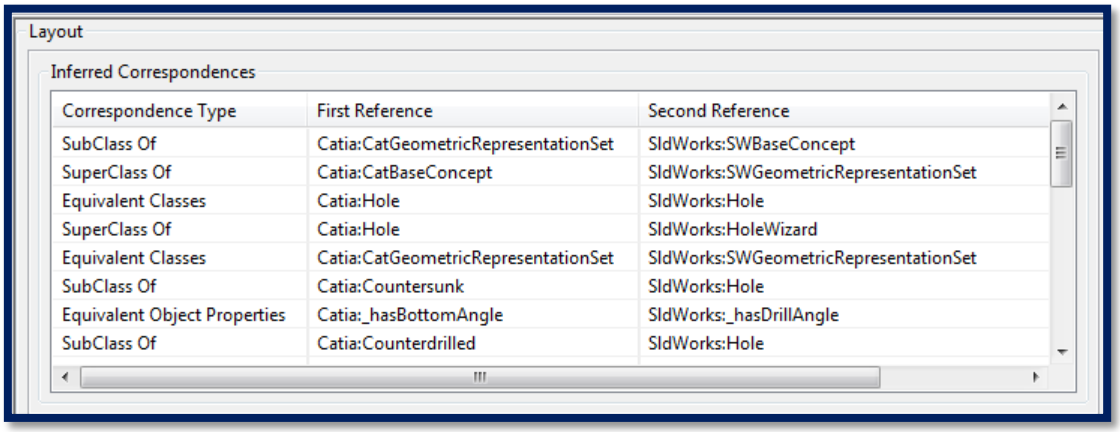


Figure 46. Liste des inférences de correspondances entre les deux ontologies

- **Inférence factuel du modèle de produit ❼:** A cette étape se réalise le transfert des instances définies pour le modèle de produit entre les deux ontologies. Cette étape consiste à inférer pour chaque instance définie dans le modèle du produit le type correspondant dans l'ontologie destinataire. La liste d'inférence des instances obtenues est illustrée dans la figure 47.

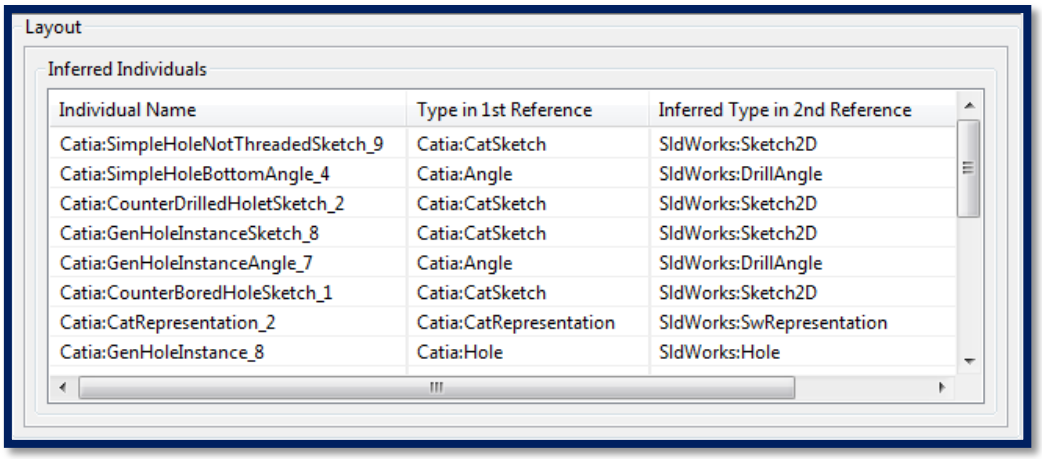


Figure 47. Inférence factuel pour les instances du modèle de produit

7.4 Exemple d'échange

Afin de tester notre prototype, nous avons considéré l'échange de données entre deux logiciels de conception assistée par ordinateur : CatiaV5 et SolidWorks. Nous allons dans un premier temps présenter les deux logiciels CatiaV5 et SolidWorks, et ensuite leurs ontologies développées dans le cadre de notre thèse.

7.4.1 Systèmes CAO utilisés dans notre exemple

Nous avons utilisé deux logiciels très connus dans le domaine de CAO, il s'agit de CatiaV5 et de SolidWorks. Ces deux systèmes ont été choisis pour des raisons de disponibilités, mais également parce que ceux-ci sont représentatifs des systèmes de modélisation à base de features utilisés dans les entreprises que nous ciblons. La mise en place de notre travail se limite aux modèles géométriques des pièces créés à partir des features de conception volumiques, disponibles dans les systèmes CAO concernés.

CatiaV5, le logiciel de CAO développé par Dassault Systèmes[28], DS, est utilisé par un grand nombre des plus grandes entreprises dans différents secteurs industriels, notamment l'aéronautique, la construction navale, l'automobile, l'électronique, et la conception des biens de consommation sophistiqués. Ce logiciel est un modeleur d'assemblage à base d'éléments

[28] http://www.3ds.com/fr/

intelligents, ou features. Les modèles créés avec CatiaV5 sont caractérisés par l'extension .CATPart (pour les pièces) ou .CATProduct (pour les assemblages).

SolidWorks, un logiciel de CAO développé par une autre division de la société mère, DS, est utilisé par des dizaines de milliers d'entreprises les plus modestes pour concevoir des outils et des équipements de production destinés à leurs grands donneurs d'ordre. Par conséquent, une minorité néanmoins importante d'utilisateurs de SolidWorks souhaite pouvoir lire des données CatiaV5. Une communauté plus importante d'utilisateurs CatiaV5 cherche à importer des modèles d'outils et d'équipements de production conçus par leurs fournisseurs à l'aide de SolidWorks.

Dans le cadre d'implémentation de notre prototype d'échange, nous avons développé deux ontologies d'application CAO, pour CatiaV5 et SolidWorks, avec le langage OWL DL, afin de fournir une représentation formelle des différentes entités définies dans chaque application. La construction de ces ontologies n'a été possible qu'après une étude approfondie des possibilités de ces systèmes CAO, ceci afin de permettre l'extraction de l'ensemble des features existants dans chaque logiciel. Cette étape d'analyse a permis donc de définir un ensemble de concepts devant faire partie de l'ontologie associée à chacun des deux logiciels. Le développement de nos ontologies est réalisé avec l'éditeur *Protégé*, et plus particulièrement son plug-in OWL.

7.4.1.1 Ontologie d'application CatiaV5

Nous avons développé une ontologie d'application CatiaV5 qui représente les entités existantes du logiciel pour la modélisation des produits volumiques (3D) en se basant sur la libraire de features fournie avec le logiciel. Le développement de cette ontologie est basé sur la classification des features dans CatiaV5 présentée dans l'annexe A. La figure 48 illustre deux segments de hiérarchies de classes et de propriétés définies dans notre ontologie.

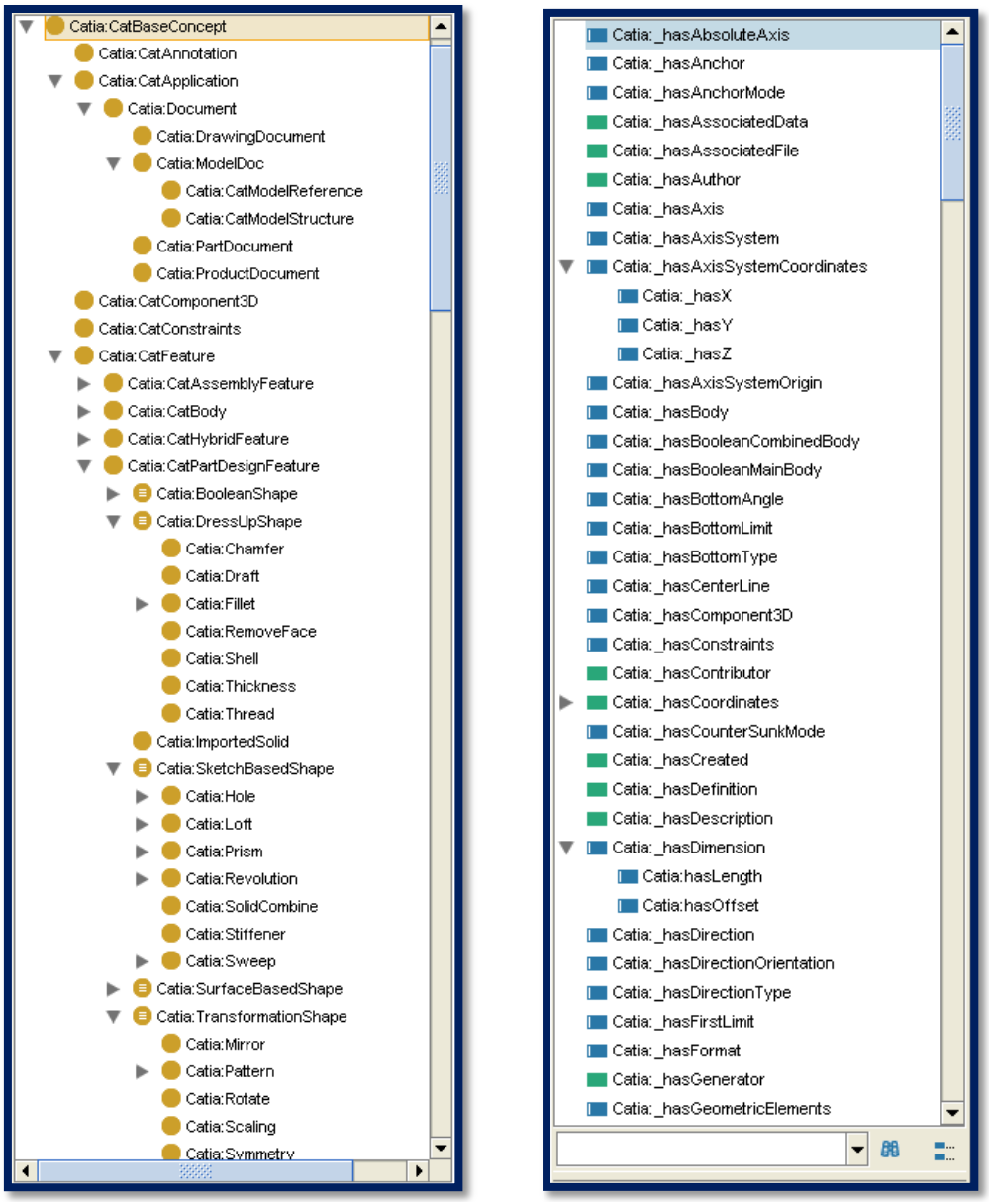


Figure 48. Segment de listes des Classes (a) et des propriétés en CatiaV5 sous *Protégé*

Plusieurs classes de cette ontologie peuvent être éventuellement définies sémantiquement en fonction d'autres classes ou de restrictions sur des propriétés. La figure 49 illustre l'exemple de description d'un trou « *CounterDrilled* », autrement dit un trou complexe ayant trois paramètres supplémentaires : angle, diamètre, et profondeur. Ces descriptions de classes s'avèrent intéressantes pour réaliser des classifications automatiques par les moteurs d'inférence.

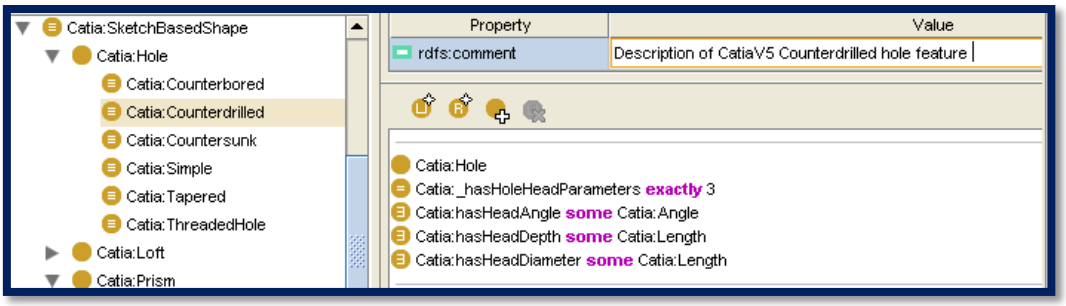


Figure 49. Description du trou "CounterDrilled" dans l'ontologie d'application CatiaV5

7.4.1.2 Ontologie d'application SolidWorks

Nous avons également développé une ontologie d'application SolidWorks comprenant les entités pour la modélisation à base de features des produits volumiques. Le développement de cette ontologie est basé sur la classification des features dans SolidWorks, présentée dans annexe A. La figure 50 illustre deux segments de hiérarchies de classes et de propriétés définies dans notre ontologie.

Plusieurs classes ont été décrites formellement, par exemple la description d'un trou « *SWCounterDrilled* ». Cette description dans *Protégé* est illustrée dans la figure 51.

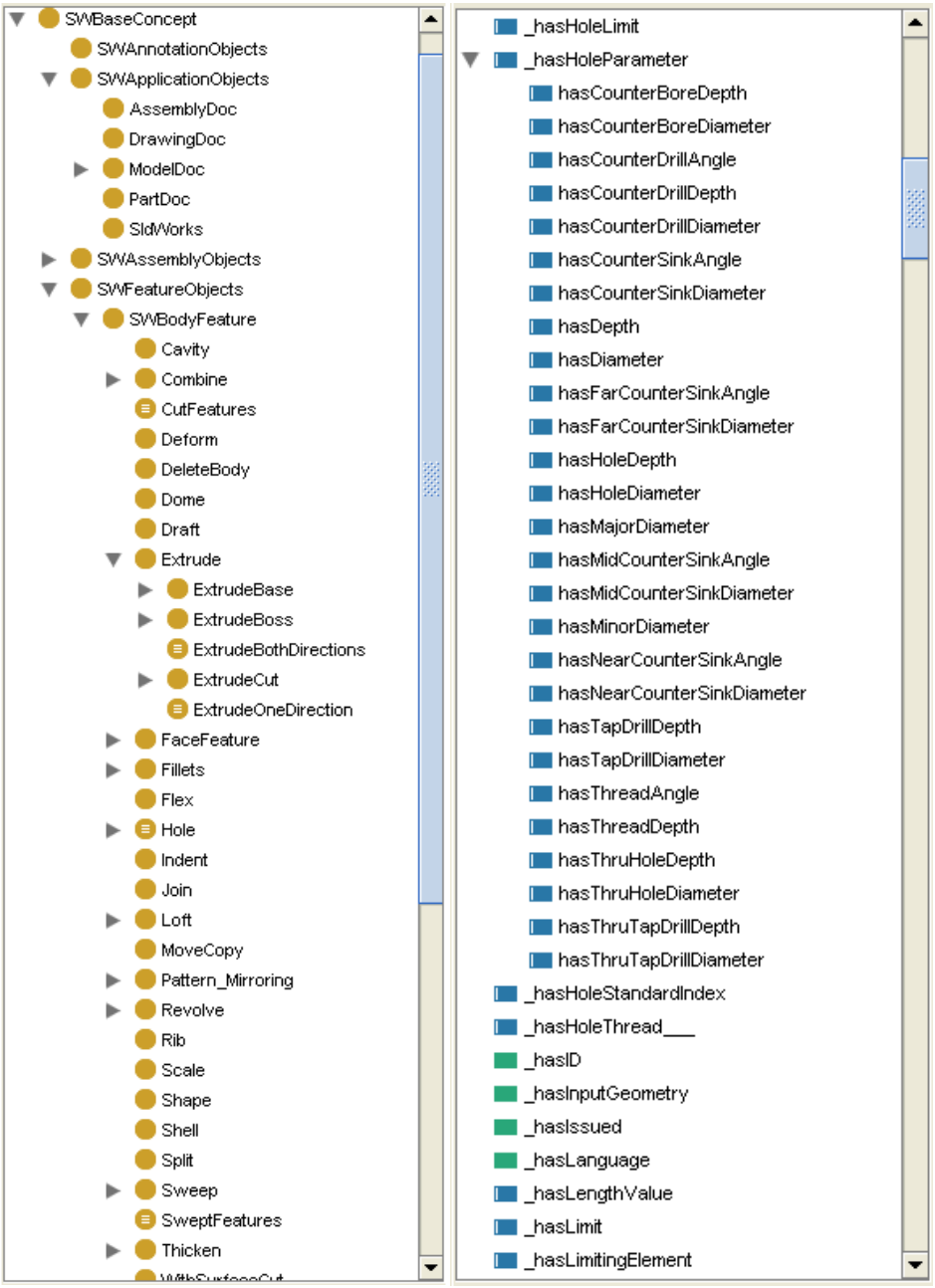


Figure 50. Segments de classes et des propriétés en SolidWorks sous *Protégé*

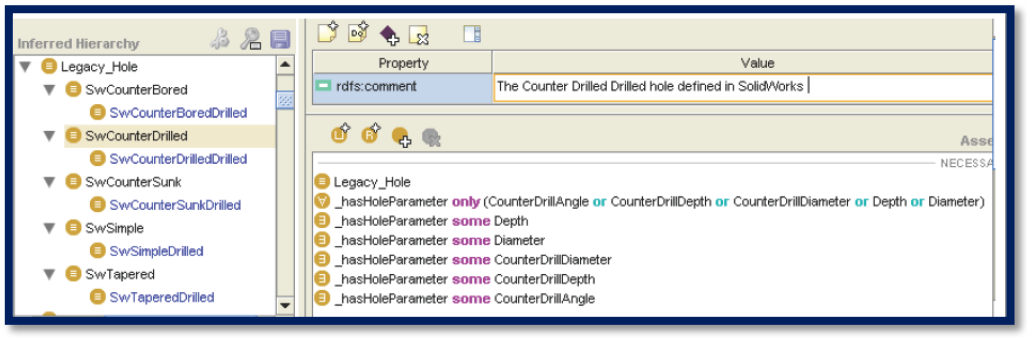
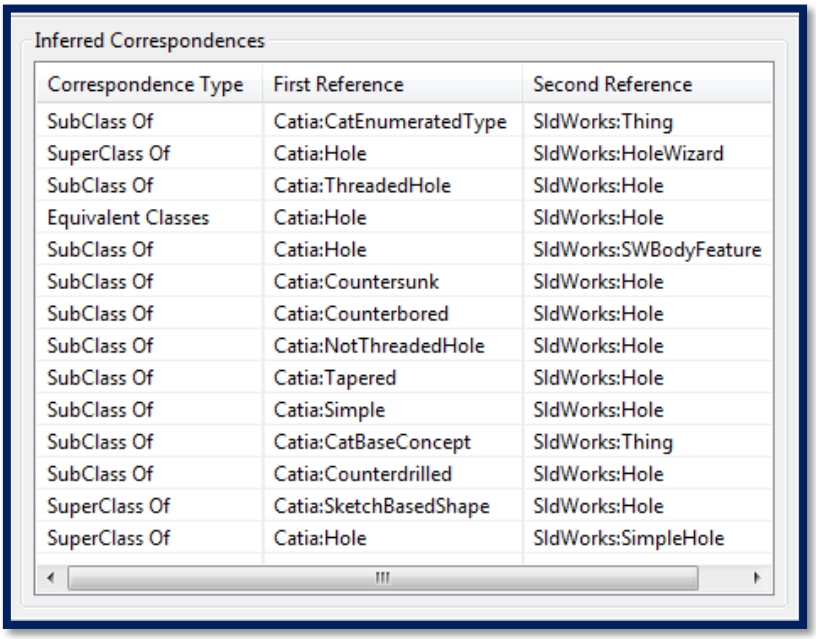


Figure 51. Description du trou "CounterDrilled" dans SolidWorks

7.4.2 Expérimentations

Nous avons effectué plusieurs tests d'expérimentation sur l'intégration des ontologies entre CatiaV5 et SolidWorks. Notre méthode se base sur l'intégration de deux ontologies moyennant une ontologie commune que nous avons développée. Concrètement, l'intégration se réalise d'abord entre l'ontologie source d'application et l'ontologie commune CDFO, ensuite entre l'ontologie commune et l'ontologie destinataire d'application.

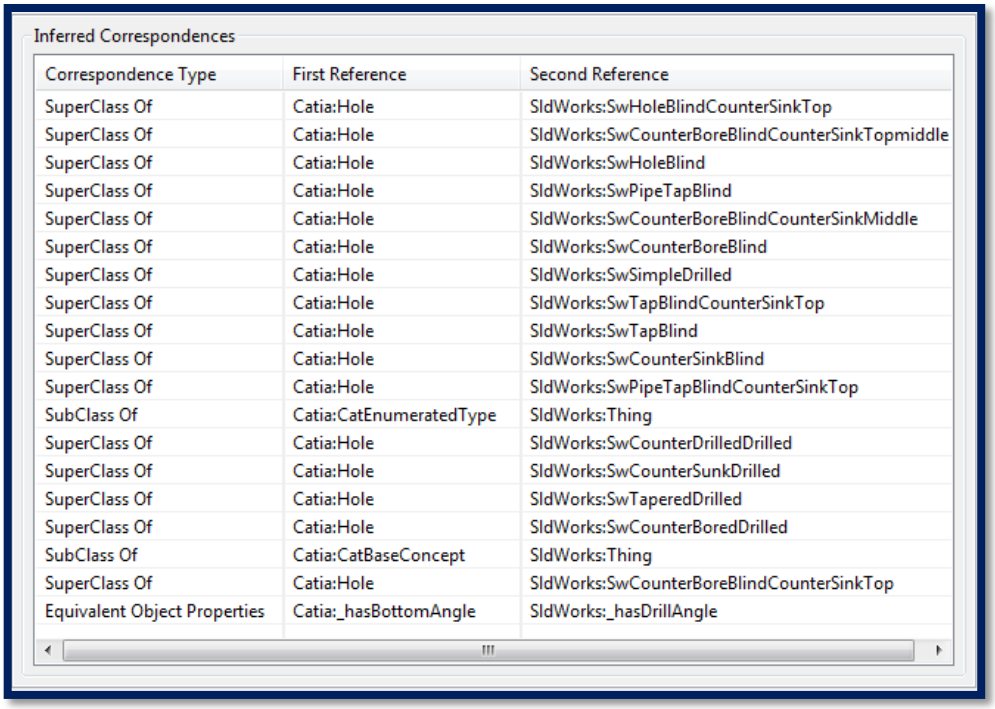
Dans ce cadre, nous ajoutons des axiomes de correspondance entre des entités des deux ontologies, et nous faisons appel au moteur d'inférence afin de déduire d'autres inférences de correspondance qui en résultent. Nous ajoutons ainsi un axiome à l'ontologie de correspondance afin de définir une relation d'équivalence, par exemple, entre deux classes des deux ontologies CatiaV5 et SolidWorks respectivement. Il s'agit par exemple de spécifier que le concept primitif « *Catia :Hole* » de CatiaV5 est équivalent au concept «*SolidWorks :Hole* » de SolidWorks. En faisant appel au raisonneur Pellet, cet axiome d'équivalence a engendré une liste de correspondances, généralement des relations de subsomption, entre différentes entités de CatiaV5 et de SolidWorks. La figure 52 illustre la liste de correspondances inférées à partir de l'axiome d'équivalence entre les trous.


Inferred Correspondences

Correspondence Type	First Reference	Second Reference
SubClass Of	Catia:CatEnumeratedType	SldWorks:Thing
SuperClass Of	Catia:Hole	SldWorks:HoleWizard
SubClass Of	Catia:ThreadedHole	SldWorks:Hole
Equivalent Classes	Catia:Hole	SldWorks:Hole
SubClass Of	Catia:Hole	SldWorks:SWBodyFeature
SubClass Of	Catia:Countersunk	SldWorks:Hole
SubClass Of	Catia:Counterbored	SldWorks:Hole
SubClass Of	Catia:NotThreadedHole	SldWorks:Hole
SubClass Of	Catia:Tapered	SldWorks:Hole
SubClass Of	Catia:Simple	SldWorks:Hole
SubClass Of	Catia:CatBaseConcept	SldWorks:Thing
SubClass Of	Catia:Counterdrilled	SldWorks:Hole
SuperClass Of	Catia:SketchBasedShape	SldWorks:Hole
SuperClass Of	Catia:Hole	SldWorks:SimpleHole

Figure 52. Correspondances inférées des trous entre CatiaV5 et SolidWorks

En définissant un axiome d'équivalence de propriétés entre les deux propriétés d'objets « *Catia :HasBottomAngle* » et « *SolidWorks :hasDrillANgle* », ceci engendre une nouvelle liste de correspondances inférées, illustrée dans la figure 53.


Inferred Correspondences

Correspondence Type	First Reference	Second Reference
SuperClass Of	Catia:Hole	SldWorks:SwHoleBlindCounterSinkTop
SuperClass Of	Catia:Hole	SldWorks:SwCounterBoreBlindCounterSinkTopmiddle
SuperClass Of	Catia:Hole	SldWorks:SwHoleBlind
SuperClass Of	Catia:Hole	SldWorks:SwPipeTapBlind
SuperClass Of	Catia:Hole	SldWorks:SwCounterBoreBlindCounterSinkMiddle
SuperClass Of	Catia:Hole	SldWorks:SwCounterBoreBlind
SuperClass Of	Catia:Hole	SldWorks:SwSimpleDrilled
SuperClass Of	Catia:Hole	SldWorks:SwTapBlindCounterSinkTop
SuperClass Of	Catia:Hole	SldWorks:SwTapBlind
SuperClass Of	Catia:Hole	SldWorks:SwCounterSinkBlind
SuperClass Of	Catia:Hole	SldWorks:SwPipeTapBlindCounterSinkTop
SubClass Of	Catia:CatEnumeratedType	SldWorks:Thing
SuperClass Of	Catia:Hole	SldWorks:SwCounterDrilledDrilled
SuperClass Of	Catia:Hole	SldWorks:SwCounterSunkDrilled
SuperClass Of	Catia:Hole	SldWorks:SwTaperedDrilled
SuperClass Of	Catia:Hole	SldWorks:SwCounterBoredDrilled
SubClass Of	Catia:CatBaseConcept	SldWorks:Thing
SuperClass Of	Catia:Hole	SldWorks:SwCounterBoreBlindCounterSinkTop
Equivalent Object Properties	Catia:_hasBottomAngle	SldWorks:_hasDrillAngle

Figure 53. Correspondances inférées des trous entre CatiaV5 et SolidWorks

7.5 Synthèse

L'objectif de ce chapitre est d'implémenter notre prototype pour l'échange de données selon la méthode d'intégration décrite dans le chapitre 5. Nous n'avons pas considéré dans ce chapitre l'implémentation de la méthode de calcul de similarité. Cette implémentation repose sur des ontologies d'application CAO et l'ontologie commune, CDFO, servant d'intermédiaire entre les ontologies d'application. Il est donc nécessaire d'utiliser des outils de développement d'ontologies et d'outils d'implémentation de notre application d'intégration de ces ontologies. Ces outils sont présentés brièvement au début de ce chapitre.

Nous avons décrit ensuite l'interface de notre application d'intégration développée en Java. Cette application fournit plusieurs fonctionnalités de chargement des ontologies en question, de l'affichage du code des ontologies en OWL, de gestion des axiomes définis par l'utilisateur et des axiomes de correspondances inférés avec le raisonneur Pellet.

Afin de tester notre application, nous avons choisi, dans un premier temps, deux logiciels de CAO, CatiaV5 et SolidWorks. Deux ontologies d'application associées à chaque application ont été développées dans le cadre de notre thèse. Nous avons terminé par deux exemples d'axiomes de correspondance, définis dans l'ontologie de correspondance, ainsi que les correspondances inférées engendrées par l'ajout de ces axiomes.

Bien que notre travail n'aboutisse pas à un logiciel de transfert complet des données, notre interface montre dans un exemple assez simple que notre approche basée sur les ontologies et les services d'inférences conduisent à de nouvelles correspondances permettant l'interopérabilité sémantique des données des produits. Cependant, d'autres applications peuvent être intégrées afin de rendre l'environnement collaboratif plus efficace.

Conclusion et Perspectives

Dans un environnement collaboratif de développement de produit, plusieurs acteurs, ayant différents points de vue et intervenant dans plusieurs phases du cycle de vie de produit, doivent communiquer et échanger des données entre eux. Ces données existent sous différents formats qui divergent au niveau syntaxique, structurel et sémantique. Les exigences industrielles de réduction du temps et du coût de production nécessitent l'amélioration de l'interopérabilité sémantique entre les différents processus de développement. Les techniques d'échange de données sont assez importantes. Ainsi, l'objectif serait de concevoir des méthodes et des outils informatiques pouvant aider à surmonter les problèmes d'hétérogénéité afin de répondre aux exigences et aux contraintes imposées sur le monde industriel. La communication peut être effectuée entre différentes applications du cycle de vie, ou entre différents systèmes situés dans le même domaine.

Dans le domaine de la CAO, les modèles géométriques étaient considérés comme les plus appropriés pour échanger des informations d'un produit. Cependant, ces informations géométriques ne sont pas suffisantes pour la réutilisation des informations sémantiques définies par les concepteurs et pour l'édition de ces modèles. Il est en effet nécessaire de pouvoir échanger d'autres types de données, notamment la sémantique associée à ces informations géométriques. Dans ce but, nous nous sommes intéressés, dans notre travail, à l'échange des modèles « *intelligents* » de CAO, définis en termes d'historique de construction, de fonctions « *intelligentes* » de conception, nommées features, de leurs paramètres et des contraintes associées aux modèles.

Dans ce cadre, nous avons focalisé notre étude, dans le 1er chapitre, sur les techniques de modélisation à base de features. Les features constituent l'élément de base dans notre approche d'échange de données entre différentes applications de CAO. Dans les différents travaux liés aux features, les chercheurs n'ont pas réussi à obtenir un consensus sur la définition de l'ensemble des features. Initialement, les features désignaient les features de forme. Ensuite, le terme a connu un sens plus approfondi où un feature peut contenir d'autres types d'informations telles que des contraintes, des informations relatives au matériau utilisé

pour la pièce, les tolérances, la couleur, etc. Dans notre étude, nous nous sommes intéressés aux features de conception définis dans les applications CAO à base de feature, où un feature est défini comme un élément paramétrique de conception, ayant une signification d'ingénierie et une influence sur la représentation géométrique de la pièce.

L'échange de données est un problème fondamental à traiter lors de la mise en œuvre d'un développement coopératif. Nous avons présenté, dans le 2ème chapitre, une étude bibliographique sur les techniques et les standards actuels d'échange de données entre les modèles CAO, en portant une attention particulière à l'échange des features correspondant à une représentation explicite de l'intention de conception. Les différentes études réalisées sur ce sujet ont montré deux grandes approches pouvant être utilisées : l'échange direct entre deux applications et l'échange indirect basé sur l'utilisation d'un format neutre. La deuxième approche constitue notre centre d'intérêt à cause des limitations posées par la première approche, notamment pour la maintenance des systèmes impliqués dans l'échange. Pourtant, la deuxième approche a un risque significatif de perte de sémantique associée aux données échangées. En effet, la plupart des méthodes d'échange de données existantes sont basées sur le transfert des données géométriques, mais ne sont pas adaptées pour prendre en considération les informations annexes du produit, notamment les features. Dans ce contexte, nous avons réalisé une étude sur différents standards servant de format neutre entre les différentes applications de développement d'un produit, tels que « IGES » et « STEP ». Ensuite, nous avons réalisé une étude sur les différentes parties du STEP qui sont liées aux features, notamment les parties 48, 55, 108, 111, et 224, dont certaines sont en cours de développement afin de permettre l'échange de l'intention de conception.

Face à certaines limites des standards actuels, notamment au niveau de la prise en compte de la sémantique associée aux données d'un produit, nous avons proposé, dans le 3ème chapitre, une méthode d'échange entre des applications CAO en se basant sur l'échange de la sémantique de leurs contenus. Un aspect fondamental dans l'échange de données est lié aux techniques de représentations de ces données ayant la possibilité de caractériser le contexte dans lequel ces informations sont considérées. Les ontologies peuvent être adaptées pour représenter les points de vue car elles nous permettent de prendre en considération le contexte dans lequel une donnée est utilisée. Ainsi, nous avons proposé une méthode d'échange de la sémantique représentée explicitement par des ontologies. Il s'agit plus particulièrement de déterminer les correspondances sémantiques entre les différentes entités représentées par des

descriptions à un niveau plus élevé des informations du produit, autrement dit par le traitement des features tels que les poches et les trous plutôt que des entités de bas niveau telles que les éléments géométriques ou topologiques. Ainsi, les technologies du Web Sémantique, telles que les ontologies OWL et le langage des règles SWRL, sont utilisées dans notre méthode. Celle-ci consiste principalement en deux grandes étapes. Premièrement, l'homogénéisation des formats de représentation des modèles CAO vers un format pivot, en l'occurrence une ontologie en OWL DL, sert à traiter les hétérogénéités syntaxiques entre les formats des modèles CAO. Deuxièmement, définir des règles permettant la mise en correspondance entre les ontologies d'applications de CAO et notre format neutre, soit l'ontologie commune de features de conception, appelée CDFO. Cette ontologie est la base de la représentation neutre qui servira d'intermédiaire entre les applications CAO qui doivent échanger des informations.

Ainsi, Nous avons élaboré, dans le 4ème chapitre, une méthode générale pour construire notre ontologie commune de façon la plus générale possible afin qu'elle soit utilisable non seulement dans le cadre de l'échange de données CAO, mais aussi dans d'autres domaines liés au processus de développement de produit. Nous avons restreint l'implémentation de cette ontologie aux features de conception volumique ayant une représentation géométrique et contenant d'autres informations d'ingénierie définissant certaines caractéristiques d'usinage par exemple. Nous avons présenté, dans un premier temps, une représentation graphique de notre ontologie, puis une représentation formelle avec le langage OWL DL. Cette représentation formelle permet l'interprétation de la signification associée aux concepts par la machine. Elle permet également de faire appel aux processus d'automatisation de l'interopérabilité sémantique grâce au raisonnement fournis par les moteurs d'inférence.

Cette ontologie que nous avons conçue constitue la base de notre prototype pour l'échange de données car elle correspond à notre représentation commune. Cela signifie que nous allons instancier les concepts présents dans cette ontologie et sauvegarder ces instances dans un fichier qui va correspondre à ce que nous appelons notre fichier neutre. Comme cette ontologie contient des informations qui sont les plus consensuelles possibles, les règles de traduction que nous devons élaborer sont plus faciles à déterminer. Un facteur clé de notre ontologie est son extensibilité, où de nouveaux concepts et de nouvelles relations peuvent être ajoutées, à condition de ne pas générer des conflits.

Ensuite, nous avons présenté, dans le 5ème chapitre, une méthode d'intégration des ontologies par la création des règles de correspondance. Cette étape consiste à développer des outils d'intégration des ontologies OWL par alignement des entités de différentes ontologies. Cette méthode d'intégration se base principalement, d'une part, sur la définition des axiomes et des règles de correspondance et d'autre part sur la reconnaissance automatique des correspondances sémantiques par des inférences effectuées grâce aux raisonneurs basés sur les logiques de description. Dans ce cadre, nous avons présenté des procédures de reconnaissance des correspondances sémantiques. Le manque de correspondances pour certaines entités dans l'ontologie destinataire a nécessité le développement d'une méthode de calcul de similarité qui repose sur les différents composants de la description d'une entité.

Cette méthode de mesure de similarité, décrite dans le 6ème chapitre, représente des correspondances approximatives entre les entités de différentes ontologies. Cette méthode est adaptée au langage OWL DL afin d'exploiter l'expressivité du langage pour le calcul de similarité. Nous avons défini une fonction globale associée à chaque type d'entité, telle qu'une classe ou une propriété, ainsi que des fonctions locales définies selon les constructeurs du langage OWL. Les fonctions locales sont définies en termes d'attributs communs et distinctifs des composants de chaque entité. Enfin, nous avons développé, dans le 7ème chapitre, une application visant à faciliter l'intégration des deux ontologies de CAO en se basant sur les inférences des correspondances réalisées par les moteurs d'inférence. Le prototype que nous avons élaboré est implémenté à l'aide du langage Java.

Contributions

Nous avons réalisé quelques contributions aussi bien au niveau scientifique qu'au niveau d'implémentation :

- D'abord, nous avons proposé une méthode d'échange de données entre différents domaines en se basant sur deux étapes : l'homogénéisation syntaxique des formats de modèles CAO par une conversion vers le langage OWL DL, et l'interopérabilité au niveau sémantique réalisée par la mise en correspondance des entités de différentes ontologies en se basant sur leur description sémantique. Cette mise en correspondance exploite les technologies du Web Sémantique, telles que l'expressivité du langage OWL DL et la définition des règles avec le langage SWRL. Cette mise en correspondance exploite également les capacités de raisonnement fournies par les moteurs d'inférence basés sur les logiques de

description, tels que Pellet, ainsi que les raisonnements à base de règles, tels que JESS.

- Notre méthode d'intégration est ensuite enrichie par une méthode de calcul de similarité basée sur la description des entités en question. Le but étant d'assister les concepteurs en leur proposant des entités approximativement correspondantes afin de favoriser l'interopérabilité sémantique dans un environnement collaboratif. Cette méthode proposée est appropriée au langage OWL DL et se base sur la définition des fonctions globales et locales de similarité. Les fonctions globales sont définies en fonction des catégories d'entités, par exemple une classe ou une propriété, tandis que les fonctions locales sont définies en termes de constructeurs OWL en se basant sur l'approche des attributs communs et distinctifs entre les composants des entités.
- Nous avons également développé une ontologie commune de conception, à un niveau assez générique, afin de servir de format neutre entre les applications de CAO. Cette ontologie définie en langage OWL DL est indépendante de toute application de CAO. De plus, elle est extensible afin d'ajouter d'autres concepts définis dans de nouvelles applications.

Perspectives

Le travail réalisé dans cette thèse a été effectué dans un cadre précis où plusieurs extensions et améliorations peuvent être envisagées comme suit :

- ***Extension des types de features*** : Nous avons considéré dans notre travail un sous ensemble de features, notamment ceux liés à la conception volumique des pièces en 3D, par exemple les extrusions, les révolutions, les trous, les poches, les congés, les chanfreins, etc. Il est également nécessaire de pouvoir tenir compte d'autres types de features qui constituent un intérêt pour certains concepteurs, par exemple les features définis par l'utilisateur, appelé UDF (*User Defined Feature*). D'autres features fréquemment utilisés peuvent être également considérés tels que les features surfaciques, les features de tôlerie, etc. La prise en compte de ces autres types de features est essentielle pour que notre prototype puisse être suffisamment exhaustif. Cela nécessite sans doute de mettre à jour notre ontologie commune de features.
- ***Considération des assemblages*** : Notre prototype doit prendre en charge des pièces plus complexes que celles que nous avons considérées. Par exemple, il est

nécessaire de considérer les features d'assemblage qui définissent des contraintes de jointure entre plusieurs instances de pièces définissant ainsi la manière dont les pièces sont assemblées les unes avec les autres.

- ***Extension à d'autres domaines*** : Certains concepts peuvent également être définis dans d'autres domaines, par exemple la fabrication. En fait, l'objectif de notre thèse était de développer une méthode d'échange qu'on a appliquée dans le cadre de l'échange de données entre les applications de CAO. Il est donc nécessaire de considérer les features définis dans les autres phases du développement de produit afin que notre méthode soit potentiellement applicable à ces phases, par exemple la fabrication.
- ***Développement des ontologies d'application*** : Dans le cadre d'implémentation de notre prototype, nous avons développé des ontologies associées à deux logiciels de CAO, CatiaV5 et SolidWorks. Ce travail a consisté à extraire des informations de ces deux logiciels. Ce travail d'expertise était nécessaire puisque nous ne disposons pas d'ontologie explicite définie pour chaque application, mais les informations étaient représentées implicitement dans le logiciel ou dans la documentation. Ainsi, pour ajouter une application dans le domaine collaboratif, il faut définir son ontologie explicite en OWL DL, puis établir les règles de mise en correspondance entre cette ontologie d'application et notre format neutre, et dans les deux sens.
- ***Réutilisation des données*** : L'ontologie commune pourra servir au stockage des données d'un produit. L'avantage réside dans la possibilité d'exploiter cette ontologie dans la recherche sémantique des informations d'une base de modèles de produits. Par exemple, il est possible de créer des requêtes sémantiques avec le langage SQWRL (*Semantic Query-Enhanced Web Rule Language*) afin de réutiliser des données des produits déjà conçus.
- ***Méthode de similarité*** : Nous pouvons également enrichir notre méthode de calcul de similarité par des algorithmes de comparaison de chaînes de caractères. Il s'agit par exemple de considérer les annotations et les commentaires associés aux concepts définis en langage naturel afin de renforcer la recherche de similarités en se basant sur des ressemblances syntaxiques. Ce qui pourrait apporter des correspondances supplémentaires pour définir des axiomes d'ancrage. De plus, des expérimentations empiriques doivent être effectuées afin de mesurer les métriques d'évaluation des résultats obtenus, par exemple pour calculer la *précision*, le

rappel, et le *Fallout* , etc. Ces métriques, largement exploitées pour évaluer la qualité des alignements obtenus, serviront à déterminer les paramètres définis dans notre algorithme de calcul, tels que les poids et le seuil d'acception, d'une manière plus réaliste.

Bibliographie

Abdul Ghafour Samer [et al.] A Common Design-Features Ontology for Product Data Semantics Interoperability [Conférence] // The 2007 IEEE/WIC/ACM International Conference on Web . - 2007.

Abdul Ghafour Samer [et al.] An Ontology-based Approach for Procedural CAD Models Data Exchange [Conférence] // 3th ISPE INTERNATIONAL CONFERENCE ON CONCURRENT ENGINEERING: RESEARCH AND APPLICATIONS, Leading the Web in Concurrent Engineering (CE2006). - Sophia Antipolis , France : [s.n.], 2006.

Abdul Ghafour Samer [et al.] Towards an Intelligent CAD Models Sharing Based on Semantic Web Technologies [Conference] // 15th ISPE International Conference on Concurrent Engineering. - Belfast : Springer, 2008 - b.

Abdul Ghafour Samer [et al.] Using the Semantic Web for the Integration of Feature-based CAD Models Information [Conférence] // 2008 International Conference on Industrial Informatics and Systems Engineering (IISE 2008). - Glasgow - Scotland : System and Information Sciences Notes - Communications of SIWN, 2008. - 1757-4439.

Abdul Ghafour Samer Méthodes et Outils pour l'intégration des Ontologies [Rapport] : techreport. - Lyon : LIRIS - Université Claude Bernard Lyon1, 2004.

Abels Sven, Haak Liane et Hahn Axel Identifying ontology integration methods and their applicability in the context of product classification and knowledge integration tasks [Rapport] : techreport. - 2005.

Alesso H P et Smith C F Developing semantic web services [Livre]. - [s.l.] : AK Peters, Ltd., 2005.

Allada V et Anand S Feature-based modelling approaches for integrated manufacturing: state-of-the-art survey and future research directions [Revue] // International Journal of Computer Integrated Manufacturing. - [s.l.] : Taylor & Francis, 1995. - Vol. 8. - pp. 411--440.

Allada V Feature based Modelling for Concurrent Engineering [Revue] // PhD Thesis. - [s.l.] : University of Cincinnati, Cincinnati, OH 45221, USA, 1994.

Anderson B Implementor's Guide, Solid Model Construction History [Revue] // PDES, Inc., April. - 2001.

Anderson Bill and Ansaldi S ENGEN Data Model: a neutral model to capture design intent [Article] // Proceedings of the Tenth International IFIP WG 5.2/5.3 Conference. - Trento, Italy : [s.n.], 1998.

Anderson DC et Chang TC Automated process planning using object-oriented feature based design [Conférence]. - 1990. - p. 247.

Angele J et Lausen G Ontologies in F-logic [Revue] // Handbook on Ontologies. - [s.l.] : Springer, 2004. - pp. 29--50.

Autodesk Autodesk Mechanical Desktop 4.0 User's Guide. - Autodesk, Inc., San Rafael, CA, USA : [s.n.], 1999.

Baader F [et al.] Basic description logics [Revue] // Journal of Logic and Computation. - 2003.

Baader F [et al.] The description logic handbook [Livre]. - [s.l.] : Cambridge University Press New York, NY, USA, 2007.

Barkmeyer E [et al.] Concepts for automating systems integration [Livre]. - Gaithersburg : US Dept. of Commerce, Technology Administration, National Institute of Standards and Technology (NIST), 2003.

Baumann R A Structure Oriented Exchange of Product Model Data [Livre]. - [s.l.] : Fraunhofer-IRB-Verl., 2004.

Bechhofer S [et al.] OWL web ontology language reference [Revue] // W3C recommendation. - 2004. - Vol. 10. - pp. 2006--01.

Bechhofer S, Volz R et Lord P Cooking the Semantic Web with the OWL API [Revue] // Lecture Notes in Computer Science. - [s.l.] : Springer, 2003. - pp. 659--675.

Benjamin PC [et al.] Idef5 method report [Revue] // Knowledge Based Systems, Inc. - 1994.

Berner-Lee T, Hendler J et Lassila O The Semantic Web [Revue] // Scientific American. - 2001. - Vol. 284. - pp. 34--43.

Bianconi F, Conti P et Di Angelo L Interoperability among CAD/CAM/CAE systems: a review of current research trends [Revue] // Geometric Modeling and Imaging--New Trends, 2006. - 2006. - pp. 82--89.

Bidarra R et Bronsvoort WF Semantic feature modelling [Revue] // Computer Aided Design. - 2000. - Vol. 32. - pp. 201--25.

Bidarra R Validity maintenance in semantic feature modeling [Revue] // PhD Thesis. - [s.l.] : Delft University of Technology. The Netherlands, 1999.

Billo RE, Henderson M et Rucker R Applying conceptual graph inferencing to feature-based engineering analysis [Revue] // Computers in Industry. - [s.l.] : Elsevier Science Publishers BV Amsterdam, The Netherlands, 1989. - Vol. 13. - pp. 195--214.

Bohring H et Auer S Mapping xml to owl ontologies [Revue] // Leipziger Informatik-Tage. - [s.l.] : GI, 2005. - Vol. 72. - pp. 147--156.

Borgida A On the relative expressiveness of description logics and predicate logics [Revue] // Artificial Intelligence. - [s.l.] : Elsevier, 1996. - Vol. 82. - pp. 353--367.

Bouquet P [et al.] Specification of a common framework for characterizing alignment. deliverable D2.2.1 [Revue] // Knowledge web NoE. - 2005.

Brachman R J et Schmolze J G An overview of the KL-ONE knowledge representation system [Revue] // Cognitive science. - [s.l.] : Elsevier, 1985. - Vol. 9. - pp. 171--216.

Brandt S, Kusters R et Turhan A Y Approximation and difference in description logics [Revue] // Proc. of KR. - 2002. - pp. 203--214.

Bray T [et al.] Extensible markup language (XML) 1.0 second edition W3C recommendation [Revue] // WorldWide Web Consortium. - 2000.

Brickley D et Guha R V Rdf vocabulary description language 1.0: Rdf schema [Revue] // W3C Working Draft. - 2003. - Vol. 23.

Broekstra J [et al.] Enabling knowledge representation on the web by extending RDF schema [Revue] // Computer networks. - [s.l.] : Elsevier, 2002. - Vol. 39. - pp. 609--634.

Bronsvoort W F [et al.] Multiple-view feature modelling and conversion [Revue] // Geometric Modeling: Theory and Practice-The State of the Art, Springer-Verlag. - 1997. - pp. 159--174.

Butterfield WR [et al.] Part features for process planning [Revue] // CAM-I report R-85-PPP-03. - 1986.

Case K, Gao X. J. et Gindy N. N. Z The implementation of a feature-based component representation for CAD/CAM integration [Revue] // Proceedings of the Institution of Mechanical Engineers. Part B. Journal of engineering manufacture ISSN 0954-4054. - 1994. - Vol. 208. - pp. 71-80,.

Choi G H, Mun D et Han S Exchange of CAD part models based on the macro-parametric approach [Revue] // International Journal of CAD/CAM. - 2002. - Vol. 2. - pp. 23--31.

Connolly D [et al.] DAML+ OIL (March 2001) reference description [Revue] // W3c note. - 2001. - Vol. 18.

Daconta M C, Obrst L J et Smith K T The Semantic Web: a guide to the future of XML, Web services, and knowledge management [Livre]. - [s.l.] : Wiley, 2003.

d'Amato C, Fanizzi N et Esposito F A dissimilarity measure for ALC concept descriptions [Conférence] // Symposium on Applied Computing: Proceedings of the 2006 ACM symposium on Applied computing. - Dijon - FRANCE : [s.n.], 2006.

Daniel M et Lucas M Towards declarative geometric modeling in mechanics [Conférence]. - 1997. - Vol. 96. - pp. 15--17.

Dartigues Christel [et al.] CAD/CAPP Integration using Feature Ontology [Revue] // Concurrent Engineering: Research and Applications. - [s.l.] : SAGE publications, jun 2007. - Vol. 15. - pp. 237--249. - 1063-293X.

Dartigues Christel Product data exchange in a collaborative environment [Rapport] : techreport. - 2003.

De Kraker K. J. Feature conversion for concurrent engineering [Livre]. - [s.l.] : PhD Thesis, Delft University of Technology, 1998.

Dixon J R, Cunningham J J et Simmons M K Research in designing with features [Conférence]. - 1989. - p. 137.

Djuric D MDA-based ontology infrastructure [Revue] // Computer Science and Information Systems (ComSIS). - 2004. - Vol. 1. - pp. 91--116.

Drummond N [et al.] Sequences in OWL [Conférence]. - 2006.

Dunn M PDES Form Feature Information Model (FFIM) [Revue] // PDES Form Features Group. - 1988.

Ehrig M et Sure Y Ontology mapping-an integrated approach [Revue] // Lecture Notes in Computer Science. - [s.l.] : Springer, 2004. - pp. 76--91.

Euzenat J [et al.] Ontology alignment with OLA [Conférence]. - 2004-b.

Euzenat J [et al.] State of the art on ontology alignment [Revue] // Knowledge Web Deliverable. - 2004. - Vol. 2.

Farquhar A, Fikes R et Rice J The ontolingua server: A tool for collaborative ontology construction [Revue] // International Journal of Human-Computers Studies. - 1997. - Vol. 46. - pp. 707--727.

Feng CX [et al.] Representation of functions and features in detail design [Revue] // Computer Aided Design. - 1996. - Vol. 28. - pp. 961--71.

Fensel D [et al.] Semantic web application areas [Conférence]. - 2002.

Fernandez-Lopez M et Gomez-Perez A Overview and analysis of methodologies for building ontologies [Revue] // The Knowledge Engineering Review. - [s.l.] : Cambridge Univ Press, 2003. - Vol. 17. - pp. 129--156.

Ferreira da Silva Catarina [et al.] Semantic Interoperability of Heterogeneous Semantic Resources [Revue] // Electronic Notes in Theoretical Computer Science. - [s.l.] : Elsevier, mar 2006. - Vol. 150. - pp. 71--85.

Ferreira Da Silva Catarina Découverte de correspondances sémantiques entre ressources hétérogènes dans un environnement coopératif [Rapport] : techreport. - Lyon : Université Claude Bernard Lyon 1, 2007.

Fontana M, Giannini F et Meirana G M A free form feature taxonomy [Article] // Computer Graphics Forum. - 1999. - 3, p.107-118 : Vol. 18.

Friedman-Hill E JESS 7.0 p1: The rule engine for the Java platform. - 2007.

Fu M. W., Ong Soh-Khim and Lu W. F. An approach to identify design and manufacturing features from a data exchanged part model [Journal] // Computer-Aided Design. - [s.l.] : Computer-Aided Design, 2003. - 11 : Vol. 35. - pp. 979-993.

Fu Z et De Pennington A Geometric reasoning based on graph grammar parsing [Revue] // Journal of Mechanical Design. - 1994. - Vol. 116. - p. 763.

Genesereth M R et Fikes R E Knowledge Interchange Format (KIF) Version 3.0 Reference Manual, Logic-92-1 [Revue] // Computer Science Department, Stanford University. - 1992.

Ghodous Parisa Modélisation Intégrée de Données de Produit et de Processus de Conception [Rapport] : techreport. - Lyon - FRANCE : Université Claude Bernard - Lyon1, 1996.

Ghodous Parisa Product and process modeling in a cooperative environment [Journal] // Int. Journal of Internet and Enterprise Management. - 2003. - p. Vol. 1.

Gil R, Garcia R et Delgado J An interoperable framework for Intellectual Property Rights using web ontologies [Conférence]. - 2005.

Gindy NNZ A hierarchical structure for form features [Revue] // International Journal of Production Research. - [s.l.] : Taylor & Francis, 1989. - Vol. 27. - pp. 2089--2103.

Goldstone R L et Son J Similarity [Revue] // Cambridge handbook of thinking and reasoning. - [s.l.] : Cambridge University Press, 2005. - pp. 13--36.

Gruber T R A translation approach to portable ontology specifications [Revue] // Knowledge acquisition. - [s.l.] : Academic Press, 1993. - Vol. 5. - pp. 199--199.

Gruninger M et Fox M S Methodology for the Design and Evaluation of Ontologies [Conférence]. - 1995.

Gruninger M et Menzel C The process specification language (PSL) theory and applications [Revue] // AI magazine. - 2003. - Vol. 24. - p. 63.

Guarino N Formal Ontology in Information Systems.. - [s.l.] : IOS press Amsterdam, 1998.

Han Jung Hyun Survey of Feature Research [Report] : Technical Report IRIS-96-346. - [s.l.] : University of Southern California Institute for Robotics and Intelligent Systems, 1996.

Heflin J et Hendler J Semantic interoperability on the web [Livre]. - [s.l.] : Defense Technical Information Center, 2000.

Holger K [et al.] The Protégé OWL Plugin: an open development environment for semantic web applications [Conférence] // Third International Semantic Web Conference-ISWC. - 2004.

Homann C M et Juan R Erep, an editable, high-level representation for geometric design and analysis [Revue] // Geometric and Product Modeling. - pp. 129--164.

Horrocks I [et al.] OWL rules: A proposal and prototype implementation [Revue] // Web Semantics: Science, Services and Agents on the World Wide Web. - [s.l.] : Elsevier, 2005. - Vol. 3. - pp. 23--40.

Horrocks I [et al.] SWRL: A semantic web rule language combining OWL and RuleML [Revue] // W3C Member Submission. - 2004. - Vol. 21.

Horrocks I, Patel-Schneider P F et Van Harmelen F From SHIQ and RDF to OWL: The making of a web ontology language [Revue] // Web semantics: science, services and agents on the World Wide Web. - [s.l.] : Elsevier, 2003. - Vol. 1. - pp. 7--26.

Horváth I [et al.] Towards an ontology-based definition of design features [Conférence] // SIAM workshop on mathematical foundations for features in computer aided design, engineering, and manufacturing. - 1998.

IDEAS SDRC. I-DEAS Master Series 7 Use's Guide. - 1999.

ISO 10303-108 Industrial automation systems and integration- Product data representation and exchange part 108: integrated application resource: parameterization and constraints for explicit geometric product models, ISO 10303-108 [Revue] // International Organisation for Standardization. - 2005.

ISO 10303-11 ISO 10303 Industrial Automation Systems--Product Data Representation and Exchange--Part 11: Description Methods: The Express Language Reference Manual [Revue] // International Organisation for Standardization. - 1992.

ISO 10303-111 Industrial automation systems and integration - product data representation and exchange - Part 111: integrated application resource: construction history features, ISO CD 10303-111. - [s.l.] : International Organisation for Standardization, 2003.

ISO 10303-21 Industrial automation systems and integration - Product data representation and exchange- Part 21: Implementation methods: Clear text encoding of the exchange structure, ISO. - [s.l.] : International Organisation for Standardization, 1994.

ISO 10303-224 Industrial automation systems and integration: product data representation and exchange. Part 224. Application protocol: mechanical product definition for process planning using machining features; ISO. - [s.l.] : International Organisation for Standardization, 1999.

ISO 10303-42 Industrial automation systems and integration - Product data representation and exchange-Part 42: Integrated generic resources: Geometric and topological representation, ISO [Revue] // International Organisation for Standardization. - [s.l.] : International Organisation for Standardization, 1994.

ISO 10303-48 STEP Product Data Representation and Exchange. Integrated Generic Ressources: Form Feature [Revue] // Tecnical, International Organisation for Standardization, Subcommittee 4,Part 48, NIST. - [s.l.] : Tecnical, International Organisation for Standardization, Subcommittee 4,Part 48, NIST, 1992.

ISO 10303-55 Industrial automation systems and integration - product data representation and exchange-part 55: integrated generic resource: procedural and hybrid representation, ISO 10303-55. - Geneva, Switzerland : International Organisation for Standardization, 2005.

Janowicz K [et al.] Algorithm, implementation and application of the SIM-DL Similarity Server [Revue] // Lecture Notes in Computer Science. - University of Muenster, Germany : Springer, 2007. - Vol. 4853. - p. 128.

Janowicz K Sim-dl: Towards a semantic similarity measurement theory for the description logic ALCNR in geographic information retrieval [Revue] // LECTURE NOTES IN COMPUTER SCIENCE. - [s.l.] : Springer, 2006. - Vol. 4278. - p. 1681.

Kalfoglou Y et Schorlemmer M Ontology mapping: the state of the art [Revue] // The knowledge engineering review. - [s.l.] : Cambridge Univ Press, 2003. - Vol. 18. - pp. 1--31.

Kim K Y, Manley D G et Yang H Ontology-based assembly design and information sharing for collaborative product development [Revue] // Computer-Aided Design. - [s.l.] : Elsevier, 2006. - Vol. 38. - pp. 1233--1250.

Kim K.Y., Chae S.H. et Suh H.W. An Approach to Semantic Mapping using Product Ontology for CPC Environment [Conférence] // CE Research and Applications: The vision for the future generation. - 2003. - pp. 291--298.

Kitamura Y [et al.] Deployment of an ontological framework of functional design knowledge [Revue] // Advanced Engineering Informatics. - [s.l.] : Elsevier, 2004. - Vol. 18. - pp. 115--127.

Kripac J A mechanism for persistently naming topological entities in history-based parametric solid models [Conférence]. - 1995. - pp. 21--30.

Lacot Xavier Introduction à OWL, un langage XML dontologies Web [Rapport] : techreport. - 2005.

Lee YS, Cheon SU et Han SH Enhancement of CAD model interoperability based on feature ontology [Conférence]. - 2003. - Vol. 10. - pp. 251--271.

Lutters D, Streppel AH et Kals HJJ The role of information structures in design and engineering processes [Revue] // Delft University of Technology. - [s.l.] : Delft University of Technology, 1997.

Maculet R et Daniel M Conception, modélisation géométrique et contraintes en CAO: Une synthèse [Revue] // Rapport de recherche CNRS RR-LSIS-2003-005, Marseille. - [s.l.] : Rapport de recherche CNRS RR-LSIS-2003-005, Marseille, 2003.

Maedche A et Staab S Measuring similarity between ontologies [Revue] // Lecture notes in computer science. - [s.l.] : Springer, 2002. - pp. 251--263.

Magleby SP et Jackson DB A Standardized Application Interface for Geometric Modelers [Revue] // Product Modeling for Computer-Aided Design and Manufacturing, North-Holland. - 1991.

Manley David G Assembly Differentiation in CAD Systems [Rapport] : techreport. - 2006.

Markman A B Structural alignment, similarity, and the internal structure of category representations [Revue] // Similarity and categorization. - 2001. - pp. 109--130.

McBride B Jena: A semantic web toolkit [Revue] // IEEE Internet Computing. - 2002. - Vol. 6. - pp. 55--59.

McGuinness D L et Van Harmelen F Owl web ontology language overview [Revue] // W3C recommendation. - [s.l.] : W3C Recommendation - 10 February 2004, 2004. - Vol. 10. - pp. 2004--03.

Medin D L, Goldstone R L et Gentner D Respects for similarity. Psychological Review. [Revue] // PSYCHOLOGICAL REVIEW-NEW YORK-. - [s.l.] : APA AMERICAN PSYCHOLOGICAL ASSOCIATION, 1993. - Vol. 100. - pp. 254--254.

Mun D [et al.] A set of standard modeling commands for the history-based parametric approach [Revue] // Computer-aided design. - [s.l.] : Elsevier, 2003. - Vol. 35. - pp. 1171--1179.

Napoli A Une introduction aux logiques de descriptions [Revue] // Cours de la Logique de Description. - 2005.

Nielsen E H, Dixon J R et Zinsmeister G E Capturing and using designer intent in a design-with-features system [Conférence]. - 1991. - p. 95.

Nosofsky RM Similarity scaling and cognitive process models [Revue] // Annual Review of Psychology. - [s.l.] : Annual Reviews, 1992. - Vol. 43. - pp. 25--53.

Noy N F et Musen M A An algorithm for merging and aligning ontologies: automation and tool support [Conférence]. - 1999. - pp. 1999--0799.

O'Connor M [et al.] Writing rules for the semantic web using SWRL and Jess [Revue] // ProtÃ©gÃ© With Rules WS, Madrid. - Madrid, Spain : 8th International Protégé Conference, Protégé with Rules Workshop, 2005 - b.

O'Connor M.J. [et al.] Supporting rule system interoperability on the semantic web with SWRL [Revue] // Lecture notes in computer science. - Fourth International Semantic Web Conference (Galway) : Springer, 2005. - Vol. 3729. - p. 974.

Pahl G et Beitz W Engineering design: a systematic approach [Livre]. - [s.l.] : Springer, 1996.

Parsia B et Sirin E Pellet: An owl dl reasoner [Conférence]. - 2004. - Vol. 104.

Patel-Schneider P F et Horrocks I OWL 1.1 web ontology language overview [Revue] // W3C Member Submssion. - [s.l.] : http://www.webont.org/owl/1.1/overview.html, 2006. - Vol. 19.

Patil L M Interoperability of formal semantics of production data across product development systems [Rapport] : phdthesis. - [s.l.] : UNIVERSITY OF MICHIGAN, 2005.

Patil Lalit, Dutta Debasish et Sriram Ram D Ontology-based exchange of product data semantics [Revue] // IEEE transactions on automation science and engineering ISSN 1545-5955. - [s.l.] : IEEE transactions on automation science and engineering ISSN 1545-5955, 2005. - 3 : Vol. 2. - pp. 213--225,.

Peters B F MicroStation Modeler: the design and implementation of an extensible solid modeling system [Revue] // Geometric Modeling: Theory and Practice--The State of the Art. - 1997. - pp. 361--378.

Pinto H S et Martins J P A methodology for ontology integration [Conférence]. - 2001. - pp. 131--138.

Pinto H S, G'omez-P'erez A et Martins J P Some issues on ontology integration [Conférence]. - Stockholm Sweden : [s.n.], 1999.

Pratt M J et Anderson B D A shape modelling applications programming interface for the STEP standard [Article] // Computer Aided Design. - [s.l.] : Elsevier, 2001. - Vol. 33. - pp. 531--543.

Pratt M J et Bedford MK Aspects of form feature modelling [Revue] // Geometric modeling: methods and applications. - [s.l.] : Springer-Verlag, 1991. - p. 227.

Pratt M J et Ohtaka A Parametric Representation and Exchange: Preparatory Knowledge about history based parametric model - ISO TC184/SC4/WG12 N189. - [s.l.] : International Organisation for Standardization, 1998.

Pratt M J Introduction to ISO 10303 - the STEP standard for product data exchange [Revue] // Journal of Computing and Information Science in Engineering. - 2001 - b. - Vol. 1. - p. 102.

Pratt M J, Anderson B D et Ranger T Towards the standardized exchange of parameterized feature-based CAD models [Revue] // Computer-Aided Design. - [s.l.] : Elsevier, 2005. - Vol. 37. - pp. 1251--1265.

Pratt MJ Extension of the Standard ISO10303 (STEP) for the exchange of parametric and variational CAD models [Article] // CDROM Proceedings of the Tenth International IFIP WG 5.2/5.3 Conference PROLAMAT98. - 1998. - Vol. 5.

ProEngineer Pro/ENGINEER Modeling User's Guide, Version 20. - Parametric Technology Corporation, Waltham, MA, USA : [s.n.], 1999.

Rahm E et Bernstein P A A survey of approaches to automatic schema matching [Revue] // The VLDB Journal The International Journal on Very Large Data Bases. - [s.l.] : Springer, 2001. - Vol. 10. - pp. 334--350.

Salomons O W Computer Support in the Design of Mechanical Products: Constraint Specification and Satisfaction in Feature Based Design for Manufacturing [Livre]. - University of Twente : Enschede, 2000.

Salomons OW, Van Houten F et Kals HJJ Review of research in feature-based design. [Revue] // Journal of manufacturing systems. - 1993. - Vol. 12. - pp. 113--132.

Seo Tae-Sul [et al.] Sharing CAD models based on feature ontology of commands history [Revue] // International Journal of CAD/CAM. - 2005. - Vol. 5.

Shah Jami J et Mantyla Martti Parametric and Feature Based CAD/Cam: Concepts, Techniques, and Applications [Livre]. - New York, NY, USA : John Wiley & Sons, Inc., 1995.

Shah JJ et Mathew A Experimental investigation of STEP form-feature information model [Article] // Computer-Aided Design. - [s.l.] : Butterworth-Heinemann Newton, MA, USA, 1991. - 4 : Vol. 23. - pp. 282--296.

Shah JJ Philosophical development of form feature concept [Conférence]. - 1990.

Sirin E [et al.] Pellet: A practical owl-dl reasoner [Revue] // Web Semantics: Science, Services and Agents on the World Wide Web. - [s.l.] : Elsevier, 2007. - Vol. 5. - pp. 51--53.

Smith A E et Dagli C H Manufacturing feature identification for intelligent design [Article] // Intelligent Systems in Design and Manufacturing. - Boston, MA, USA : [s.n.], 1995. - 17-21. - pp. 213--230.

Smith B Initial graphics exchange specification (IGES), version 4.0 [Revue]. - [s.l.] : US Dept. of Commerce, National Bureau of Standards [Springfield, VA: Order from NTIS] Warrendale, PA: Society of Automotive Engineers Inc., Gaithersburg, MD, 1988.

Smith M K, Welty C et McGuinness D L Owl web ontology language guide [Revue] // W3C recommendation. - 2004. - Vol. 10.

Spitz S et Rappoport A Integrated feature-based and geometric CAD data exchange [Conférence]. - 2004. - pp. 183--190.

Stiteler M Construction History And ParametricS: Improving affordability through intelligent CAD data exchange [Revue] // CHAPS Program Final Report. - Advanced Technology Institute, 5300 International Boulevard, North Charleston, SC 29418, SC, USA : [s.n.], 2004.

Szykman S, Sriram R D et Regli W C The role of knowledge in next-generation product development systems [Revue] // Journal of computing and information Science in Engineering. - 2001. - Vol. 1. - p. 3.

Thibau J. P. Similarité et catégorisation [Revue] // LannÃ©e Psychologique. - Paris : Université René Descartes, Paris 5, 1997. - Vol. 97. - pp. 701--736.

Touzani M Alignement des ontologies OWL-Lite [Livre]. - [s.l.] : Université de Montréal, 2005.

Tversky A Features of similarity [Revue] // Psychological review. - 1977. - Vol. 84. - pp. 327--352.

Uschold M et Gruninger M Ontologies: Principles, methods and applications [Revue] // Knowledge engineering review. - 1996.

Vieira AS Consistency management in feature based parametric design [Conférence]. - 1995. - pp. 977--987.

Wache H [et al.] Ontology-based integration of information-a survey of existing approaches [Conférence]. - 2001. - Vol. 2001. - pp. 108--117.

Wang N et Ozsoy T M A scheme to represent features, dimensions, and tolerances in geometric modeling [Revue] // Journal of Manufacturing Systems. - 1991. - Vol. 10. - pp. 233--240.

Wang N et Ozsoy T M Representation of assemblies for automatic tolerance chain generation [Article] // Engineering with Computers. - [s.l.] : Springer, 1990. - 2 : Vol. 6. - pp. 121--126.

Wilkinson K [et al.] Efficient RDF storage and retrieval in Jena2 [Conférence]. - 2003. - Vol. 3. - pp. 7--8.

Wilson P R et Pratt MJ A taxonomy of form features for solid modeling [Article] // Geometric Modeling for CAD Applications.. - North Holland : Elsvier Science, 1988. - 125-135.

Wingard L. Introducing form features in product models: a step towards CADCAM with engineering terminology [Livre]. - Dept. Of manufacturing Systems, Royal Institute of Technology; Stockholm : Computer System for Design and Development, 1991.

Zghal S [et al.] Edola: Une nouvelle méthode d'alignement d'ontologies OWL-Lite [Revue] // Coria 2007. - [s.l.] : Université de Saint-Etienne, 2007 - b. - p. 351.

Zghal S [et al.] Soda: an OWL-DL based ontology matching system [Revue]. - 2007.

Zghal S [et al.] Soda: Une approche structurelle pour l'alignement d'ontologies OWL-DL [Conférence] // Proceedings of the first French Conference on Ontology (JFO 2007). - Sousse, Tunisia : [s.n.], 2007.

Annexes A

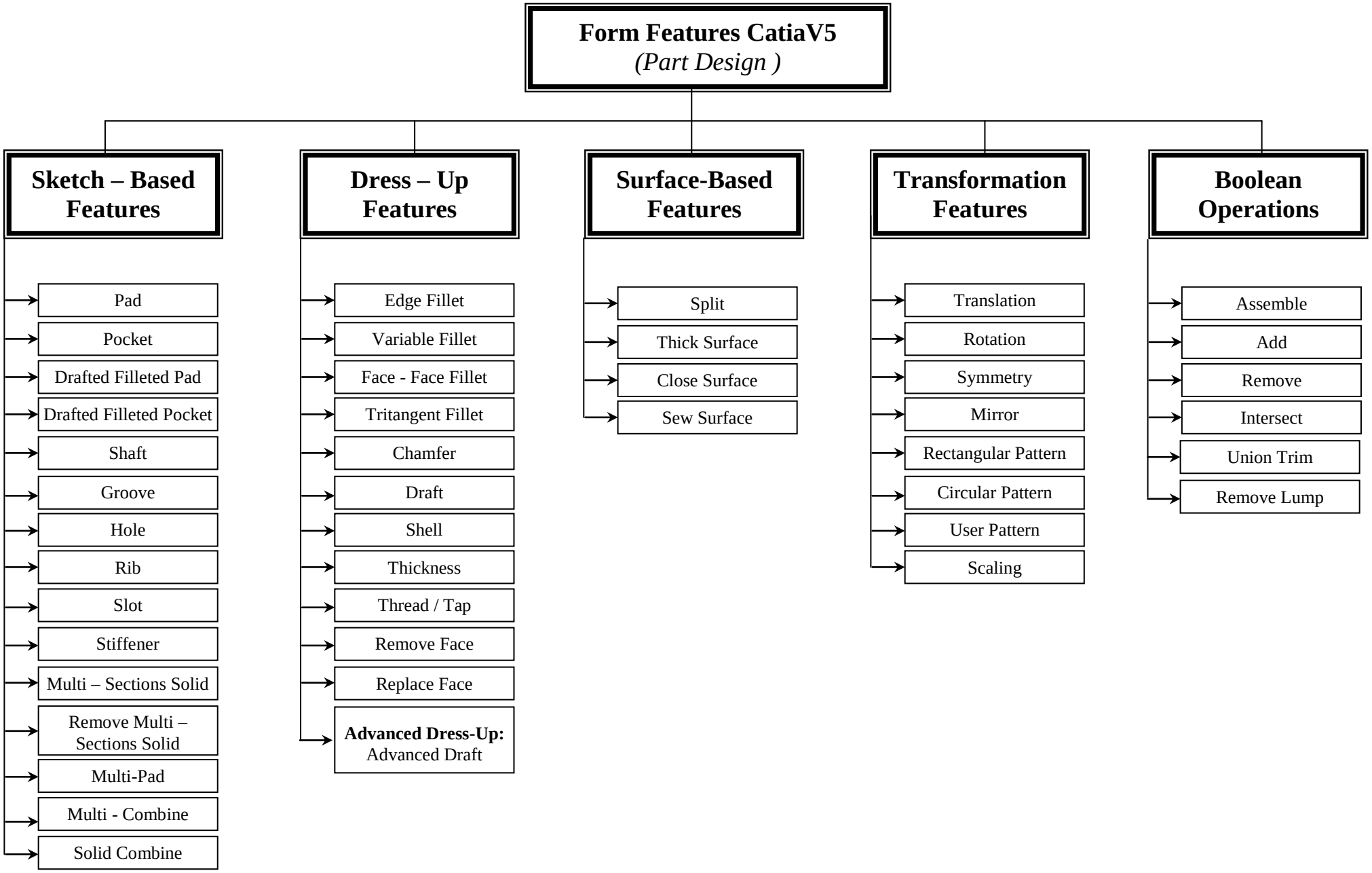


Figure 54. Représentation des features de forme dans CatiaV5

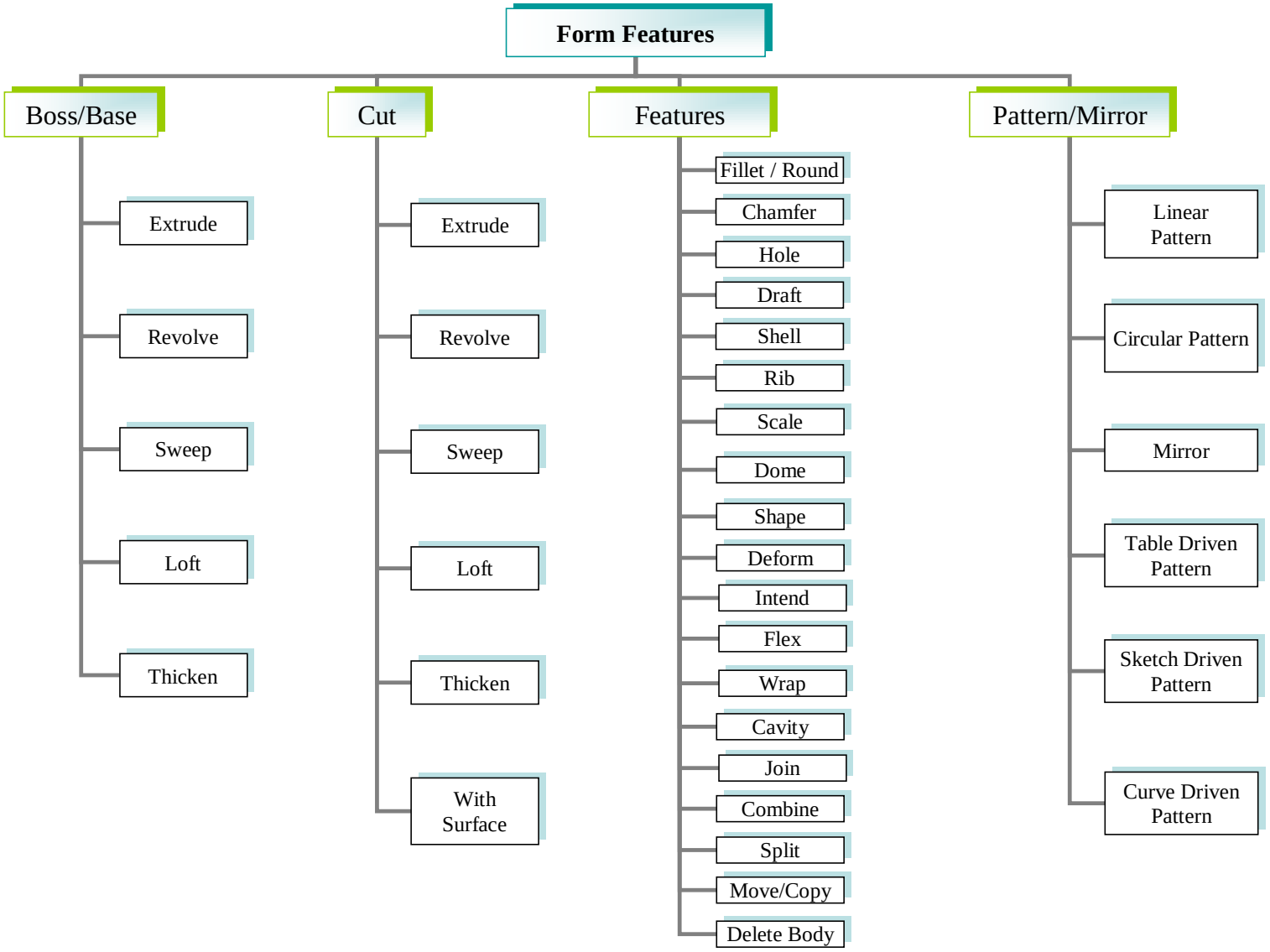


Figure 55. Représentation des features de forme dans SolidWorks

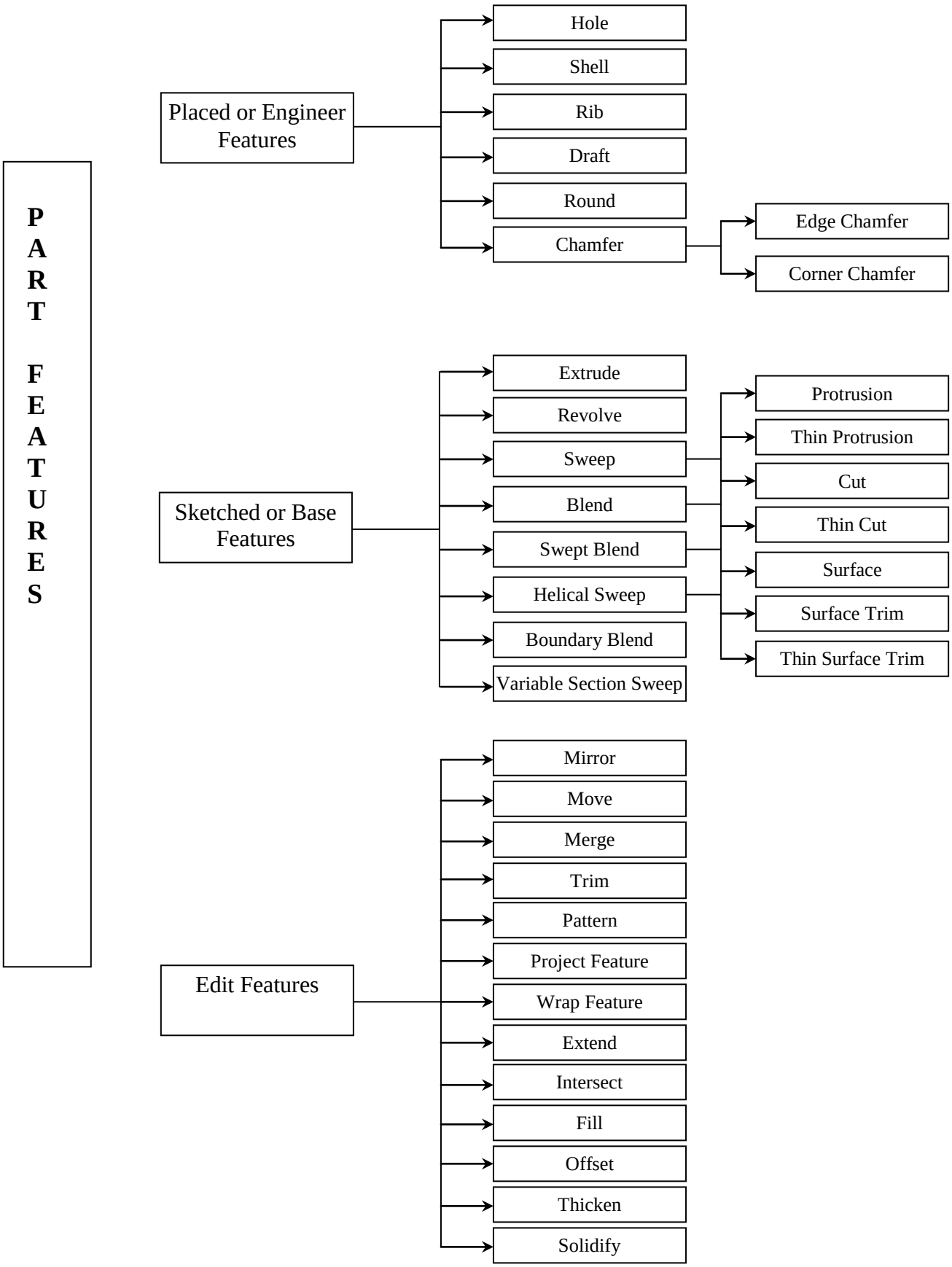


Figure 56. Représentation des features de forme dans Pro/Engineer

Annexes B

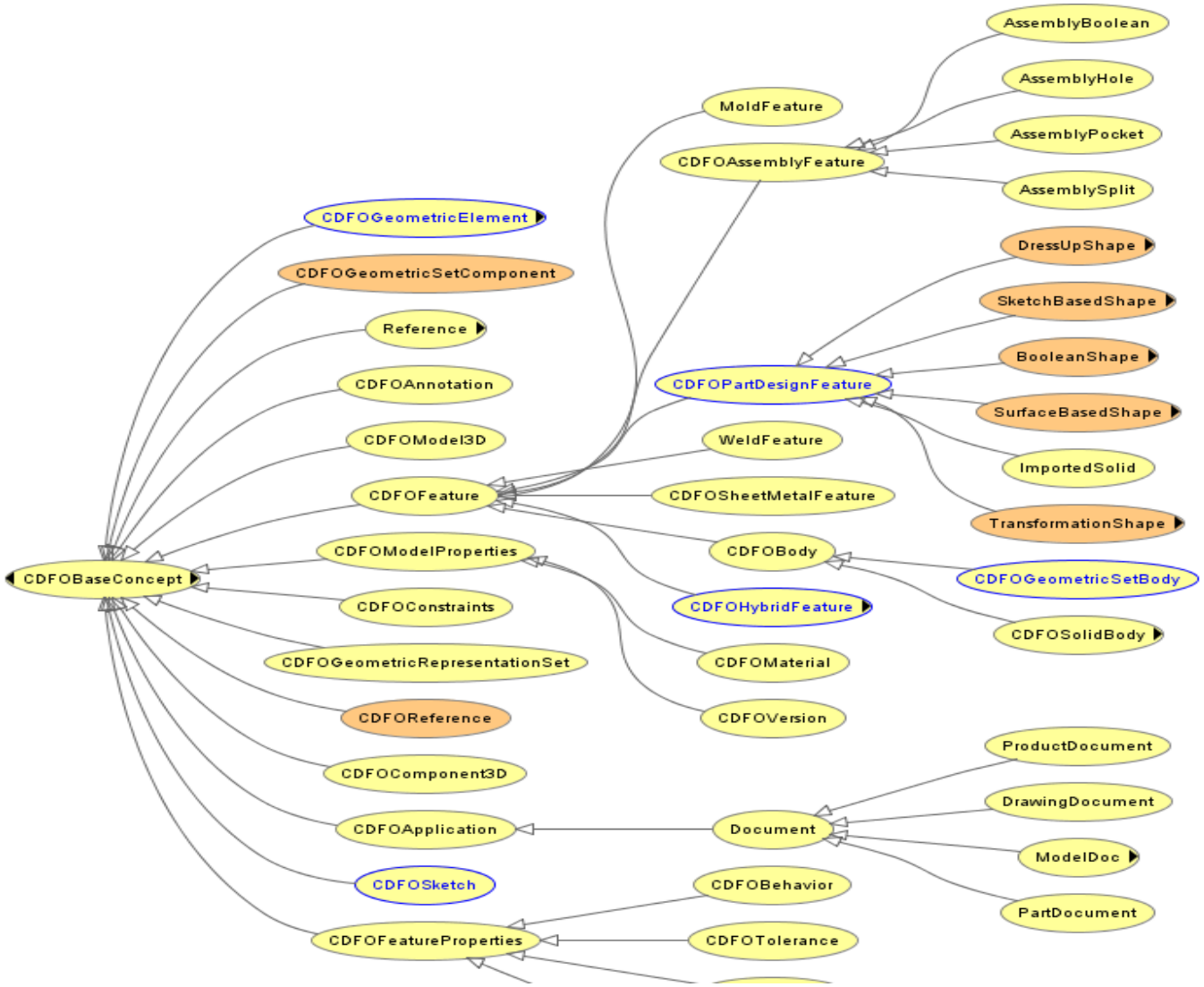


Figure 57. Une vue générique de l'ontologie commune CDFO avec Jambalaya de *Protégé*