HAL
archives-ouvertes.fr

# Structural and semantic comparison to synchronize Digital Mock-Up for Construction

Geoffrey Arthaud

▶ **To cite this version:**

Geoffrey Arthaud. Structural and semantic comparison to synchronize Digital Mock-Up for Construction. Mathematics [math]. Ecole des Ponts ParisTech, 2007. English. <pastel-00004854>

**HAL Id: pastel-00004854**

**https://pastel.archives-ouvertes.fr/pastel-00004854**

Submitted on 5 Mar 2009

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.


CSTB
le futur en construction

## ÉCOLE NATIONALE DES PONTS ET CHAUSSÉES

— INFORMATION, COMMUNICATION, MODÉLISATION ET SIMULATION —

# THÈSE

pour l'obtention du diplôme de
**Doctorat de l'École Nationale des Ponts et Chaussées**

Discipline : **Mathématiques, Informatique**

**Geoffrey ARTHAUD**

# APPORTS DE MODÈLES DE COMPARAISON STRUCTURELLE ET SÉMANTIQUE À LA SYNCHRONISATION DE LA MAQUETTE NUMÉRIQUE DE CONSTRUCTION – APPLICATION AUX PONTS –

***Thèse dirigée par* Jacques RILLING**

Soutenance publique le 13/12/2007 devant le jury composé de :

| | | |
|---|---|---|
| Gilles HALIN | Maître de conférence HDR en Informatique à l'Université Nancy 2 | Rapporteur |
| Pierre LECLERCQ | Professeur de l'Université de Liège 1, Belgique | Rapporteur |
| Jean-Armand CALGARO | Professeur de l'ENPC, Champs-sur-Marne | Examinateur |
| Nobuyoshi YABUKI | Professeur de l'Institut Technologique de Muroran, Japon | Examinateur |
| Ghassan AOUAD | Professeur de l'Université de Salford, Royaume Uni | Examinateur |
| Souheil SOUBRA | Docteur de l'ENPC, CSTB, Sophia-Antipolis | Examinateur |
| Jacques RILLING | Professeur de l'ENPC, Champs-sur-Marne | Directeur de thèse |

# Remerciements

Je tiens à remercier toutes les personnes qui m'ont aidé à mener à bien ce travail de trois ans. En particulier, je remercie chaleureusement mon directeur de thèse, Monsieur Jacques Rilling, pour ses précieux conseils et l'expérience qu'il a su me faire partager.

Je remercie également Monsieur Souheil Soubra, responsable de la division MOD-EVE du CSTB, pour son accueil, sa patience et la confiance qu'il a bien voulu m'accorder.

Je remercie vivement les rapporteurs Monsieur Gilles Halin et Monsieur Pierre Leclercq d'avoir accepté de relire mon manuscrit et d'établir un rapport, dont les commentaires m'ont aidé à terminer ce travail.

Je tiens aussi à remercier les examinateurs de ma thèse. En particulier, je suis très honoré que Monsieur Nobuyoshi Yabuki ait effectué le déplacement jusqu'en France et de m'avoir accueilli chaleureusement à deux reprises au Japon. Je remercie également Monsieur Ghassan Aouad d'avoir accepté d'assister à ma soutenance ainsi que Monsieur Jean-Armand Calgaro pour ses nombreux conseils et son sens du concret.

Je ne saurais trop remercier tous les membres de l'équipe MOD-EVE de Sophia-Antipolis, tant pour le partage de leur compétence que pour la très bonne ambiance générale qu'ils ont su créer : je remercie ainsi Florent, Mathieu, Jean-Christophe, Guillaume, Julien, Elisa, Eric, Sandra, ainsi qu'Audrey et David lors de leurs stages respectifs. Je remercie aussi Rémi Vankeisbelck et Guillaume Besse pour leurs conseils pragmatiques et nos discussions animées.

Je remercie également Alain Zarli, Ahmed Laaroussi et Damien Hanser pour m'avoir fait partager leur connaissance en modélisation d'information au sein du secteur de la construction.

J'adresse aussi mes remerciements à Geneviève pour m'avoir fait progressé considérablement en anglais durant ces trois années, ainsi qu'à Marie-Noëlle pour ses formations qui m'ont aidé à rédiger ce manuscrit. Je remercie aussi Mme Alice Tran pour sa patience et le temps qu'elle a consacré aux formalités administratives de ma thèse.

J'exprime toute ma reconnaissance à mes parents, pour leur soutien et leur réconfort qui ont rendu tout cela possible. A Maman, pour le temps passé à la relecture de mes écrits. A Papa, pour ses nombreuses recommandations.

Mes derniers remerciements vont à Emilie qui a su m'apporter beaucoup de sérénité durant cette dernière année de thèse.

# Table des matières

# Introduction générale

L'essor des nouvelles technologies de l'information et de la communication (NTIC) ne touche pas à sa fin. Bien au contraire, après la généralisation de l'informatisation et des outils numériques, l'émergence des technologies communicantes confirme le rythme *presto* des innovations. Toutes les organisations sont d'accord sur les avantages immédiats et potentiels de l'adoption des NTIC.

Au niveau des données, la productivité augmente concernant la saisie d'information, ce qui induit une baisse des coûts. L'accès aux données est facilité au point de supprimer les barrières spatiales : délocalisation de la production. Plus généralement, la facilité grandissante de l'accès à l'information permet aux organisations d'être plus réactives et de mener des ***veilles stratégiques*** plus approfondies.

Au niveau des activités et des acteurs, les NTIC possède aussi des atouts et des limites. Elles favorisent la transversalité des échanges au sein d'une organisation, ou même entre organisations différentes. Le poids de la hiérarchie semble donc allégé, de même pour la gestion des carrières et des ressources humaines. D'un autre côté, cette adoption implique la formation du personnel et une *conduite du changement* délicate.

Ainsi, dans le secteur du BTP, la résistance au changement se révèle par nature très forte. En effet, la tradition orale de ce secteur, l'implication d'un grand nombre d'acteurs de cultures différentes, la part importante donnée à la phase d'éxécution des travaux — où bâtir devient plus prioritaire qu'accumuler des connaissances — sont autant d'exemples qui freinent la transition vers les NTIC. Plus généralement, nous estimons que c'est l'ensemble des règles implicites existant dans un projet de génie civil (expression des besoins, coopération entre acteurs, etc.) qui rend la transition vers le « tout numérique » plus difficile, comparée à d'autres secteurs d'activités.

Si les logiciels de CAO, les échanges de données informatisées géométriques et la génération de plans électroniques connaissent déjà un succès majeur au sein du secteur du BTP, il reste à montrer les apports d'une centralisation numérique globale de l'information pour n'importe quel projet architectural ou d'infrastructure.

## Contexte de l'étude et finalité de nos travaux

Ce travail de thèse a été élaboré sous la responsabilité de l'école doctorale *Information, Communication, Modélisation, Simulation* (ICMS) de *l'École Nationale des Ponts et Chaussées* (ENPC). Au quotidien, ces travaux ont été menés au CSTB (Sophia-Antipolis), à la division Modélisation et Environnements Virtuels Enrichis (MODEVE), au sein du département *Technologies de l'Information et Diffusion du Savoir* (TIDS).

Nos travaux se situent à l'interface entre plusieurs disciplines de recherche : d'un point de vue informatique, ils s'inspirent de la thèse de Christelle Urtado [Urtado98] sur le versionnement structurel d'objets et celle de Andrew Philip Broad [Broad03] sur la comparaison de modèles d'un point de vue sémantique. Cependant, la spécificité du secteur de la construction nous a amené à exploiter des concepts des sciences de l'information et des sciences cognitives, appliqués au secteur du BTP. Par conséquent, nous nous sommes basés sur les travaux des thèses de Damien Hanser [Hanser03] et Ahmed Laaroussi [Laaroussi07a]. Enfin, grâce à l'environnement de travail dans lequel a été conduit cette thèse, la notion de **Maquette Numérique** revient de manière récurrente.

La Maquette Numérique (MN en français, Digital Mock-Up : DMU en anglais) est associée à diverses définitions complémentaires. Nous pensons cependant qu'elles restent trop liées à la notion de maquette purement visuelle. Comme il s'agit d'un concept central dans nos travaux, nous apportons ici notre propre définition de la maquette numérique, intégrant celles que nous avons pu lire dans notre étude bibliographique : la maquette numérique est en premier lieu une représentation numérique, centralisée et hiérarchisée de l'ensemble des productions des acteurs au cours d'un projet. De plus, à cette représentation est associée une intention particulière : il s'agit de comprendre le produit avant qu'il n'existe physiquement. Cette compréhension peut inclure la visualisation 3D, mais aussi d'autres informations : la dimension temporelle (pour des projets basés sur la CAO 4D), d'autres dimensions comme le coût (pour la CAO nD), des informations techniques (matériaux, contraintes structurelles, etc.). L'objectif consiste alors à tester le produit à la manière d'un prototype, grâce à différentes simulations. Par conséquent, le processus de création de la maquette numérique est parfois nommé prototypage virtuel (Virtual Prototyping, VP).

La Maquette Numérique devient donc un modèle d'***artefact*** de projet en cours de conception. Dans le cadre du secteur du BTP, nous utiliserons l'expression **Maquette Numérique de Construction** (MNC). Il s'agit de la maquette numérique pour un projet de génie civil. Contrairement à ce que pourrait évoquer le nom, il s'agit d'un artefact valable durant tout le cycle de vie de l'ouvrage : il est créé durant la conception, mis à jour durant la construction et exploité durant la maintenance. Lors de la conception, de nombreux échanges entre acteurs ont lieu autour de la MNC. L'activité qui consiste alors à valider, à un moment donné, toutes les propriétés d'une MNC entre tous les acteurs se nomme *synchronisation de la MNC*.

De plus, la conception en génie civil semble se distinguer de celle des produits manufac-

turés, notamment à cause de la création d'un ouvrage unique. Par conséquent, les processus en jeu durant cette phase nécessitent une analyse particulière. Notre travail consiste donc à proposer des mécanismes d'aide à la synchronisation de la MNC, après avoir montré que les outils existants, adaptés à la conception de produits manufacturés, ne peuvent être directement transposés au secteur du BTP. Pour cela, cette thèse se focalise sur le suivi des modifications de la MNC en cours de conception, par l'intermédiaire de comparaisons d'information structurées (comparaisons dites *structurelles*), tout en prenant en compte la signification des concepts utilisés (comparaisons sémantiques).

Cette étude ne se limite pas à un seul aspect particulier de la MNC, comme la visualisation 3D ou la CAO 4D. Cette thèse traite la globalité du système d'information sous-jacent, c'est-à-dire l'ensemble des concepts, des objets et de leurs relations au sein de la MNC. Néanmoins, une attention particulière est portée sur la conception des ponts : cette dernière implique les mêmes étapes générales que pour la conception de bâtiments, mais les problématiques de collaboration semblent plus simples, notamment par le nombre plus réduit d'acteurs impliqués. Nous estimons à ce titre que l'application de nos travaux à ce secteur portera directement ses fruits. Néanmoins, une des finalités de cette thèse consiste à produire des algorithmes aussi généralisables que possible.

Dans ce cadre, les apports visés par cette thèse sont les suivants :

- Analyser les logiciels et les modèles existants autour de la MNC et de sa synchronisation. Détailler cette analyse pour la conception de ponts : types données échangés, énumération des modèles de données existants.
- Proposer un processus-type, pour la gestion de la MNC, adapté aux pratiques professionnelles du bâtiment et des ouvrages d'arts.
- Concevoir et implémenter des algorithmes pour le suivi des modifications de la MNC, indépendamment de la ***persistance*** de cette dernière.
- Intégrer les algorithmes développés au sein d'environnements logiciels existants, en particulier la CAO pour le bâtiment, et le calcul de structure pour les ponts.
- Évaluer les outils développés par rapport aux activités de conception et au-delà, c'est-à-dire lors de l'éxécution des travaux et de la maintenance de l'ouvrage.

## Plan du mémoire

Le corps de ce mémoire est divisé en cinq chapitres principaux :

- Le **chapitre 1** rappelle l'évolution historique des échanges de données informatisées (EDI) vers la MNC, en passant par les standards d'échange. De plus, les concepts principaux pour la compréhension de nos travaux seront définis. Ils permettront de comprendre les offres existantes et les dernières innovations concernant la MNC et sa synchronisation. Enfin, certaines limites concernant l'adaptation des solutions existantes au secteur du BTP nous amèneront à développer un processus-type de gestion de la MNC. Il servira de base pour notre démarche et introduit la notion de suivi des modifications de la MNC.

- Le **chapitre 2** propose une première méthodologie pour le suivi des modifications : la comparaison structurelle. L'objectif consiste à définir le contenu du résultat attendu par une telle comparaison et de concevoir les algorithmes qui la mettent en oeuvre.
- Le **chapitre 3** complète le système de comparaison structurelle par une analyse sémantique : le suivi des modifications prend en compte non seulement la structure des objets (propriétés, relations), mais aussi leur signification au sein de la MNC. Cette analyse permettra entre autres d'extraire l'information à comparer (dans le cas de comparaisons partielles) et de définir des équivalents sémantiques (dans le cas où il existe plusieurs manières de représenter une même information).
- Le **chapitre 4** propose des implémentations informatiques, en langage C++, correspondant aux algorithmes conçus dans les deux premiers chapitres. Cette partie sera suivi par plusieurs exemples d'intégration au sein d'environnements logiciels existants.
- Le **chapitre 5** évalue la méthodologie choisie dans ces travaux, tant au niveau du processus-type qu'au niveau du comparateur sémantique. Cette évaluation étudie en particulier la portée de nos travaux par rapport à la problématique de fusion des modèles d'ouvrage. Elle nécessitera d'analyser les liens entre la MNC et les activités de conception.

Le choix d'un tel plan et notamment l'ordonnancement des chapitres découlent du processus de réflexion suivant :

1. **Analyse** du problème et de l'environnement existant. Il s'agit de la quasi-totalité du premier chapitre.
2. **Propositions** de solutions et d'innovations pour résoudre la problématique posée. Ainsi, à la fin du premier chapitre, le processus-type constitue le premier apport de cette thèse. Ensuite, le chapitre 2 et 3 représentent le corps de notre démarche théorique. La première moitié du chapitre 4 propose enfin une implémentation informatique de notre démarche.
3. **Évaluation** des innovations proposées. La deuxième moitié du chapitre 4 étudie l'applicabilité de nos travaux en relation avec des logiciels existants. Le chapitre 5 analyse la portée de nos travaux sur des phases déterminantes de la conception et au-delà, en phase de construction et de maintenance.

## Remarques lexicographiques

Si cette thèse porte l'analyse en priorité sur les documents et les modèles d'ouvrages, notamment lors de la conception, les liens avec les activités et les acteurs eux-mêmes sont suffisamment forts pour qu'ils viennent parfois illustrer notre développement. Concernant les acteurs de projet, nous citerons souvent deux types d'acteurs :

- Le constructeur d'application : il s'agit d'une personne ou d'une organisation capable d'élaborer un logiciel ou un système d'informations. Par exemple, un éditeur de logiciels comme Autodesk ou Graphisoft, une société de services en ingénierie informatique (SSII), ou bien un membre du service informatique d'une grande entreprise. Quand il est cité, cela signifie qu'une expertise est nécessaire pour effectuer un développement informatique car nos travaux n'ont pas pour but de fournir un outil fini « clef en main ».

- Le coordinateur de projet : au sein d'un projet en conception, l'adoption des NTIC favorise le travail collaboratif. Certaines tâches nécessitent des coordinations. Afin d'illustrer simplement certains points sur la synchronisation de la MNC, nous confondrons l'activité de coordination et l'acteur dédié à cette coordination. En effet, nous ne souhaitons pas imposer à une équipe-projet de désigner une personne comme étant coordinateur de projet. Les tâches de coordination pourraient par exemple être réparties entre plusieurs acteurs. Lorsqu'une tâche de coordination est nécessaire, nous utiliserons cette expression.

Une dernière remarque concerne la mise en forme de ce document. A chaque fois qu'un mot sera représenté ainsi : ***concept*** (à la fois en **gras** et en *italique*), cela signifie qu'une définition de ce mot existe dans le glossaire, situé au début de ce mémoire, après la table des figures.

# Chapitre 1

# Etat de l'art sur la Maquette numérique de construction et sa synchronisation

## 1.1 Introduction

L'informatisation est une évolution des méthodes et des outils de travail qui a concerné tous les domaines industriels. Nous pouvons y distinguer deux phases successives. D'abord, cela consiste à définir un modèle informatique pour le support des données, afin d'augmenter la productivité en conception de projet. En effet, les outils de CAO permettent d'accélérer le travail du concepteur. Ensuite, pour que la productivité continue de progresser, la communication et la collaboration des acteurs de projet s'améliorent. Cela peut se traduire par une amélioration de l'accès aux données, que ce soit en lecture ou en écriture. Dans le domaine du BTP, s'il existe déjà des modèles de données standardisés, leur utilisation pour améliorer la collaboration entre acteurs n'a pas encore été pleinement adoptée.

Dans ce contexte, nous introduisons le concept de Maquette Numérique de Construction (MNC). Il englobe la notion de modèle informatique d'ouvrage, de centralisation d'information et de dématérialisation de l'information. Les standards d'échanges permettent de spécifier la structure de n'importe quel modèle d'ouvrage. La première phase de l'informatisation permet de résoudre ce problème. La centralisation de l'information permet d'échanger des données entre acteurs venant de métiers différents. Cette problématique concerne en particulier le domaine du BTP. Il s'agit de la transition entre les deux phases de l'informatisation : le standard d'échanges est défini et facilite l'accès aux données, y compris entre acteurs utilisant des outils différents. Enfin, la dématérialisation complète de l'information indique que les projeteurs n'ont plus du tout besoin d'un support papier, ils ont donc adopté les méthodes de conception autour de modèles entièrement informatisés, ce qui correspond à l'aboutissement de la deuxième phase d'informatisation.

Ce chapitre vise donc à détailler les approches existantes selon ces deux phases : d'une part la définition des modèles de données pour le domaine de la construction, et leur évolution vers la MNC (section 1.2), d'autre part les méthodes d'utilisation des modèles dans le cadre d'une

conception collaborative (section 1.3). Pour la dernière phase, nous avons choisi d'analyser les mécanismes existants pour synchroniser la MNC. Nous verrons que de tels mécanismes ont un impact sur la collaboration entre acteurs de projets. A partir de cet état de l'art, une analyse des limites des méthodes existantes sera effectuée (section 1.4) afin de définir un cadre d'étude et des objectifs pour les travaux de cette thèse (section 1.5).

## 1.2 Du modèle de données à la MNC : approches existantes

La conception de structures porteuses était encore réalisée dans les années 90 par des services d'ingénierie publique et n'impliquait essentiellement que des ingénieurs et techniciens. Cependant la complexité croissante des projets d'infrastructures nécessite des compétences pluridisciplinaires incluant ainsi des architectes, paysagistes, urbanistes, etc. [Lamour05]. De plus la construction de ponts trouve des points communs avec celle de bâtiments : métiers impliqués, méthodes de construction, notion d'ouvrage unique. Plus généralement, la construction est un secteur industriel à part entière, où la communication et une collaboration optimales entre acteurs différents est nécessaire pour améliorer la conduite d'un projet. Or les méthodes de projet et les techniques de conception sont plus avancées dans des domaines tels que l'aéronautique ou l'automobile par rapport au BTP. Il existe néanmoins des modèles de données standardisés et des solutions de MNC dans le domaine de la construction.

C'est pourquoi, après avoir détaillé les types de données échangeables dans le génie civil et avoir tracé un bref historique des standards d'échanges industriels (section 1.2.1), nous aborderons la norme internationale STEP (**ST**andard for the **E**xchange of **P**roduct model data) qui définit de manière exhaustive la façon de créer un standard d'échange adapté (section 1.2.2) puis son application dans le domaine du bâtiment avec les IFC (Industry Foundation Classes) (section 1.2.3). Enfin l'extension du modèle IFC au génie civil sera détaillée à la section 1.2.4.

### 1.2.1 Échanges de données informatisées durant le cycle de vie d'un ouvrage

L'étude de structures porteuses fait l'objet d'échanges de notes techniques accompagnées de plans. Ces derniers génèrent un flux de grand volume. Cela s'explique par le nombre d'intervenants sur un projet donné : maître d'oeuvre, Bureau d'études, Entreprises, Architecte, Géomètres, etc. Après l'informatisation des données techniques est né le besoin d'assurer leur communication directe de logiciel à logiciel sans rupture de charge provoquée par la saisie manuelle. Cependant, pour qu'il y ait échange, il faut qu'il y ait compréhension et donc existence d'un langage commun. Cette voie a été suivie en premier par les industries aéronautiques puis automobiles.

#### Inventaire des données échangeables pour un projet de pont

[AFPC94] établit un inventaire détaillé des données échangées durant les phases d'études techniques des structures porteuses. Nous pouvons y distinguer deux types de données :

- **Les données initiales** : elles correspondent aux données de l'environnement existant avant la construction du projet. Ces sont les données fonctionnelles (gabarits routiers, ferroviaires et fluviaux), les données topographiques, les données géologiques, et les données d'implantation de l'existant ou du projet (liens avec le projet routier sous-jacent). Parmi ces différents thèmes, nous retiendrons les données topographiques, en particulier les relevés de points, et les données d'implantation définissant complètement la position de l'axe de référence de l'ouvrage et des structures porteuses.
- **Les données de conception** : toutes les autres données principales d'une étude d'ouvrage sont abordées en phase de conception de projet. Elles correspondent aux informations géométriques, techniques, mécaniques, de construction, et sur les matériaux.

Parmi l'ensemble des données à échanger, le tableau 1.1 regroupe celles qui ont le plus besoin d'être échangées entre acteurs de projet.

| DONNÉES | VOLUME DE DONNÉES | SIMPLICITÉ DES DONNÉES | BESOIN D'ÉCHANGE |
|---|---|---|---|
| - relevés de points topographiques | *** | * | *** |
| - données d'implantation d'un ouvrage d'art : ligne rouge | * | *** | *** |
| - implantation des éléments porteurs | ** | ** | *** |
| - coffrage courant de poutre ou de poteau | ** | ** | *** |
| - coffrage courant de dalles et de murs | ** | ** | *** |
| - nomenclature des aciers | ** | ** | *** |
| - géométrie de la précontrainte | *** | ** | *** |
| - éléments prismatiques : données simplifiées | ** | *** | *** |
| - données de construction des éléments en béton | ** | ** | *** |
| - données de construction des éléments de liaison | ** | ** | *** |
| - données mécaniques d'une modélisation barres | *** | ** | *** |
| - données mécaniques d'une modélisation éléments finis | *** | ** | *** |

**TABLEAU 1.1:** Récapitulatif des données les plus échangées d'après [AFPC94]. Le nombre d'étoiles (de 1 à 3) qualifie l'importance d'un critère pour une donnée particulière.

Ce tableau montre que des échanges sont nécessaires entre acteurs de projet de pont, ils peuvent être nombreux et porter sur des gros volumes de données (relevés topographiques, géométrie de la précontrainte, résultats de modélisation en éléments finis...). Néanmoins les types de données échangeables et le besoin d'échanges dépendent aussi du projet lui-même, notamment de la relation entre la maîtrise d'ouvrage et la maîtrise d'oeuvre [Lamour05]. En effet, conformément aux dispositions de la loi du 12 juillet 1985 sur la maîtrise d'ouvrage publique, dite « loi MOP », il appartient à la maîtrise d'ouvrage d'élaborer un pré-programme et un programme, auxquels répond la maîtrise d'oeuvre par des études préliminaires puis des études d'avant-projet. Le contenu du programme et la façon dont la maîtrise d'oeuvre sera choisie (concours, procédure négociée spécifique, appel d'offres, ou encore conception-réalisation) ont un impact déterminant sur le type de données échangeables. [Ku06] montre ainsi qu'un pro-

jet collaboratif autour d'une maquette numérique 3D ne peut fonctionner correctement que si la maîtrise d'ouvrage spécifie les modèles 3D comme documents contractuels. Cette démonstration s'appuie sur une étude de cas autour du projet d'un pont à Columbus dans l'Ohio.

Néanmoins nous gardons comme hypothèse que l'automatisation des échanges améliore la qualité et la rapidité de la communication des données. A cette fin une solution potentielle pour assurer l'automatisation est celle du standard d'échange qui fournit un langage commun entre acteurs de projet.

### Les premiers standards d'échange

Les échanges de données informatisées ont explosé depuis les années 1980. C'est pourquoi les standards d'échanges ont été nécessaires très tôt dans l'industrie. En effet, le tableau 1.2 résume quelques dates-clef sur les premiers standards d'échange.

| ANNÉE | ÉVÉNEMENT |
|---|---|
| 1979 | BOEING créé le standard IGES dans le secteur de l'aéronautique [Nagel80]. |
| 1982 | Autodesk sort la première version du format d'échange DXF avec AutoCAD 1.0. |
| 1983 | Le sous-comité ISO/TC184/SC4 lance le projet STEP |
| 1984 | La France se dote son propre standard SET dans le cadre du projet AIRBUS [AFN93] . |
| 1994 | Première publication de STEP en tant que norme internationale, notamment le format AP203 [ISO94b] |
| 1994 | Autodesk publie les spécifications du format DXF avec la sortie de AutoCAD 13. |
| 2001 | Publication du protocole STEP AP214, pour l'automobile, supporté par CATIA. |

**TABLEAU 1.2:** Historique des premiers standards d'échange

Dans un premier temps les échanges se sont focalisés sur les informations graphiques, ces dernières s'enrichissant suivant l'évolution des logiciels de CAO : données filaires, surfaciques puis volumiques. De nouvelles entités géométriques, mécaniques et technologiques ont ensuite enrichi les échanges. En outre le secteur de l'automobile a suivi l'aéronautique en s'équipant de systèmes CAO puissants et en utilisant les standards pour faire communiquer ces systèmes et ceux des sous-traitants. Très rapidement les standards nationaux se sont multipliés et la nécessité de créer un standard international s'est imposée. Cela a conduit à la création de la norme STEP, comme nous le verrons dans la section 1.2.2.

Nous avons identifié trois caractéristiques principales d'un standard d'échange, quelque soit le secteur industriel concerné :

- Un standard permet de constituer un fichier ou plus généralement une base de données modélisant un objet de plusieurs manières : géométriquement, mécaniquement, ou encore technologiquement. Les modèles les plus aboutis prennent en compte des informations temporelles (chaîne de fabrication, phasage de construction) et de projet (coûts, tâches, activités, acteurs).

- Les données générées sont qualifiées de neutres ou d'interopérables car elles ne sont liées à aucun système matériel ou logiciel. Elles suivent pour cela un schéma de description (cf. section 1.2.2).
- La définition d'un standard fait l'objet d'un accord entre les représentants de chaque corps de métier concerné et de chaque pays impliqué dans le cadre de standards internationaux. La normalisation permet de formaliser cet accord.

Pour établir un standard d'échange, les acteurs sont toujours conduits à construire un ***modèle de données*** qui fixe les règles concrètes pour hiérarchiser et interpréter les données sous-jacentes. Lorsqu'un tel modèle est validé par tous les types d'acteurs concernés, il peut être considéré comme standardisé. Il peut même devenir normalisé si cela est nécessaire, en particulier pour des échanges internationaux. La notion de modèles de données, qui sera définie dans la partie suivante, appartient aux sciences de l'information. Afin de caractériser un standard d'échange sous cet angle, nous confondrons la notion de standard d'échange et celle de modèle de données (standardisé) dans la suite de cette présentation.

### 1.2.2 Définition d'un modèle de données standardisé — la norme STEP

Lors de la conception d'un système d'information, la modélisation de données analyse et conçoit l'information contenue dans le système. Pour cela, nous pouvons distinguer deux niveaux principaux : le modèle d'information et le modèle de données. Comme tous les domaines industriels sont potentiellement concernés par la modélisation de données, la norme STEP vise à fournir un environnement complet — dont la description est très exhaustive — pour construire un système d'information et en particulier modéliser les données. Cette partie vise donc à rappeler quelques notions élémentaires des sciences de l'information avant d'introduire l'approche choisie par la norme STEP.

#### Du modèle d'information au modèle de données

Dans un domaine industriel fixé, les différents projets associés exploitent un ensemble d'informations. Elles sont alors utilisées et réutilisées par beaucoup de programmes informatiques. Afin de comprendre la démarche de la modélisation de données, voici les définitions de plusieurs concepts élémentaires des sciences de l'information.

**Définition 1.** *On appelle* **univers du discours** *le domaine réel ou conceptuel auquel on s'intéresse, et les points de vue associés. Il se réfère généralement à tout ensemble de termes utilisés dans un discours spécifique, c'est-à-dire une famille de termes sémantiques spécifiques au domaine concerné.*

Une première difficulté consiste à intégrer plusieurs points de vue, en particulier dans le cas d'échanges de données entre acteurs différents. De plus, les objets modélisés sont des entités réelles dont la structure est complexe et changeante avec l'évolution et les progrès du domaine industriel concerné. C'est pourquoi cette modélisation doit faire appel à tous les types d'acteurs concernés et nécessite une collaboration optimale entre eux.

**Définition 2.** *On appelle* **modèle d'information** *l'ensemble des informations d'un domaine dans un certain contexte sous une forme compréhensible par un humain. Il est indépendant des applications et des implémentations.*

**Définition 3.** *On appelle* **modèle de données** *la description abstraite et logique d'une représentation des données dans une organisation métier ou un système d'information. Contrairement au modèle d'information, cette description est automatiquement implémentable, grâce à des règles d'interprétation explicites ou implicites.*

De manière formelle, [Pierra00] présente la modélisation d'information par trois niveaux (cf. figure 1.1). Dans ce type de représentation, à partir d'un modèle d'information dont l'approche est purement conceptuelle et sémantique, on peut en déduire plusieurs modèles de données possibles. Ces derniers sont décrits suivant un langage de description. Le langage EXPRESS [ISO94a], le schéma XML (appelé XSD, décrit par [Brown01]) et ses extensions au Web sémantique (RDF [Brickley04] et les ontologies OWL [Mcguinness04]) sont autant d'exemples possibles pour décrire un modèle de données. Dans tous les cas, nous nous focaliserons sur l'approche orientée objet pour la conception de modèles au vu de ses avantages [Turk92], [Turk93]. A partir de cette spécification logique de données, le modèle choisi devient implémentable car il ne peut pas être soumis à une interprétation arbitraire. A cette fin, plusieurs implémentations sont possibles, en fonction du langage, des choix de conception de l'application et des supports pour la persistance de données. Cette dernière étape de ***mapping*** est automatisable car le modèle en entrée a suivi un langage de description.

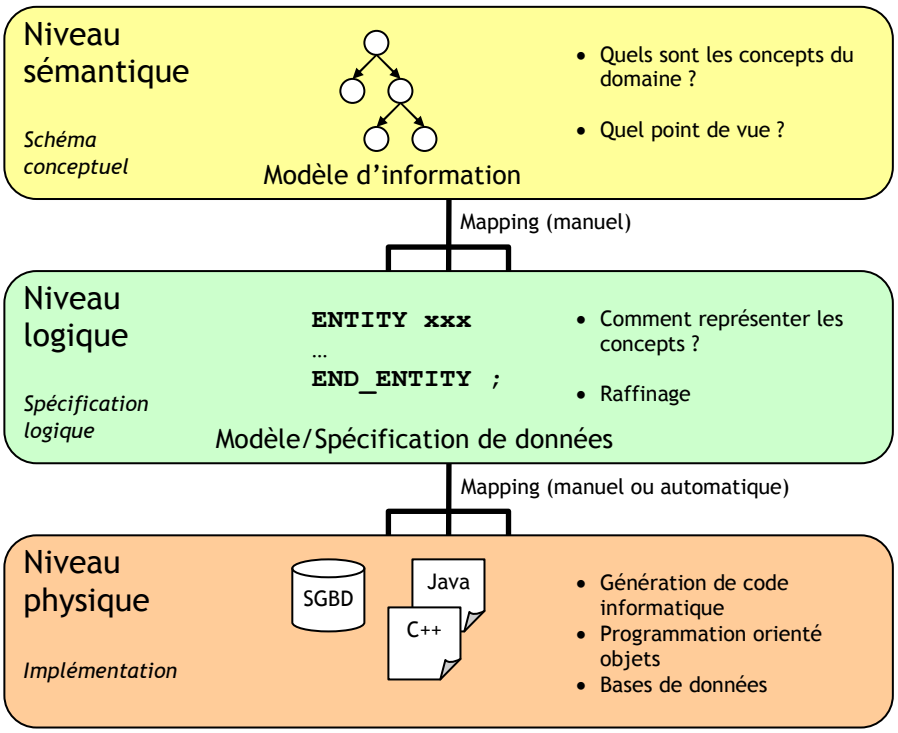


**FIGURE 1.1 :** Illustration de la modélisation de données à trois niveaux d'après [Pierra00]

A propos de ce formalisme, si la frontière entre le niveau logique et physique est nette, il n'en est pas de même pour les deux premiers niveaux. Certes, [Pras03] confirme la distinction à faire entre modèle d'information et modèle de données, dans le cadre de la transmission de données

en réseau. Dans le même temps il avoue que certains formalismes peuvent être considérés tantôt comme un modèle d'information, tantôt comme un modèle de données. En effet, au moment de la modélisation de l'information, le concepteur peut formaliser ses idées grâce à des schémas sous forme de graphes ou d'arbres. Afin qu'il se fasse comprendre par ses collègues (et par habitude), il est tenté d'utiliser des descriptions visuelles normalisées comme UML, les ontologies, NIAM ou EXPRESS-G. Cependant cette description peut être accompagnée de remarques, de justifications ou de pistes de réflexion. A travers cet exemple, nous pouvons constater qu'au moins une partie du modèle d'information créé est déjà un modèle de données sous forme graphique et donc directement implémentable. Au moment de créer un modèle de données sous-jacent approprié, il convient d'adapter la structuration des informations en fonction des possibilités du langage choisi. Par exemple, le langage EXPRESS propose des mécanismes d'héritages complexes, des fonctions et des règles alors que XSD ne propose que de l'héritage simple et ne permet pas de fixer des contraintes évoluées sur les objets.

Dans tous les cas, une fois le modèle de données développé, il reste encore à concevoir et implémenter la façon dont les objets (appelés aussi ***instances***) du projet seront créés et échangés.

#### Les échanges de données à travers les instances du modèle

Dans le cadre d'un projet industriel, l'instance d'un modèle de données est lui-même un modèle, celui du produit à concevoir et à construire, que ce soit un véhicule, un bâtiment, un pont, etc. c'est-à-dire les données elles-mêmes, sous une forme hiérarchisée suivant le modèle de données. Pour bien différencier modèle de données et instance de modèle, voici un exemple simple. Nous pourrions en première approximation associer la langue française à un modèle de données avec tout son vocabulaire (l'univers du discours), son orthographe et sa grammaire (ensemble de règles et des contraintes définies dans le modèle). Une instance de ce modèle serait alors n'importe quelle phrase, texte ou roman écrit correctement en langue française. L'instance est donc un ensemble d'objets hiérarchisés de façon à suivre le formalisme défini sans ambiguïté par le modèle de données.

L'information contenue dans l'instance est donc échangeable sans interprétation arbitraire grâce au modèle de données. Néanmoins, avec l'utilisation de l'outil informatique, l'échange proprement dit s'effectue par fichiers ou par réseau et nécessite à ce titre d'être standardisé.

**Définition 4.** *On appelle* **sérialisation** *le procédé visant à transformer une structure en mémoire en un flux de données séquentiel «transportable» par fichiers ou à travers un réseau. L'opération inverse de décodage du flux pour retrouver la structure originale est appelée par extension* **désérialisation**.

Certains langages de spécification de données permettent de définir à la fois le modèle de données et la sérialisation des instances. Le formalisme XML regroupe à fois la spécification de données via DTD et XSD [Brown01] et sa sérialisation [Bray04]. Le web sémantique propose aussi des variantes, basées sur XML. Dans ce formalisme les instances et les modèles tendent à fusionner autour d'un méta-modèle unifié, car l'objectif est différent, il s'agit ici de modéliser la connaissance et non pas des données particulières à un projet.

STEP propose aussi un langage de spécification, le langage EXPRESS, et plusieurs méthodes de sérialisation.

### STEP : une approche générique pour tout projet industriel

**Définition 5.** *On appelle **cycle de vie** un ensemble ordonné de phases décrivant la vie d'un projet, la phase $n$ ne pouvant commencer que si la phase $n-1$ est terminée [CNRS06].*

Le formalisme ***STEP*** (STandard for Exchange of Product model data) a pour mission la construction d'un standard permettant de traiter la représentation et l'échange de données de modèles de produit en couvrant leur cycle de vie. Il doit donc être :

- Un standard multi-applications traitant tous les produits manufacturés, tous les métiers et tous les stades du cycle de vie d'un produit.
- Un standard multi-utilisations pour les échanges de données en définissant un format neutre, l'archivage en couvrant la durée de vie des produits, les bases de données produit en permettant l'intégration des applications.

STEP propose ainsi l'environnement le plus complet pour prendre en charge un projet industriel. [Nell03] résume les différentes parties de STEP autour de six axes majeurs. Citons simplement quatre axes fondamentaux dans le cadre de la modélisation de données :

- Méthodes de description : cet axe est lié à la spécification des données, en particulier la partie STEP-11 (langage EXPRESS) [ISO94a].
- Méthodes d'implémentation : cet axe propose plusieurs sérialisations des données (STEP-21 en ASCII [ISO94c] et STEP-28 en XML [ISO03]) ainsi que des méthodes de mapping (STEP-22) pour passer du niveau logique au niveau physique (cf. fig. 1.1) [ISO95], dans différents langages (STEP-23 en C++ [ISO96], STEP-24 en C [ISO03], STEP-27 en Java [ISO01]).
- Ressources intégrées génériques : cet axe contient un ensemble de structurations thématiques d'informations pouvant servir à tous les domaines industriels. Par exemple STEP-42 propose une définition implicite et générale des formes géométriques [ISO00].
- Protocoles d'application (AP en anglais) : il s'agit de ***mappings*** de l'ensemble des ressources intégrées génériques pour des domaines techniques et industriels particuliers. Par exemple, l'AP203 sert à la CAO 3D, l'AP210 à l'assemblage électronique, l'AP214 à l'industrie automobile [Nell03].

Nous verrons dans les sections suivantes que la construction n'a pas directement suivi la voie du protocole d'application mais s'est seulement inspirée des parties fondamentales de STEP. Pour la suite de l'exposé, il est nécessaire d'aborder plus en détails deux parties essentielles de STEP : la partie STEP-11 (ISO 10303-11) qui définit le langage EXPRESS [ISO94a] et la partie STEP-21 (ISO 10303-21) qui propose une sérialisation ASCII des données.

Le langage de spécification de données EXPRESS est ensembliste et orienté objet. Développé entre 1985 et 1993 pour atteindre un premier statut normatif en 1994, il possède une partie algorithmique via les *RULES* et les *FUNCTIONS* lui permettant d'exprimer n'importe quel type de contrainte. Comme il s'agit d'un langage de spécification de données, il est bon de rappeler que les entités EXPRESS ne décrivent pas de comportement, de ***méthodes*** dans le paradigme de

```
ENTITY IfcCartesianPoint
SUBTYPE OF (IfcPoint);
Coordinates : LIST [1:3] OF IfcLengthMeasure
DERIVE
Dim : IfcDimensionCount := HIINDEX(Coordinates);
WHERE
WR1 : HIINDEX(Coordinates) >= 2;
END_ENTITY
```

**FIGURE 1.2 :** Définition d'une entité dans le langage EXPRESS. Elle se nomme `IfcCartesianPoint`, elle hérite des attributs de l'entité `IfcPoint`. Elle possède de plus un attribut `Coordinates` qui est une liste de 3 éléments qui sont de types `IfcLengthMeasure`. De plus, elle possède un attribut 'Dim' qui se calcule automatiquement et qui vaut ici 3.

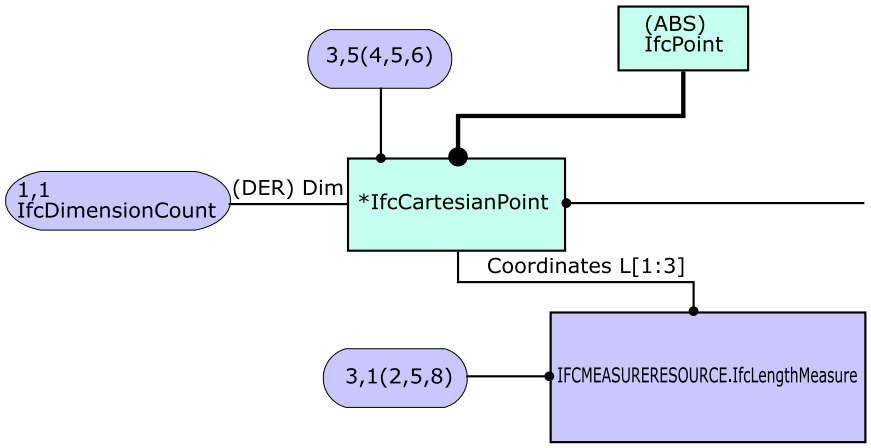


**FIGURE 1.3 :** Formalisme graphique EXPRESS-G de la classe définie sur Fig. 1.2

la ***POO*** (Programmation Orienté Objets). Ce langage possède aussi un formalisme graphique, EXPRESS-G, pour représenter les classes sous forme de diagramme. (cf. figure 1.3). EXPRESS permet donc d'exprimer un modèle de données sémantiquement riche et représentable sous forme de graphe. Il permet ainsi de rapprocher le modèle d'information du modèle de données [Pierra00]. Cette fusion entre le niveau conceptuel et le niveau logique (cf. figure 1.1) facilite le travail du concepteur de modèles en allégeant le mapping manuel entre ces deux niveaux.

La partie STEP-21 (ISO 10303-21) est responsable de la sérialisation des données. Elle propose pour cela un encodage ASCII des instances de modèles. Les fichiers générés sont ainsi lisibles par un simple éditeur de texte. La figure 1.4 propose un exemple de cet encodage. Elle montre entre autres qu'une instance d'un modèle STEP est finalement assimilable à un graphe, dans lequel les noeuds portent un identifiant unique sous la forme `#<nombre>`, et la déclaration d'objets complexes créé un lien père-fils avec les paramètres de la déclaration. Ces derniers sont eux-même des objets déclarés avec un identifiant spécifique. Par exemple, sur la figure 1.4, l'objet `#13` est un objet de type IfcAxis2Placement3D et a besoin de trois paramètres, un IfcCartesianPoint, `#10`, et deux IfcDirection, `#11` et `#12`.

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(('IFC 2x'),'2;1');
FILE_NAME('C:\\Testing\\ADT3.3\\BeamColumn\\Beam.dwg','2002-09-23T17:12:02',('Anyone'),
('Autodesk'),'Ifc2x 22Aug2002 - IFC Toolbox Version 2.x (00/11/07)','AutoCAD','AutoCAD');
FILE_SCHEMA(('IFC2X_FINAL'));
ENDSEC;
DATA;
#1=IFCSIUNIT(*,.TIMEUNIT.,$,.SECOND.);
#2=IFCSIUNIT(*,.MASSUNIT.,$,.GRAM.);
#3=IFCDIMENSIONALEXPONENTS(1,0,0,0,0,0,0);
#10=IFCCARTESIANPOINT((0.,0.,0.));
#11=IFCDIRECTION((0.,0.,1.));
#12=IFCDIRECTION((1.,0.,0.));
#13=IFCAXIS2PLACEMENT3D(#10,#11,#12);
...
```

**FIGURE 1.4 :** Début d'un fichier au format ASCII STEP-21

Avec le succès du formalisme XML, la sérialisation STEP-28 est de plus en plus utilisée. Cette dernière encode l'instance d'un modèle STEP en format XML. La lecture d'un tel format y est donc aisée car de nombreuses bibliothèques sont disponibles pour traduire des données XML. Néanmoins, cette forme de sérialisation demande environ cinq fois plus d'espace disque que la sérialisation STEP-21. Pour des modèles d'ouvrages réels, cela peut devenir un handicap non négligeable. C'est pourquoi nous n'exploitons que la sérialisation STEP-21 dans le cadre de cette thèse.

Pour n'importe quel projet industriel, STEP propose des solutions complètes pour tout le cycle de vie d'un produit. Cependant son application est relativement lourde, c'est pourquoi l'investissement nécessaire à son adoption dissuade nombre d'acteurs de passer à des modes de gestion de projet partagé. Une solution alternative consiste à n'utiliser que certaines parties fondamentales de STEP dans le cadre d'une application ou d'un domaine d'étude précis. C'est le cas du secteur de la construction avec le modèle des *Industry Foundation Classes* (IFC).

### 1.2.3 IFC : un modèle standardisé de maquette numérique pour le bâtiment

L'application des nouvelles technologies à la construction n'a pas échappé à la règle des « islands of automation » [Björk95] : il existe d'innombrables applications logicielles couvrant tous les aspects de la construction, mais la représentation de l'information suit des formats propriétaires peu capables d'interopérer [Debras98]. La figure 1.5 illustre de façon humoristique le phénomène des îles des automatisations dans le domaine de la construction [Hannus00].

Avant STEP et les IFC, seuls des standards « de fait » comme DXF ont permis de communiquer des données purement graphiques, ce qui reste bien insuffisant pour des projets de bâtiments, ou même pour la conception de n'importe quel produit [AFNOR94]. L'absence d'interopérabilité provoque une fragmentation du mode de travail (cf. figure 1.6), et cause ainsi de nombreux problèmes de pertes d'information et des difficultés de communication entre corps de métier. Ce mode de travail fragmenté affecte le coût des projets de construction.

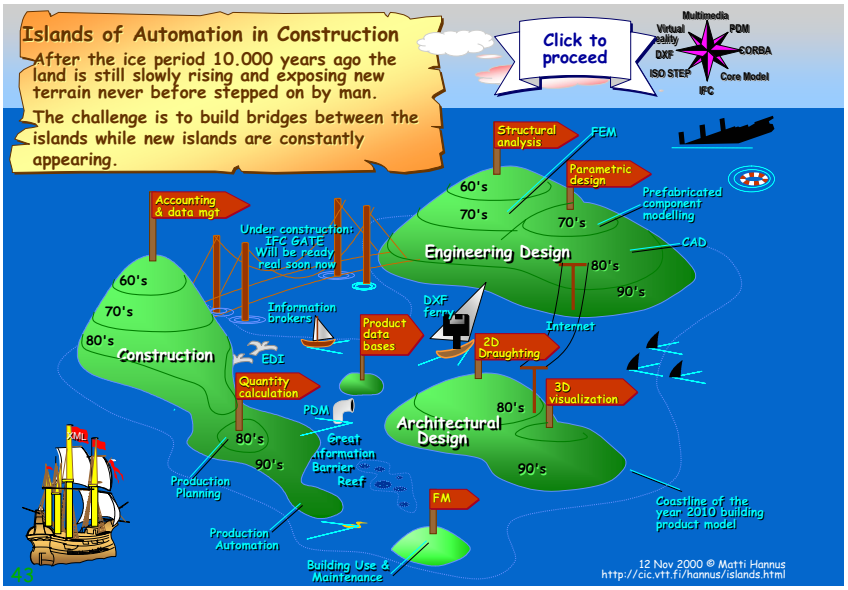


**FIGURE 1.5 :** Les îles d'automatisation dans la construction d'après [Hannus00].

Avec une approche plus aboutie de modélisation d'information (cf. partie précédente), le but est de réconcilier ces différents pôles d'automatisation grâce à un système d'information partagé par tous (cf. figure 1.7). Ce dernier fait appel au même dictionnaire de données et à la capacité de parler le même langage. Cette méthode améliore la productivité et la qualité du processus global de construction.

### Les premières tentatives avant la création des IFC

Afin de résoudre le problème des îles d'automatisation, des standards ont été élaborés. Afin que ces derniers soient promus efficacement, deux méthodes ont été exploitées : la normalisation et l'effet de masse des entreprises BTP. La normalisation (internationale), très adaptée pour les produits manufacturés, s'inscrit dans un contexte différent pour la construction. Le concept de produit est remis en cause car le bâtiment et la structure porteuse sont des ouvrages uniques [Ameziane95] [Hyenne93]. Ce contexte normatif a été néanmoins pris en compte par STEP pour créer des protocoles d'application adaptés. Plusieurs tentatives ont donc vu le jour [Nell03] :

- AP 225 : Structural Building Elements Using Explicit Shape Representation [Haas97] (approuvé et publié)
- AP 228 : Building Services : Heating, Ventilation and Air Conditioning (abandonné)
- AP 230 : Building Structural Frame : Steelworks (abandonné)
- Part 106 : Building Construction Core Model (abandonné)

Seule l'AP 225 a été retenue. Cette norme tente d'intégrer le contexte normatif de la construction et propose une description des éléments de construction à partir de leur représentation tridimensionnelle [Maïssa03]. Plusieurs expérimentations ont été menées autour de cette norme, notamment des possibilités d'export par le logiciel ALLPLAN de Nemetschek et des conversions

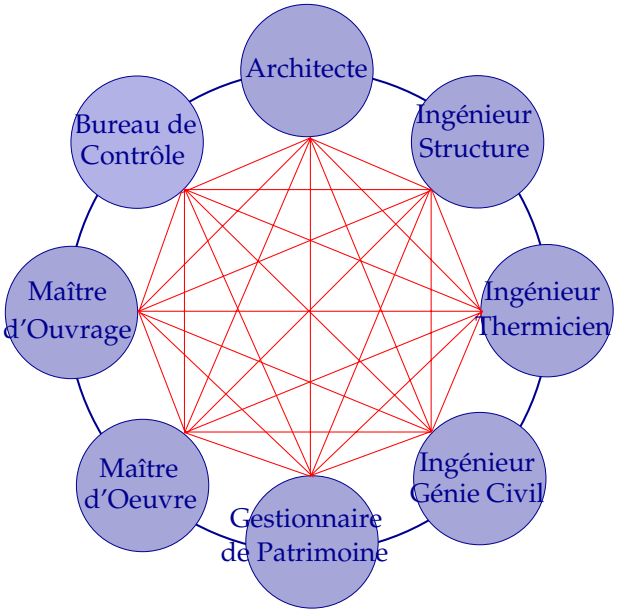


**FIGURE 1.6 :** Mode de travail fragmenté

vers le format VRML [Marache00]. Nous estimons cependant que la voie du protocole d'application de STEP a été un frein à la promotion cette norme. En effet, STEP suit un processus normatif ISO strict et jugé lent (3 ans) par nombre d'industriels de la Construction.

C'est pourquoi ces derniers ont décidé de se regrouper pour proposer une approche alternative. L'IAI (International Alliance for Interoperability) a été créée à cette fin. A l'origine, elle était composée de douze sociétés impliquées dans l'industrie de la Construction qui désiraient toutes être capables d'interopérer. Cela signifie que chacune veut pouvoir travailler avec l'information des autres sans se préoccuper du logiciel qu'elle utilise ou que les autres utilisent. Des premiers prototypes logiciels ont été présentés au salon AEC Systems '95 show à Atlanta. En 2004, L'IAI compte neuf Chapitres dans le monde, chacun couvrant les besoins d'une région géographique, et ses membres représentent aujourd'hui plus de 600 entreprises dans 20 Pays. L'objectif officiel de cette organisation est de favoriser l'interopérabilité des logiciels dans le secteur de la Construction. Sa mission principale consiste par conséquent à définir une base universelle pour l'amélioration du processus et le partage d'informations dans le secteur de la construction et de la gestion de patrimoine. En France, le chapitre français de l'IAI est l'Association MediaConstruct.

### Description générale

Les IFC (Industry Foundation Class) répondent au problème d'interopérabilité soulevé par l'IAI en proposant un format standard d'échanges pour les métiers de la construction. Ils représentent ainsi une alternative stable, pérenne et normalisée aux formats DXF/DWG qu'utilisent les logiciels commerciaux actuels comme AutoCAD©. Il s'agit d'une approche orientée objets de construction. Elle englobe non seulement les composants tangibles comme les portes, les murs,

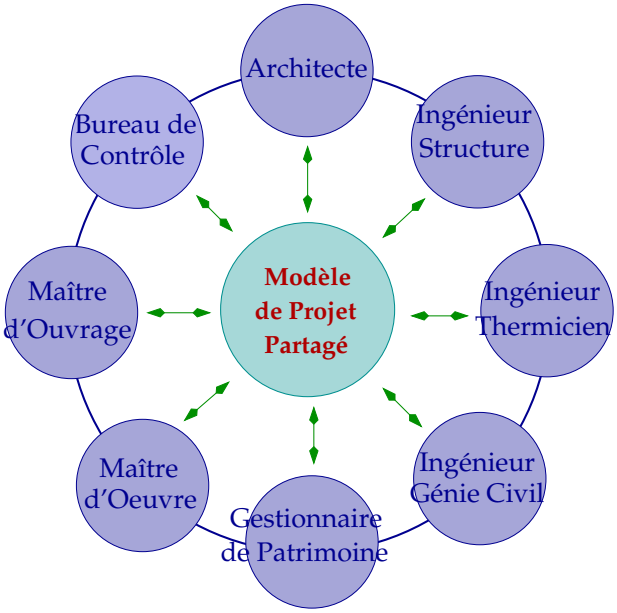


FIGURE 1.7 : Solution interopérable

les ventilateurs, etc. mais aussi les concepts abstraits comme les espaces, l'organisation, les processus, etc.

De plus, cette approche s'accorde avec le formalisme théorique des modèles d'information et de données abordé dans la partie 1.2.2. En effet, les IFC utilisent certaines parties de STEP car le modèle de données est spécifié en langage EXPRESS, conformément à la norme STEP-11. Les échanges d'instances se font grâce à des fichiers suivant le formalisme STEP-21. De plus, le modèle de données équivalent ifcXML [Nisbet05] permet d'échanger des instances IFC en XML, suivant le formalisme STEP-28. Enfin, la description géométrique des objets IFC est fortement inspirée de la norme STEP-42.

La première version IFC 1.0 a été créée en janvier 1997 et couvre les domaines suivants : la conception architecturale, le génie climatique, la maîtrise d'oeuvre et la gestion de patrimoine [Poyet97]. Viennent ensuite les versions 1.5, 1.5.1, 2.0, 2x, 2x2 et enfin 2x3 en février 2006.

L'architecture des IFC est très modulaire comme le montre la figure 1.8. [Liebich04] documente cette architecture et guide ainsi les implémenteurs de ce modèle. Voici une description succinte des concepts principaux de ce modèle.

Comme son nom l'indique, le module *Kernel* fournit toutes les structures de base du modèle :
- Des informations élémentaires : identification et propriétaire de l'objet.
- Des informations de relations entre objets : associations, agrégations.
- Des concepts de types et de propriétés liés aux objets.
- Des connexions vers des représentations géométriques et des localisations.

La figure 1.9 donne les premiers niveaux de hiérarchisation du noyau des IFC. L'entité

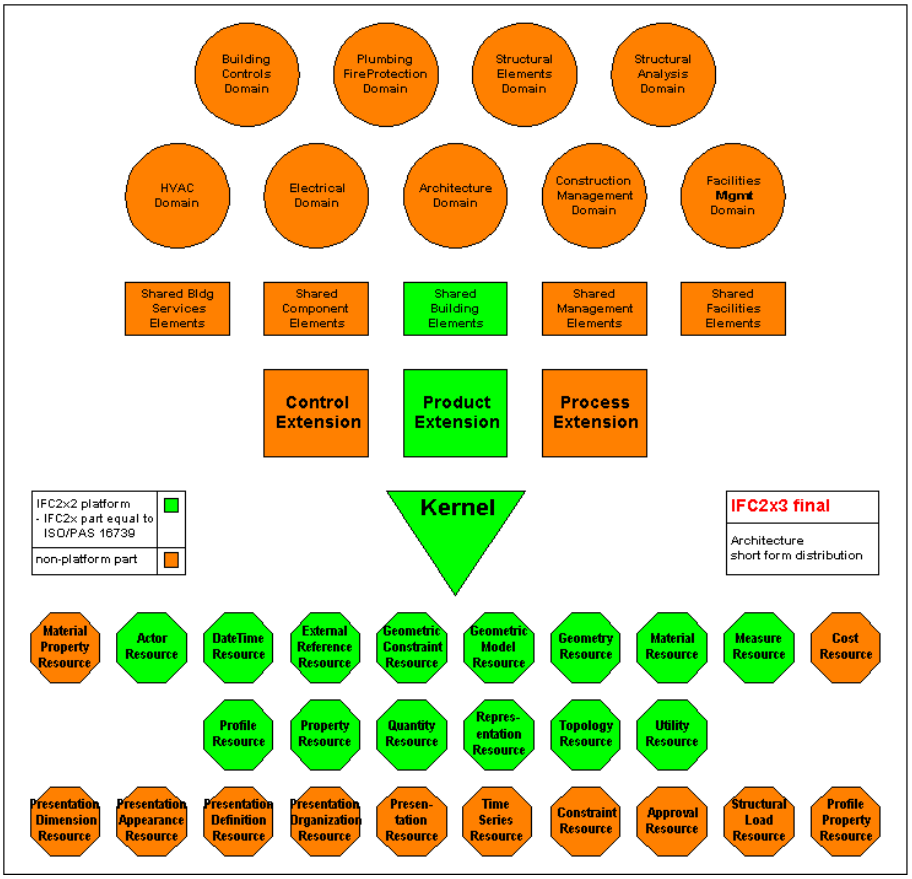


**FIGURE 1.8 :** Architecture générale des IFC (version 2x3)

`IfcRoot` est l'entité de base permettant d'identifier l'objet et de lui définir un propriétaire. L'entité `IfcObject` représente un objet au sens large du terme : objet physique, processus, tâche, acteur, etc. Un objet particulier est un produit, représenté par l'entité `IfcProduct`. Ce dernier représente véritablement un objet physique, et à ce titre, on peut lui adjoindre une représentation graphique et une localisation, sous la forme de propriétés de la classe `IfcProduct`.

La classe `IfcPropertyDefinition` permet d'adjoindre des propriétés particulières à des objets pour une utilisation métier précise. Il s'agit du mécanisme des *PropertySets* qui rend le modèle IFC plus flexible. Par exemple, un acousticien souhaiterait adjoindre des paramètres acoustiques avancés à un matériau, ce que ne prévoit pas le modèle de données IFC. Il va donc adjoindre des `PropertySets` personnels. L'échange de telles données nécessite donc de connaître la façon dont le spécialiste a encodé ses données, l'interopérabilité en est donc affaiblie. Cependant ces données très précises demandent rarement des échanges (sinon il faudrait étendre le modèle IFC avec de nouvelles classes) et permettent de centraliser l'information sur le modèle pour éviter le phénomène de dispersion d'information.

Enfin, les relations entre entités sont fortement typées et nécessitent de définir des classes intermédiaires de relations, héritant de `IfcRelationship`. Voici un autre exemple simple : un bâ-

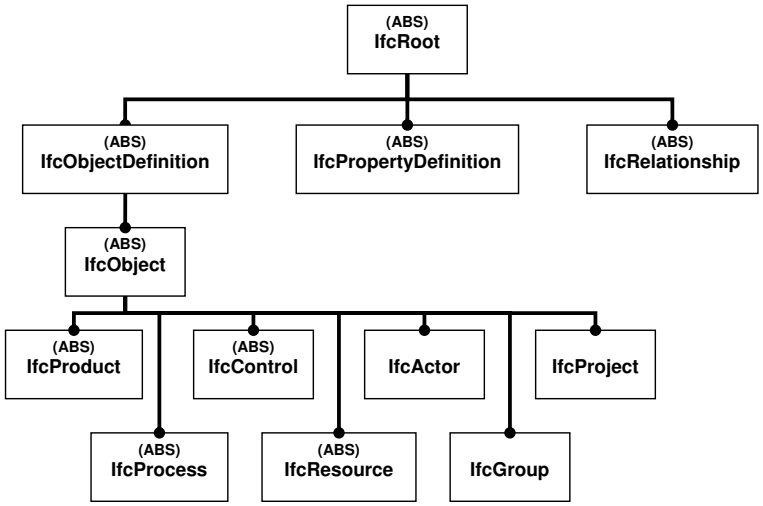


**FIGURE 1.9 :** Diagramme partiel EXPRESS-G illustrant les premiers niveaux de hiérarchisation du noyau des IFC2x3. Les liens représentés sont des liens d'héritage au sens de la POO. Seule la branche de `IfcObjectDefinition` a été sommairement développée. Les entités « (ABS) » sont dites abstraites, elles ne peuvent être instanciées directement car trop généralistes.

timent contient des étages. Une première approche consisterait à définir une classe `IfcBuilding` ayant une propriété correspondant à une liste d'étages, instances de la classe `IfcBuildingStorey`. L'inconvénient d'une telle modélisation est la pauvreté de typage du lien entre `IfcBuilding` et `IfcBuildingStorey`. L'approche IFC consiste donc à lier une instance du bâtiment avec des instances d'étage par l'intermédiaire d'une instance de la classe `IfcRelAggregates`. Le lien n'est plus direct mais fortement typé, il s'agirait ici d'un lien d'agrégation. La figure 1.10 illustre ce procédé.

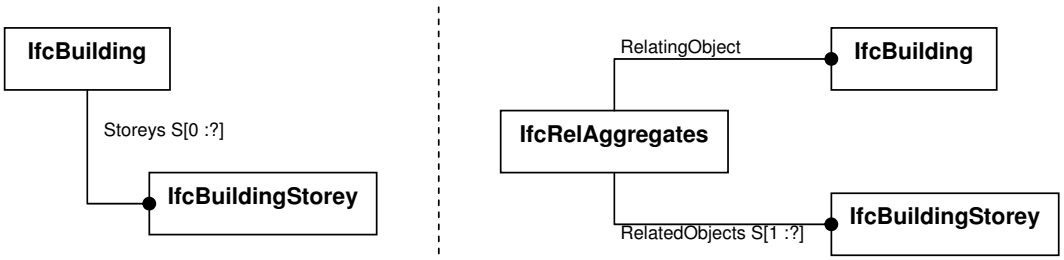


**FIGURE 1.10 :** Classes de relations intermédiaires entre entités IFC. A gauche, le bâtiment possède un attribut qui est une liste d'étages mais on ne sait pas quelle est la caractéristique de ce lien. A droite, grâce à l'utilisation d'une classe de relation IfcRel[...], on sait à présent que le lien précédent est une agrégation.

Le modèle IFC représente un modèle orienté objets de construction complet. Contrairement à l'AP 225, il connait une adoption croissante de la part des implémenteurs et des pôles Recherche et Développement de certaines sociétés de BTP.

**Des recherches actives autour de ce standard**

Les IFC suscitent beaucoup d'engouement, en premier lieu dans les conférences internationales et les pôles de rencontres entre industriels. Nous pouvons notamment citer les conférences ECPPM (eWork and eBusiness in Architecture, Engineering and Construction) et CIB-W78 (International Council for Building Research Studies and Documentation, W78 correspond au pôle « IT in Construction »), les journaux scientifiques *Automation in Construction* (chez Elsevier) et *Journal of Computing in Civil Engineering* (chez ASCE), etc. Une partie des communications qui en découlent traitent de l'exploitation des IFC, du modèle, des processus, de la persistance, etc. Nous n'allons donc pas citer ici tous les articles traitant de l'amélioration et l'utilisation des IFC mais seulement aborder plusieurs projets représentatifs, en lien avec la modélisation de données et leur persistance.

Les premières pistes de développement consistent à faciliter l'accès et la persistance des modèles de bâtiments, sous forme de serveurs Web [Cruz02] [Chen04]. Nous pouvons citer en particulier deux projets majeurs :

- Active 3D [Cruz02] : il s'agit d'un serveur IFC permettant de conduire une conception de projet de bout en bout, avec des mécanismes d'authentification, de notifications automatiques, etc. Cela permet de rendre la maquette numérique plus robuste en termes de responsabilité pour les acteurs de projet. L'application spécifique à la conception est Active3D-Build (cf. figure 1.11).
- Projet SABLE [BLIS-Project05] : le projet *Simple Access to the Building LifeCycle Exchange* a pour objectif de séparer nettement le modèle de données des interfaces. En effet, pour améliorer l'accès aux données d'un modèle lourd, on crée différentes interfaces spécialisées pour un métier donné, des *vues* qui simplifient grandement l'accès aux données pertinentes pour un type d'acteur donné. En interne, ces vues traduisent le modèle (IFC par exemple) en un modèle spécialisé, un format dit *pivot* qui ne contient que les informations nécessaires à la vue concernée. La figure 1.12 illustre l'architecture de ce projet.

Une autre piste consiste plutôt à proposer des extensions aux IFC pour des objectifs variés. Certains travaux tentent de donner plus de fonctionnalités ou d'intelligence aux objets [Halfawy02], d'autres essayent d'étendre les IFC en modélisant les processus de manière avancée [Hassanain01] ou en les associant à de nouvelles méthodes de conception comme la CAO 4D [Tanyer04]. Ces recherches dépassent le cadre de la définition même de la MNC. Dans la section 1.3, nous analyserons en particulier les innovations sur des aspects de synchronisation. Plus généralement, nous constatons que l'alternative la plus aboutie et la plus utilisée pour des développements innovants est le modèle IFC, pour le cadre bâti. Concernant le domaine des ouvrages d'arts et en particulier les ponts, il n'existe pas de modèles de maturité équivalente. Il existe néanmoins plusieurs développements spécifiques.

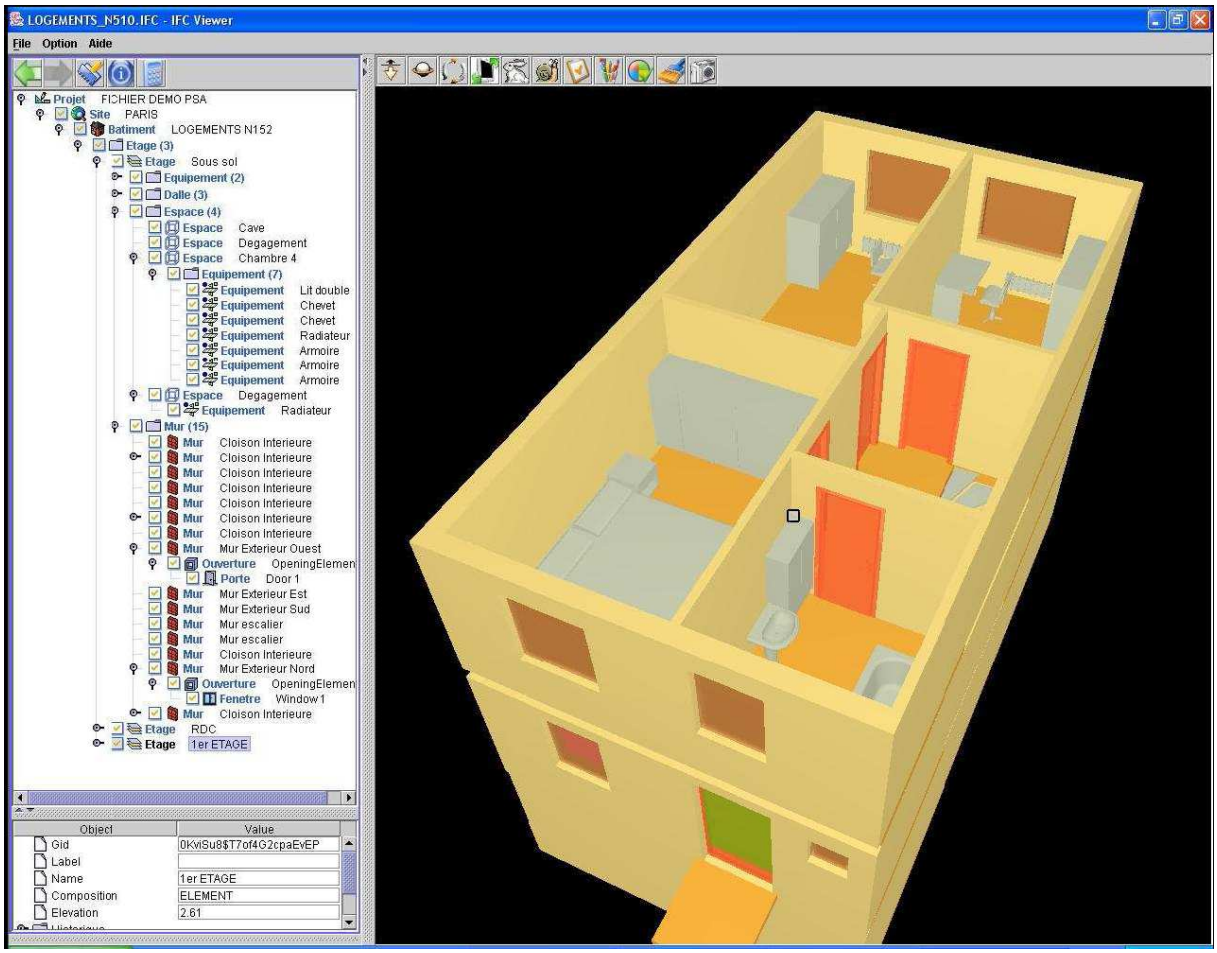


**FIGURE 1.11 :** Allure de l'interface de Active3D-Build, d'après [Cruz04],[Vanlande03] et [Vanlande03]

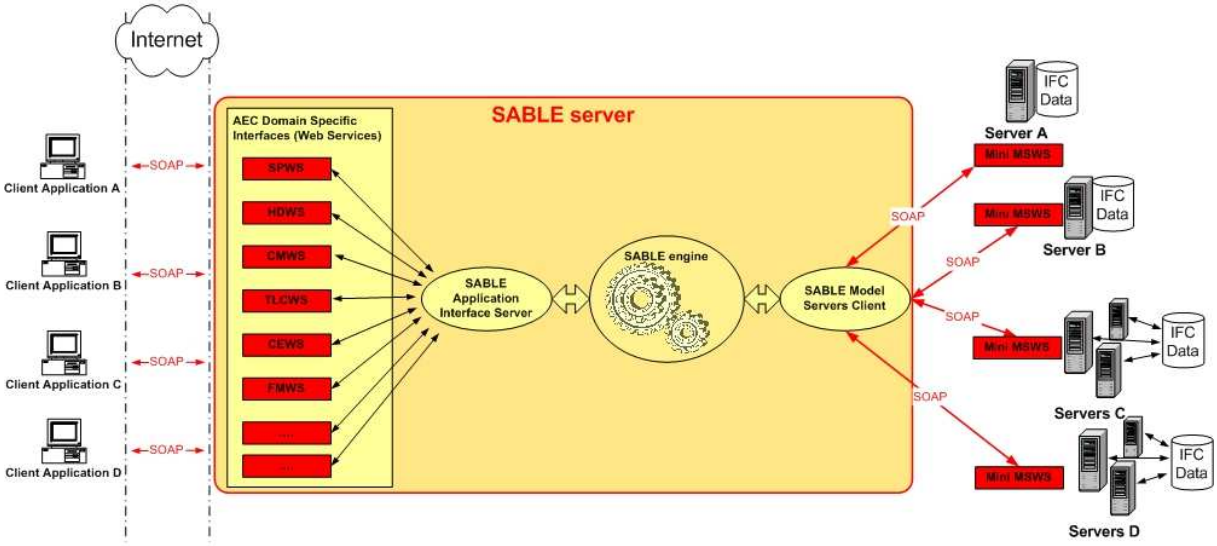


**FIGURE 1.12 :** Architecture proposée pour la plate-forme SABLE

### 1.2.4 Développements spécifiques aux ponts : de OA-Express à IFC-Bridge

**Première utilisation de STEP pour les ponts : OA-Express**

Le SETRA[1] a créé le modèle OA EXPRESS (OA pour Ouvrage d'Arts) en partenariat avec la SNCF, Dumez/GTM, Campenon-Bernard, Spie-Batignolles, Baudin-Chateauneuf et Bouygues [Medi@construct04]. La division CTOA du SETRA a ainsi développé une interface adaptée avec son logiciel de calcul de structure PCP. Il s'agit de la définition d'un modèle de donnée STEP-11 pour la structure d'un ouvrage d'art. Ce projet ne s'inscrit cependant pas dans un contexte de standardisation internationale. En effet, les entités définies sont de langue française, et aucune collaboration avec des partenaires étrangers n'a eu lieu pour harmoniser les types de données nécessaires et les méthodes pour les manipuler. C'est pourquoi le projet a été confié au chapitre francophone de l'IAI pour proposer une standardisation internationale, par exemple via une extension du modèle IFC. Cette proposition, coordonnée aussi par les chapitres Nord Américain et Australien, fut acceptée à l'unanimité lors de l'ITM (International Technical Meeting) à Munich le 18 janvier 2002. Le projet Bridge, aussi appelé IFC-Bridge est né.

**Vers un modèle unifié avec le bâtiment : IFC-Bridge**

L'objectif du projet IFC-Bridge est de définir une extension des IFC pour les échanges de données de conception des ouvrages d'arts. Il permet de couvrir les domaines géométriques et technologiques suivants (la liste n'est pas exhaustive) :

- Définitions géométriques des tabliers, piles, pylônes, câbles, haubans, liaisons.
- Définition des aciers passifs et des câbles de précontrainte.
- Définitions des matériaux
- Définitions liées au phasage.
- Définitions des paramètres nécessaires à des analyses de structure.
- Définition des superstructures : garde-corps, trottoirs, chaussée, bordures et équipements divers.
- Définitions des implantations sur site.

Une première version (sans la définition de la précontrainte) de IFC-Bridge a été disponible fin 2003, et la version Ifc-Bridge V2 fin 2005. Cette dernière version fut l'objet d'un partenariat franco-japonais financé par l'EGIDE dans le cadre du programme SAKURA [Yabuki06]. Il s'agit du « développement d'un modèle 3D pour l'étude d'ouvrages de génie civil en environnement CAO et réalité virtuelle enrichie » . L'objectif de cette collaboration est de permettre aux deux équipes de se rencontrer afin de définir un modèle de données en commun et de contribuer à sa promotion internationale, notamment en Europe et en Asie. Les deux équipes ont l'avantage d'être complémentaires. En effet, le thème de recherche sur la modélisation des ouvrages de génie civil est commun, mais avec des sensibilités différentes selon les aspects culturels (Europe-Asie) et les aspects statutaires (EPIC-université). Dans le cadre de cette thèse, ce partenariat a notamment permis de fournir un cas d'étude simple. L'annexe A détaille la structure de la dernière version à l'heure où sont écrites ces lignes avant son intégration à la prochaine révision du

[1] Service d'Études sur les Transports, les Routes et leurs Aménagements

modèle IFC.

Plus généralement, l'évolution des modèles de données pour la construction, génie civil ou bâtiment, a suivi les deux phases de l'informatisation (cf. section 1.1) : d'abord, les standards ont surtout permis de servir de supports aux outils de CAO, et les échanges se sont limités aux données graphiques : IGES et DXF par exemple. Ensuite, les acteurs ont eu besoin d'échanger plus d'informations. Ceci a induit une centralisation globale des données d'un projet, grâce à STEP, et en particulier aux IFC et IFC-Bridge pour la construction. Il s'agit donc de l'amorçage de la deuxième phase d'informatisation, c'est-à-dire faciliter l'accès et l'enrichissement des données. Le but est alors d'améliorer la productivité grâce à une collaboration optimale entre acteurs de projets, en particulier dans le cadre de l'ingénierie concourante. C'est pourquoi les modèles de données centralisées telles que les MNC nécessitent des mécanismes favorisant l'accès concurrentiel aux données, par exemple la synchronisation.

## 1.3 Méthodes courantes pour la synchronisation de modèles d'ouvrages

Dans un projet utilisant la conception collaborative, un ***modèle d'ouvrage*** centralisé de conception subit tout au long du processus de conception des mécanismes de synchronisation. Dans un environnement où plusieurs acteurs peuvent modifier en même temps différentes parties du modèle, la synchronisation prend en compte tous ces changements pour mettre à jour et enrichir le modèle de façon cohérente, c'est-à-dire sans information contradictoire ou n'ayant pas de sens. Pour cela, nous avons choisi d'analyser la synchronisation à travers deux étapes principales : la comparaison des données et l'obtention du modèle mis à jour. Dans le cadre de la synchronisation de modèles, la comparaison des données s'effectue entre le modèle d'ouvrage *original* et un modèle d'ouvrage *modifié*. Si plusieurs acteurs modifient le modèle d'ouvrage original, plusieurs modèles modifiés co-existent. Tous les résultats de comparaison sont alors utilisés pour obtenir un nouveau modèle d'ouvrage cohérent. C'est l'objet de la deuxième étape. Cette partie vise donc à décrire les techniques courantes de comparaison en informatique (section 1.3.1), avant de détailler les méthodes de mises à jour du modèle d'ouvrage basées sur des techniques de versionnement (section 1.3.2), pour enfin aborder les transformations de modèles afin d'améliorer en amont le processus de synchronisation (section 1.3.3).

### 1.3.1 De la comparaison de fichiers à la comparaison structurelle

#### La comparaison de fichiers texte

En informatique, la comparaison d'informations s'est longtemps focalisée sur une comparaison de fichiers, en particulier les fichiers texte. Le plus connu à ce titre reste l'outil libre du monde UNIX : GNU *diff* [FSF02]. L'algorithme utilisé est celui de la plus longue sous-séquence commune (LCS en anglais) [Myers86]. Les applications permettant de versionner les fichiers sources d'un projet informatique, comme CVS ou Subversion, utilisent GNU *diff* pour détecter les différences entre deux versions [Cederqvist06] [Pilato04]. La figure 1.13 montre un exemple

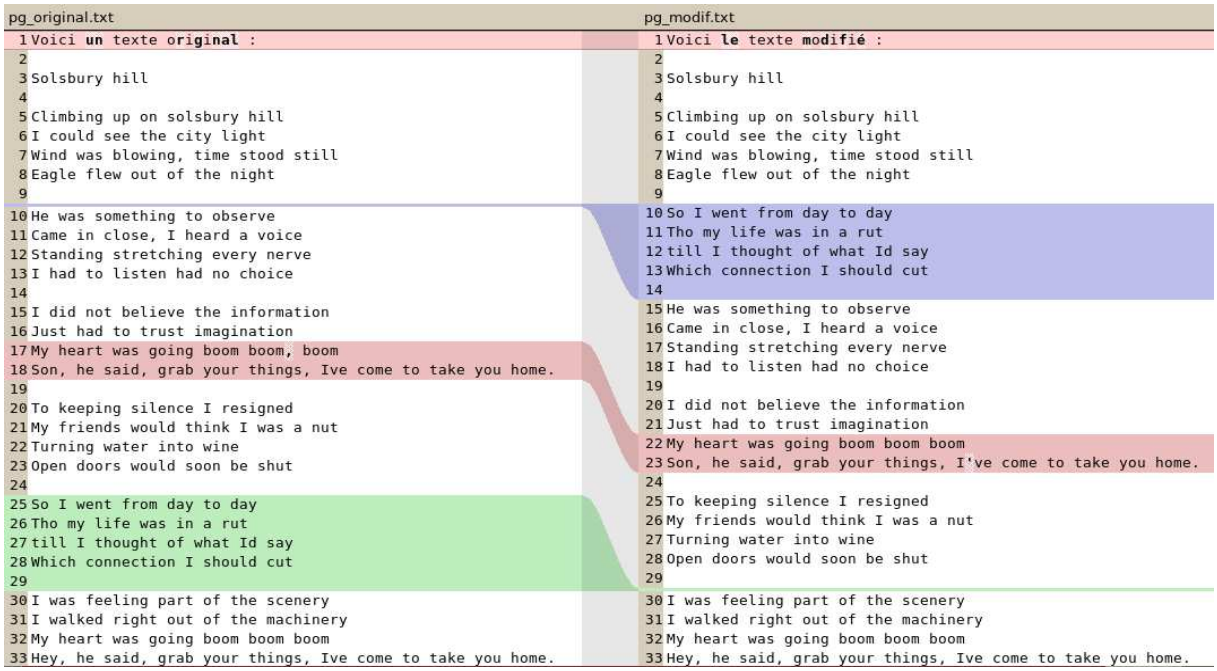


**FIGURE 1.13 :** Illustration de l'interface *kompare* utilisant l'algorithme LCS.

```
<Bibliothèque>
  <Livre>
    <Titre>Conception des ponts</Titre>
    <Auteurs>
      <Auteur>Anne Bernard-Gély</Auteur>
      <Auteur>Jean-Armand Calgaro</Auteur>
    <Auteurs>
  </Livre>
  <Livre>
    <Titre>The War of The Worlds</Titre>
    <Auteurs>
      <Auteur>Herbert George Wells</Auteur>
    </Auteurs>
  </Livre>
  <Thèse>
    <Auteur>Guillaume Picinbono</Auteur>
    <Titre>Modèles géométriques et physiques
pour la simulation d'interventions chirurgicales</Titre>
  </Thèse>
</Bibliothèque>
```

**FIGURE 1.14 :** Exemple de fichier XML

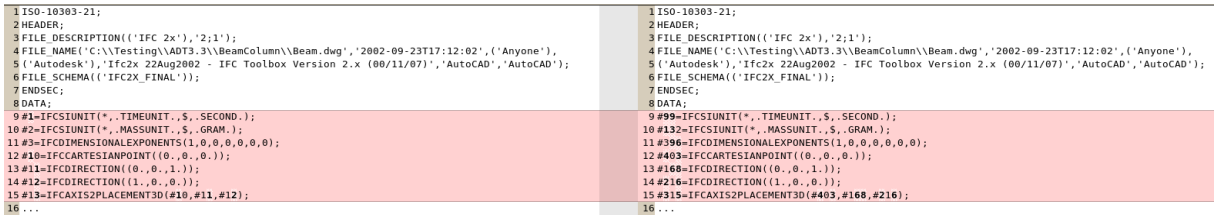


**FIGURE 1.15 :** Comparaison avec *kompare* de deux structures STEP-21 identiques.

d'application utilisant l'algorithme LCS pour la comparaison de fichiers texte. Les résultats de comparaison issus d'un outil comme GNU *diff* attribuent généralement trois types de statut à chaque ligne de fichier texte :

- **Modifié** : ce statut indique que la ligne a été modifiée entre l'ancienne version et la nouvelle. Sur la figure 1.13, ce sont les lignes surlignées en rouge. Ce statut concerne aussi bien la ligne dans son ancienne version que dans sa nouvelle.
- **Supprimé** : ce statut indique qu'une ligne de l'ancienne version du fichier n'existe plus dans la nouvelle. Sur la figure 1.13, ce sont les lignes surlignées en vert. Ainsi ce statut ne concerne que les lignes de fichier associées à l'ancienne version.
- **Ajouté** : ce statut indique qu'un ligne de la nouvelle version n'existait pas auparavant. Sur la figure 1.13, il s'agit des lignes surlignées en bleu. On remarque de la même façon que les lignes de fichier concernées par ce statut se trouvent seulement dans la nouvelle version.

Dans ce mode de comparaison, il n'est pas possible de détecter un « déplacement » de l'information. En effet, une ligne ou un paragraphe déplacé dans un document sera détecté comme *supprimé* dans l'ancienne version et *ajouté* dans la nouvelle. Par exemple, sur le texte de la figure 1.13, le déplacement du paragraphe a été interprété comme un couple suppression/ajout.

Une deuxième limite dans ce type d'algorithme réide dans son incapacité à manipuler des structures hiérarchisées sauvées sous forme de fichier texte (sérialisées) [Chawathe96]. En effet, nous pouvons notamment distinguer deux contextes où l'algorithme LCS donne des résultats incorrects :

1. **La gestion de données encapsulées par des tags :** Lors de la sérialisation de données structurées, la méthode la plus usuelle consiste à placer des tags (symboles ou mots-clefs réservés) au niveau de chaque segment de données pour indiquer le contexte et son niveau hiérarchique. C'est l'approche suivie par le langage XML [Bray04], comme le montre la figure 1.14. Un moteur de comparaison basé sur le texte pur sera incapable de différencier le contenu des tags de structure et peut ainsi comparer des éléments n'appartenant pas au même contexte [Chawathe96].

2. **Multiplicité des sérialisations pour une structure donnée :** Rien ne garantit qu'un même ensemble de données structuré aura une et une seule sérialisation possible. En effet, le standard STEP-21, vu à la section 1.2.2, identifie chaque noeud du graphe à un numéro unique. Le choix de ce numéro est totalement libre. Si l'on reprend l'exemple de la figure 1.4, nous sommes libres de numéroter autrement les noeuds du graphe. Nous obtenons alors un fichier texte différent même si les données structurées sont identiques. Les labels `#<nombre>` sont analogues à des variables muettes en mathématiques. La figure 1.15 montre ainsi que l'outil GNU *diff* est incapable de détecter l'équivalence entre deux sérialisations d'une même structure.

Nous pouvons donc en conclure que la comparaison de fichier texte ne saurait être adaptée dans le cadre de modèles d'ouvrages. Ces derniers sont fortement structurés à cause de la quantité et de la complexité de l'information contenue.

### La correspondance de graphes

La comparaison de structures est traitée par le domaine de la *Modélisation et la Recherche Opérationnelle* (MRO) puisque le modèle d'un ouvrage est associable à un graphe orienté. Le but serait donc de comparer les deux versions d'un modèle via leur graphe associé. Contrairement aux sérialisations, on s'assure ici que le passage du graphe de données au modèle de l'ouvrage est bijectif, c'est-à-dire que tout modèle d'ouvrage possède un graphe associé et que si deux modèles sont équivalents alors les graphes associés sont les mêmes. Une littérature assez riche aborde les algorithmes de comparaison d'arbres et de graphes.

Tout d'abord, le problème connu de correspondance de sous-graphes, « subgraph matching » en anglais, consiste à répondre à la question suivante : à partir des graphe $G_1$ et $G_2$, est-ce que $G_1$ est isomorphe à un sous-graphe de $G_2$ ? Ce problème est reconnu pour être NP-complet [Garey90]. Cela signifie que n'importe quel algorithme répondant de manière générale à cette question a une complexité au moins exponentielle[2]. Le problème de correspondance de graphe (est-ce que $G_1$ est isomorphe à $G_2$ ?) n'a pas été démontré comme étant NP-complet mais nous n'avons pas trouvé dans l'étude bibliographique d'algorithme performant, c'est-à-dire en temps polynomial, pour répondre à cette question [Foggia01]. Néanmoins, en restreignant ou en prétraitant la structure du graphe de données, la complexité peut-être améliorée dans certains cas [Corneil70]. Aujourd'hui, si l'algorithme de Ullmann [Ullmann76] est encore le plus utilisé, il existe de nombreuses alternatives comme celui proposé par Schmidt et Druffel [Schmidt76] ou plus récemment l'algorithme « VF » [Cordella99]. Foggia [Foggia01] montre ainsi que l'on peut obtenir de très bons résultats, avec un temps de résolution de l'ordre de la seconde pour des graphes à mille noeuds pour l'algorithme « VF ».

Dans tous les cas, la correspondance de graphes répond juste par oui ou par non et n'indique pas quels changements ont eu lieu entre le graphe $G_1$ et le graphe $G_2$. C'est pourquoi les algorithmes précédents ne peuvent s'appliquer directement à la comparaison de modèles d'ouvrage. De plus la correspondance de graphes s'attache surtout à la structure même des données, c'est-à-dire l'ensemble des liens entre les noeuds, au détriment des noeuds eux-mêmes. Il s'agit d'une deuxième grande limitation. Nous verrons dans le chapitre suivant que la sémantique d'un modèle d'ouvrage se situe dans les noeuds du graphe, beaucoup plus que dans les liens. Il s'agit d'une des hypothèses fondamentales de ce travail de thèse.

Si l'on réussit à transformer le graphe associé au modèle d'ouvrage en un arbre, procédé qui sera vu au prochain chapitre, nous pouvons alors explorer d'autres pistes de solutions existants en MRO : la comparaison de structures d'arbres.

### La comparaison d'arbres

Au-delà du problème de la correspondance d'arbres, qui n'est qu'un cas particulier de la correspondance de graphes, le problème de comparaison d'arbres évalue la différence entre deux

[2] A moins de démontrer que $P = NP$, problème d'informatique théorique encore ouvert. Aujourd'hui la conjecture tend plutot à affirmer que $P \neq NP$. Le lecteur pourra se référer à [Garey90].

structures données : soit $A_1$ et $A_2$ deux arbres. Le but est de trouver la succession d'étapes élémentaires pour transformer l'arbre $A_1$ en l'arbre $A_2$ (changement de noeud, effacement, insertion). Ce problème est plus connu sous le nom *Tree Edit Distance* en anglais. Plusieurs algorithmes existent et ont été comparés [Bille03]. Cette fois, les noeuds ont des noms (le changement du contenu d'un noeud est ainsi détectable), et les algorithmes les plus performants supposent que l'arbre soit ordonné. Ceci signifie qu'à un noeud donné, les fils sont exposés dans un seul ordre possible. Il s'agit d'une contrainte importante dans la mesure où l'ordre d'apparition des fils peut avoir une importance sémantique. Ainsi la figure 1.16 montre que deux arbres différents *a priori* deviennent équivalents avec l'hypothèse des arbres ordonnés.

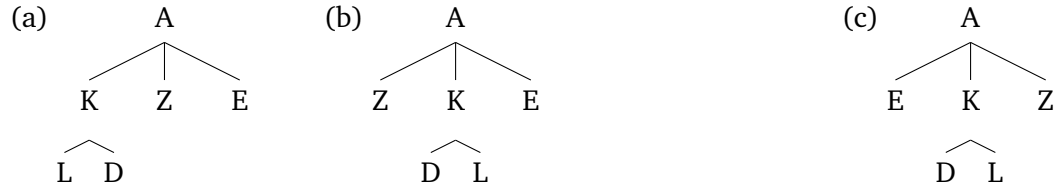


**FIGURE 1.16 :** Obtention d'un même arbre ordonné à partir de deux sources différentes. A gauche (a) et (b) sont deux arbres différents. A droite, l'hypothèse des arbres ordonnés permet de définir une classe d'équivalence dont le représentant (c) devient équivalent à (a) et (b). Il suffit pour cela de trier les noeuds frère par ordre alphabétique.

Les applications de la comparaison d'arbres sont nombreuses car la plupart des structures hiérarchisées de données sont exprimables sont forme d'arbres. Des moteurs de comparaisons se sont notamment spécialisé dans certains formats comme XML avec XyDiff [Cobena02] ou encore X-Diff [Wang03]. Généralement cette comparaison correspond à l'enchainement de deux étapes :

1. **Prétraitement des données source :** La première étape consiste à charger la structure en mémoire à partir d'un fichier. Cette structure peut être ensuite optimisée selon le type de données ou le type de comparaison (élagage, ordonnancement des noeuds, etc...). Par exemple des techniques de ***hashing*** des noeuds sont généralement utilisées pour optimiser la comparaison de sous-arbres entiers, comme DOMHash [Maruyama00] et X-Hash [Wang03] dans le cadre de la comparaisons de fichiers XML.
2. **Mise en correspondance et comparaison :** Ensuite, l'algorithme de mise en correspondance permet de calculer le passage de la première structure à la deuxième en un nombre minimal d'étapes. En effet, il existe plusieurs chemins possibles pour passer d'un arbre à l'autre et il convient de trouver le chemin optimal, cf. figure 1.17. Pour cela, il convient de définir les opérations autorisées de transformation (renommage d'un noeud, ajout et suppression d'un noeud ou d'un sous-arbre, etc.), et leur coût à l'unité, pour le calcul du chemin optimal. Les hypothèses pour restreindre le problème (arbres ordonnés, comparaison de noeuds de même niveau, hashing des sous-arbres) sont alors nécessaires car le problème général de comparaison d'arbres non ordonnés est NP-complet [Zhang92].

Lors de la synchronisation de modèles d'ouvrages, la comparaison permet de détecter les changements et leur nature. Le versionnement va plus loin en stockant les changements sur une

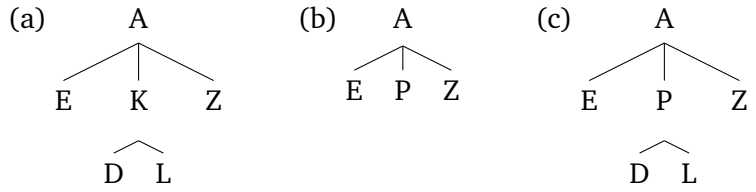


**FIGURE 1.17 :** Chemins de transformation d'un arbre à un autre. Le but consiste de transformer (a) en (c). Il y a deux possibilités. La première consiste à supprimer le sous-arbre de racine K, on obtient alors l'arbre (b), de créer un fils P de A et d'y insérer deux fils D et L. La deuxième possibilité consiste simplement à renommer le noeud K en P. La deuxième possibilité représente donc un chemin plus court.

base de données, et surtout, en assurant la cohérence du modèle lors de son évolution.

### 1.3.2 Aboutissement de la comparaison : le versionnement d'objets

Le versionnement est un mécanisme qui permet non seulement de reconstruire un modèle d'ouvrage $M_n$ à partir du modèle $M_{n-1}$ antérieur et d'un ensemble de modifications élémentaires, mais qui permet aussi de proposer plusieurs variantes d'un modèle d'ouvrage durant le processus de conception [Urtado98]. Le versionnement d'objets s'applique généralement aux systèmes à objets. Un rappel de certains concepts fondamentaux permet de comprendre les différentes approches choisies par les méthodes de versionnement existantes.

#### Les systèmes à objets versionnables

Les concepts de modèle d'information ont été définis dans la section 1.2.2, modèle de données, et instance de modèle. Lorsque l'on étudie la pérennité des données lors d'un projet, les modèles de données et les instances peuvent être manipulés via des systèmes à objets, en particulier versionnables. en effet, tout au long du projet ce système est appelé à évoluer. C'est pourquoi ils intègrent, pour la plupart, à la fois le modèle de données et ses instances. En effet, ils prennent en compte l'évolution des modèles d'ouvrages (les instances) mais aussi l'évolution possible du modèle de données lui-même. Les données et les structures sont donc suivies de manière cohérente. De nombreux systèmes de gestion de données persistantes intègrent la définition du modèle de données et des notions de version, comme *Encore* [Skarra87], *ObjectStore* [Lamb91], *Versant* [Clair94] ou encore *Version Server* [Katz86].

Pour comprendre le fonctionnement global de tels systèmes, voici quelques définitions.

**Définition 6.** *On appelle entité tout concept, abstrait ou concret, modélisé grâce à un modèle de données ou à une de ses instances.*

De plus, une instance n'existe que si elle est associée à un (ou plusieurs parfois) modèle de données. [Urtado98] distingue deux types d'entités.

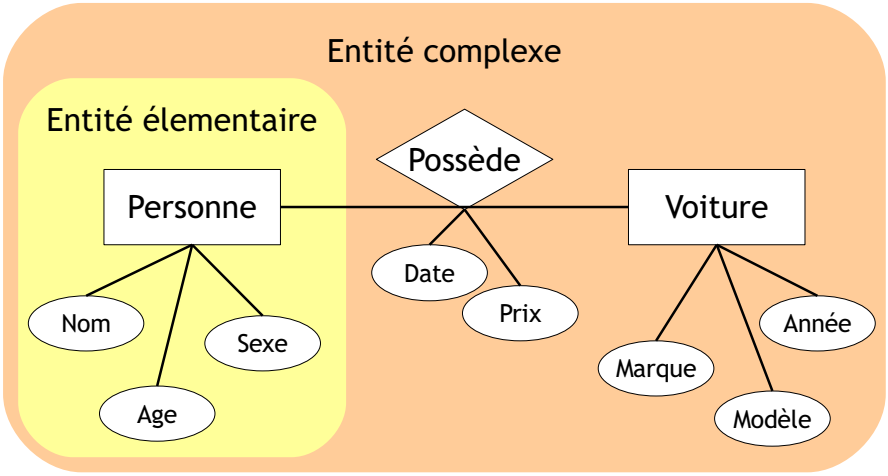


**FIGURE 1.18 :** Exemple d'entité élémentaire et d'entité complexe dans le cadre de la modélisation E/R. Une personne caractérisée par des attributs simples que sont le nom, l'age et le sexe peut être considérée comme une entité élémentaire. La voiture ayant d'autres attributs simples peut elle aussi être considérée comme une entité élémentaire. Ces deux entités sont liées par un lien sémantisé, la possession. Ce lien possède lui aussi des attributs propres, nommés attributs de liens en modélisation E/R. L'ensemble formé par les deux entités et le lien sémantisé est appelé entité complexe.

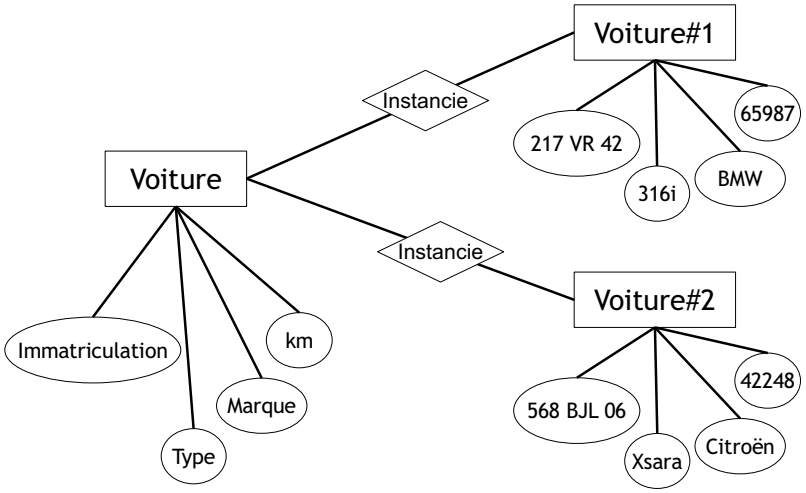


**FIGURE 1.19 :** Modélisation d'une classe d'entité liée à ses instances. Les deux objets sont des entités. Parmi elles, `Voiture` est une classe d'entités, `Voiture#1` et `Voiture#2` sont des instances. Le lien d'instanciation oblige `Voiture#1` et `Voiture#2` à posséder une immatriculation, un type, une marque et un kilométrage.

**Définition 7.** *Une entité est dite élémentaire lorsque l'on considère sa description propre, c'est-à-dire l'ensemble de ses attributs, en ignorant ses relations avec d'autres entités. Les attributs peuvent être simples (nombres entiers, réels, chaînes de caractères), composés (point 3D composée de trois nombres réels x,y et z), ou multi-valués (liste quelconque de nombres réels).*

**Définition 8.** *Une entité est dite complexe lorsqu'elle est associable à un graphe dont les noeuds sont des entités élémentaires et les liens sont des relations de dépendance.*

Les relations de dépendance ont souvent une signification bien particulière, appelée sémantique de relation. La figure 1.18 illustre les concepts définis précédemment. Un type de relation particulier permet de relier une classe d'entité (celle du modèle de données) à une instance d'entité, appelé relation d'instanciation, comme l'illustre la figure 1.19. Ce lien force les attributs des instances d'entité à suivre la structure de sa (ou ses) classe(s) associée(s).

Le rassemblement du modèle de données et de ses instances au sein du même système permet d'adopter une perspective conceptuelle pour le problème de l'évolution. Il est notamment nécessaire pour la prise en compte des effets de l'évolution du modèle de données sur les instances [Nguyen89], [Clamen92].

Le versionnement permet de détecter et suivre l'évolution des entités. Il existe ainsi deux types de versionnement :

- **Le versionnement d'entités élémentaires** : il permet de stocker les modifications appliquées à une entité élémentaire, sans prendre en compte ses liens avec les autres objets d'un modèle.
- **Le versionnement d'entités complexes** : il permet de stocker les modifications appliquées à une entité complexe, c'est-à-dire un ensemble constitué de l'entité élémentaire et de ses dépendances : d'autres entités ainsi que les liens entre ces entités.

Le versionnement d'entités complexes est plus difficile, notamment pour des questions de cohérence : quand différentes versions de chaque objet existent, que deviennent les relations entre ces objets [Borhani92] ? L'approche classique pour résoudre ce problème consiste à formaliser la répercussion d'une modification sur une entité élémentaire vers ses dépendances. Ce mécanisme s'appelle la propagation de versions [Rumbaugh88].

**Définition 9.** *La propagation est l'application automatique d'une opération à un réseau d'objets quand l'opération est appliquée à un objet quelconque.*

Un système à objets versionnable peut être alors vu comme un support ***persistant*** pour un ensemble structuré d'entités complexes, muni d'une gestion de versions, avec notamment un mécanisme de propagation de versions. Pour obtenir une base de données homogène et cohérente, la version d'une entité peut-être aussi considérée comme une entité. Un nouveau type de relation est alors créé afin de relier une entité à une version d'entité : la relation de dérivation.

Selon l'application et/ou les données à manipuler, [Urtado98] distingue deux approches complémentaires pour répondre au problème du versionnement : l'approche microscopique et l'approche macroscopique.

### Approches microscopique et macroscopique du versionnement

Une première possibilité de versionnement consiste à considérer chaque entité complexe comme un graphe d'entités élémentaires, comme cela a été fait à la section précédente. Chacune de ces entités (instances ou classes) est versionnée. Il s'agit d'une approche microscopique, aussi appelée versionnement *local* [Borhani92]. Ce type de versionnement considère les objets dans leur granularité la plus fine, la gestion des relations et le mécanisme de propagation des changements sont donc appliqués au niveau des relations élémentaires (les liens du graphe). Ce type de versionnement résout alors deux problèmes principaux :

- **La gestion des relations entre entités de version différente** : soit $r$ une relation liant deux entités élémentaires $A$ et $B$. Si $A'$ et $B'$ dérivent respectivement de $A$ et $B$, que devient $r$ ? Reformulons la question autrement : quelle relation faut-il créer entre $A'$ et $B'$ ? Les solutions existantes proposent soit de *dupliquer* la relation $r$, une nouvelle relation $s$ liant alors $A$ et $B$ [Skarra87], soit de *versionner* la relation $r$, une version $r'$ de $r$ est créée pour lier $A'$ et $B'$ [Katz86]. [Urtado98] définit aussi deux autres modes : *l'évolution séparée* et *la relation dynamique*. Chacun présente des avantages et des inconvénients selon l'utilisation faite de ces supports.
- **la propagation du versionnement d'une entité** : en reprenant l'exemple précédent, si seulement $A'$ dérive de $A$, le fait que $r$ lie $A$ à $B$ peut contraindre $B$ à créer automatiquement une version $B'$. Il s'agit du mécanisme de propagation de versions. Donc selon la sémantique de la relation (composition, référence, équivalence, etc ...), le système peut adopter une propagation sélective ou bien systématique. Par exemple Version Server [Katz86] adopte une propagation systématique des relations de composition.

Les applications à ce type de versionnement sont nombreuses. Par exemple, une application CAO a souvent besoin de comparer plusieurs solutions potentielles pour la conception d'un objet au niveau le plus fin, pour chaque entité élémentaire.

Dans de nombreux secteurs, un modèle de données servant à la conception de projets devient très complexe. Le modèle IFC 2x3 possède ainsi plus de 600 entités élementaires. C'est pourquoi l'approche microscopique peut s'avérer très contraignante si le versionnement se fait au niveau des entités élémentaires. Une autre méthode consiste donc à introduire un versionnement haut-niveau des entités complexes. On introduit pour cela la notion de conteneur :

**Définition 10.** *Un conteneur est une entité complexe regroupant des entités élémentaires dépendantes. Le versionnement d'un conteneur permet de versionner conjointement ses entités constituantes.*

Les entités et les relations au sein d'un conteneur sont alors appelées constituants de ce conteneur. Il s'agit d'une approche dite macroscopique, ou versionnement global [Borhani92]. Un système supportant ce type de méthode permet de surmonter les difficultés suivantes :

- **Identification d'un conteneur** : selon le modèle de données, le système est capable de caractériser les conteneurs, en fonction des relations existantes, des paramètres utilisateurs, etc ...
- **Relations entre versions d'entités constituantes** : si les entités se voient subir un versionnement cohérent au sein du conteneur, la gestion des relations (ou références) entre

versions est définie par le système, comme pour l'approche microscopique.

- **Création d'une version de conteneur** : cette gestion suppose d'avoir identifié les circonstances de création d'une version de conteneur, en fonction notamment des relations entre entités de conteneurs différents, ainsi que de connaitre les effets de la création d'une version de conteneur sur ses constituants.

L'approche macroscopique convient par exemple en génie logiciel où les programmes ou bibliothèques sont vus comme des boites noires par les utilisateurs, en suivant les versions globales, sans s'attacher au versionnement de chaque fichier source.

Le versionnement aide le processus de conception en assurant une traçabilité et en proposant des versions alternatives d'un même produit à concevoir. Cependant lorsque le modèle de données sous-jacent est un standard d'échange, ce dernier peut se révéler trop complexe pour un projet particulier. Par exemple, les projeteurs utiliseront moins de concepts et moins de types d'entités pour un projet de maison individuelle par rapport à un bâtiment industriel, pour un pont-dalle poussé par rapport à un viaduc haubané. C'est pourquoi les mécanismes de comparaison vus dans la section précédente et ceux de versionnement vus dans cette partie deviennent superflus voire inproductifs s'ils se basent directement sur le standard d'échange. Pour répondre à cela, il existe des solutions de transformations de modèles et de comparaison sélective.

### 1.3.3 Transformations de modèles et comparaison sélective

**Définition 11.** *La transformation (mapping en anglais) d'un modèle de données $A$ en un modèle de donnée $B$ est une fonction qui associe à n'importe quel modèle d'ouvrage $a$, instance de $A$, un modèle d'ouvrage $b$, instance de $B$.*

Ce mécanisme permet de mettre en relation des modèles de données, afin d'échanger des informations issues de standards différents. Il permet aussi d'extraire d'un modèle de données complexe et générique un ensemble d'informations spécifique, à un corps de métier par exemple. Dans ce cas, le terme de *vue* est classiquement utilisé. Pour gérer les deux cas qui précèdent, la fonction de transformation réciproque existe et est implémentable. [Katranuschkov06] propose un processus de transformation des données adapté et compatible avec une démarche de conception collaborative, illustré par la figure 1.20. Le projet SABLE [BLIS-Project05], abordé à la section 1.2.3, est aussi basé sur des transformations de modèles, via la création de vues métiers, pour simplifier l'accès aux données.

Dans le cadre de la comparaison de données, la transformation de modèles donne la possibilité de simplifier la sémantique avec ou sans perte d'informations. Dans un premier temps, elle permet d'extraire l'information qui peut – et nécessite – d'être comparée. Cette extraction dépend de l'étape à laquelle se trouve le processus de conception, des acteurs concernés, et d'autres raisons qui font que la comparaison globale donnerait un résultat rempli d'informations inutiles. Il s'agit donc d'une simplification sémantique *avec* perte d'information, mais l'information perdue est supposée superflue selon le contexte défini. Ce mécanisme de comparaison sélective, proposé par [Broad03], évite l'extraction d'information au sein du résultat de la comparaison globale elle-même. Elle évite ainsi d'effectuer un travail superflu sur les modèles d'ouvrages. Dans un deuxième temps, le mapping peut définir des classes d'équivalence sémantique, c'est-

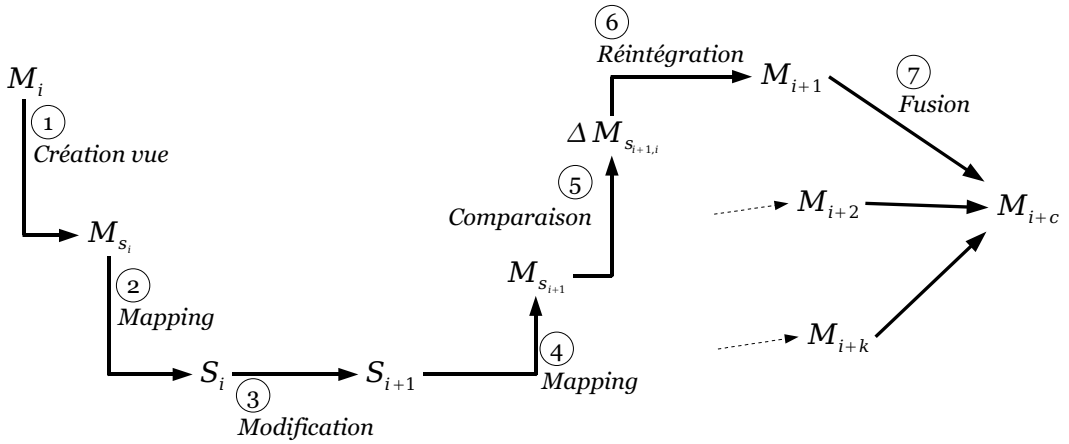


**FIGURE 1.20 :** Transformations d'un modèle d'ouvrage lors d'un processus de conception collaborative, d'après [Katranuschkov06]. Les étapes 1, 2, et 4 correspondent à des transformations de modèles. En 1, la création d'une vue donne un sous-modèle respectant la structure du même modèle de données. En 2 et 4, les mappings assurent l'interopérabilité entre le modèle de données centralisé pour le projet et un modèle spécifique à une application métier. L'étape 6 ne peut être directement considérée comme une transformation de modèles car elle obtient le modèle $M_{i+1}$ à partir d'un résultat de comparaison $\Delta M_{s_{i+1,i}}$ et non pas à partir de $M_{s_{i+1}}$ directement.

à-dire qu'elle transforme deux informations structurellement différentes mais sémantiquement équivalentes dans le modèle de données $A$ en une même information dans le modèle $B$. Un exemple typique est le type de représentation d'un point 3D : cartésienne, cylindrique, ou sphérique. Pour des raisons de flexibilité, le modèle de données $A$ permet de stocker des points 3D en représentation cartésienne, cylindrique ou sphérique. Si l'on utilise un algorithme de comparaison « classique », deux points 3D de représentations différentes seront toujours différents. Grâce à une transformation vers un modèle $B$ où un seul mode existe, par exemple le mode cartésien, la comparaison permettra de détecter correctement tous les points équivalents. La figure 1.21 illustre le propos précédent.

La transformation de modèles est donc une technique de prétraitement des structures à comparer pour augmenter la pertinence des résultats du traitement principal.

Les méthodes existantes répondent à de nombreux besoins de la part des industriels. Le succès des systèmes de versionnement orientés objets les rendent indispensables pour de grands projets industriels en aéronautique et en automobile. Cependant, le domaine de la construction semble en retard, en termes de conception collaborative. Pour expliquer ce phénomène, nous avons identifié plusieurs limites de ces méthodes.

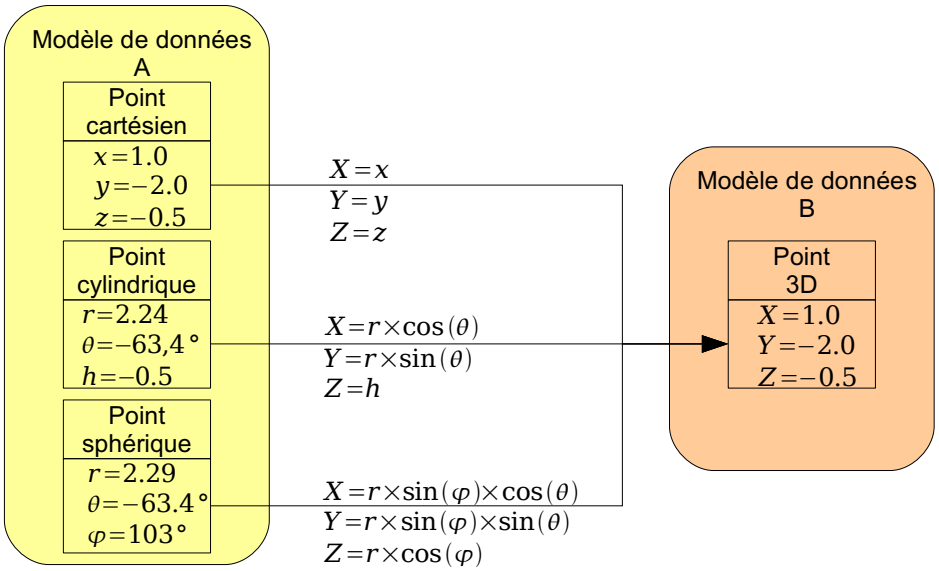


**FIGURE 1.21 :** Simplification sémantique lors d'une transformation de modèles. Dans le modèle de données $A$, il est possible de définir des points selon des repères cartésiens, cylindriques, ou sphériques. Si un modèle d'ouvrage possède trois points selon trois représentations différentes, une comparaison automatique directe conclurait toujours que les points sont différents. En utilisant une transformation vers un modèle $B$ où seule la représentation cartésienne existe, la comparaison automatique sur le modèle d'ouvrage transformé indiquerait que les points sont équivalents, ce qui est sémantiquement correct.

## 1.4 Limites des méthodes existantes

Après avoir constaté le retard pris dans le domaine de la construction concernant l'utilisation de modèles centralisés pour la conception collaborative, le but est ici de comprendre ce qui empêche une utilisation directe des MNC et des outils collaboratifs existants dans cette activité. A cette fin, une première partie vise à expliquer pourquoi les IFC n'ont pas été complètement adoptés par les projeteurs, malgré ses avantages potentiels et ses dix ans d'existence. Ensuite, l'objectif est d'expliquer pourquoi la comparaison générique d'arbres ou de graphes n'est pas adapté à la synchronisation de la MNC. Enfin, il s'agit de montrer quelles sont les limites d'une adaptation directe des méthodes de versionnement existants.

### 1.4.1 Exploitation difficile des IFC

Durant ces dix dernières années, le fossé s'est creusé entre les innovations proposées autour des MNC (dont les IFC) et les méthodes de travail de la plupart des projeteurs. D'un côté, les différentes enquêtes menées au CSTB montrent qu'une majorité d'acteurs du BTP ne connaissent pas les IFC, même de nom, malgré leur existence depuis 10 ans. De l'autre côté, les IFC deviennent une norme ISO/PAS 16739 et servent de support à de nombreuses recherches et de nombreux développements, comme cela a été détaillé à la section 1.2.3. Par exemple, Le logiciel Active3D [Cruz02] a reçu la médaille d'or de l'innovation technologique au salon BATIMAT

2003-2004. De plus, ce décalage est confirmé par [Aranda-Mena06] qui mesure le degré d'adoption des IFC grâce à un croisement de plusieurs modèles de diffusion de l'innovation dont celui de [Rogers03].

De manière théorique, ce problème de diffusion de l'innovation a été formalisé par [Moore01] qui décrit le risque pour une innovation d'être complètement abandonnée entre la 2ème phase d'adoption (Early Adoptors) et la 3ème (Early Majority), transition où se trouve actuellement les IFC d'après [Aranda-Mena06]. Le phénomène se nomme *Moore's Chasm*, traduisible littéralement par « fossé de Moore », et est induit par le décalage des attentes entre innovateurs et implémenteurs. La figure 1.22 illustre ce propos.

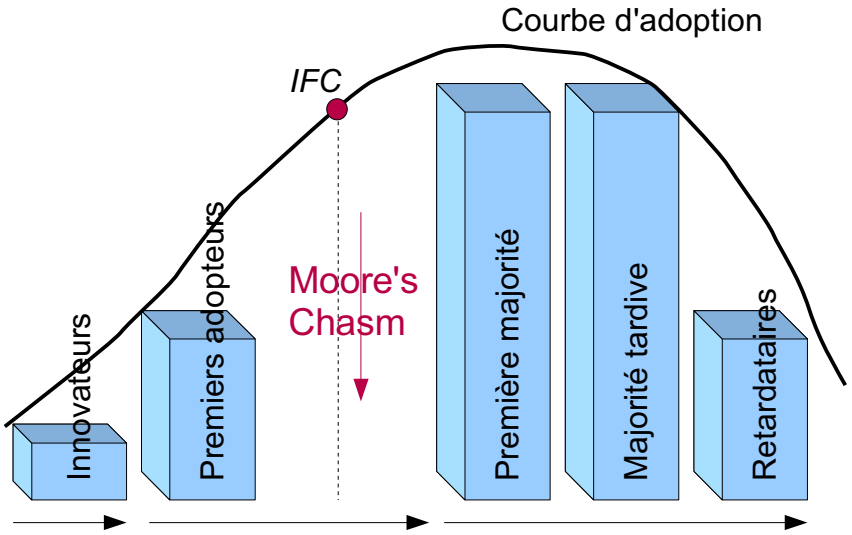


**FIGURE 1.22 :** Position des IFC par rapport à la courbe d'adoption, d'après [Moore01] et [Aranda-Mena06].

D'autres problèmes « sur le terrain » expliquent ce fossé entre innovation et exploitation. Les enquêtes du CSTB mettent ainsi en relief une première difficulté : changer la conduite de projet, en passant des plans à la maquette, demande un investissement que les acteurs concernés estiment trop grand. Cet investissement se répercuterait alors sur l'offre chiffrée de la maîtrise d'oeuvre lors d'un concours ou d'une procédure d'appel d'offre. Or ces derniers n'exigent pas l'utilisation d'une maquette virtuelle, et la plupart du temps n'en font même pas une recommandation [Ku06]. L'adoption des IFC ou de n'importe quel modèle d'information centralisé se heurte donc à des barrières d'ordre financier, organisationnel, et réglementaire [Aranda-Mena06]. Cela est d'autant plus regrettable, car une organisation qui n'a pas migré vers la MNC aura un lourd investissement au départ, certes, mais par la suite, l'investissement supplémentaire en modélisation sera compensé par une collaboration améliorée et des réunions de synthèse grandement facilitées. Un deuxième problème, remettant en cause la maturité des IFC, tient au fait que les implémenteurs n'ont pas toujours pris soin de respecter le standard IFC et toutes ses contraintes. En effet, ils interprètent arbitrairement certains types d'informa-

tions. [Pazlar06] montre, au niveau de la géométrie, qu'une altération partielle des données a lieu lors du passage du modèle de bâtiment d'un logiciel à l'autre. De manière plus générale, [Amor06] pointe et classifie des problèmes structurels et sémantiques lors de l'import/export d'un modèle IFC par un logiciel de CAO : par exemple, la perte de précision d'un nombre en virgule flottante. Enfin, un dernier problème affecte au quotidien les avantages apportés par les IFC : dans le cadre de l'étude d'un cas réel, [Drogemuller06] teste et relève les difficultés d'ordre informatique (taille des fichiers, performances) et sémantique (par exemple : le manque d'informations concernant l'impact environnemental et la qualité de l'air intérieur) qui peuvent entraver l'efficacité de l'activité collaborative avec l'utilisation de logiciels basés sur les IFC.

Les IFC représentent l'alternative la plus active concernant la MNC, notamment grâce à une forte activité R&D autour de ce standard et un appui de la part des implémenteurs logiciels. Cependant nous avons montré que l'adoption des IFC n'est pas encore assurée. Des mécanismes synchronisation *ad-hoc* sont donc nécessaires pour rendre la MNC accessibles à tous les projeteurs, et contourner les défauts propres à ce standard. Pour cela plusieurs approches ont été exposées dans les sections précédentes. Si l'utilité de ces méthodes a été démontrée, elles ont néanmoins plusieurs limites dans le cadre de la synchronisation de la MNC.

### 1.4.2 Comparaison de structures contraintes

Parmi les méthodes de comparaison existantes, celles qui ont été vues à la section 1.3.1, notamment la comparaison d'arbres, sont limitées à des structures de données *libres*, c'est-à-dire soumises à aucune hiérarchisation, imposée par un modèle de données complexe. Or, pour la comparaison de données issues d'une MNC, ces dernières suivent la structure d'un modèle qui peut être très complexe comme les IFC – plus de 600 entités élémentaires. Il y aurait une perte à la fois de performance et de sémantique. En effet, une comparaison d'arbres de deux modèles d'ouvrage IFC reviendrait à effectuer les deux étapes suivantes :

1. **Transformation du modèle IFC en un modèle d'arbres générique** : on considère qu'un arbre ou n'importe quelle structure dite libre suit en fait une hiérarchisation d'un modèle d'arbre n-aire : c'est un modèle qui possède une seule entité élémentaire, en plus des types simples comme les nombres réels ou les chaînes de caractère, l'entité *noeud* (cf. figure 1.23). Cette dernière possède des attributs concernant ses propriétés propres (en chaînes de caractères) et est liée à d'autres entités *noeuds*. On effectue alors une transformation de modèles (vue à la section 1.3.3) depuis le modèle IFC vers le modèle d'arbre n-aire.
2. **Comparaison des structures** : la comparaison entre arbres génériques est effectuée.

Cette technique n'est donc clairement pas adaptée car une perte d'information intervient durant la transformation, c'est-à-dire les entités ne sont plus clairement distinctes puisqu'elles deviennent toutes instances de l'entité *noeud*. Potentiellemement le système se retrouve à comparer tous les noeuds fils d'un même noeud deux à deux, ce qui n'arriverait pas dans le modèle de départ car les objets ne seraient pas systématiquement instances de la même entité. La perte de sémantique induit ainsi une perte de performance.

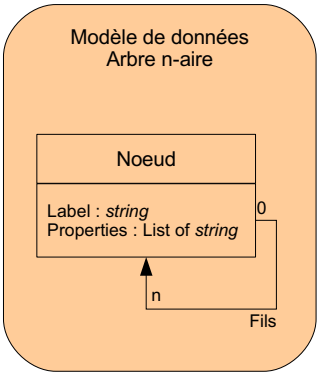


**FIGURE 1.23 :** Modélisation générique d'un arbre n-aire. Un noeud labélisé possède un ensemble de propriétés sous forme de chaînes de caractères et est lié à une liste d'autres noeuds.

Cette constatation conduit donc à abandonner l'utilisation d'algorithmes de comparaison d'arbres ou d'autres structures libres, sans prendre en compte la définition du modèle de données. Les mécanismes de versionnement, vus à la section 1.3.2, prennent en compte la structure du modèle de données en l'intégrant pleinement dans leur système au même titre que les instances, les modèles d'ouvrages. Si de tels systèmes ont un succès croissant dans le monde de l'aéronautique et l'automobile, l'adaptation directe de ces systèmes au monde de la construction pose plusieurs problèmes.

### 1.4.3 Adaptation des systèmes de versionnement orientés objets

Si les systèmes de versionnement existants répondent aux besoins de certains industriels, nous avons identifié plusieurs limites dans leur adaptation à la MNC. Trois types de limites ont été caractérisés.

#### Dépendance sur la persistance des données

Les systèmes de versionnement orientés objets proposent un suivi avancé du modèle de l'ouvrage avec des mécanismes de synchronisation. Ils autorisent ainsi la modification du modèle par plusieurs acteurs simultanément. Ces systèmes stockent aussi les données elles-mêmes dans une base centralisée. Si cette centralisation permet d'offrir une solution complète aux projeteurs, elle peut cependant diminuer l'interopérabilité entre acteurs. En effet, une solution globale de stockage des données, avec des mécanismes de synchronisation, impose à tous les acteurs l'utilisation de cette solution, si l'on souhaite un suivi et une synchronisation optimale des données. Dans le domaine de l'aéronautique ou l'automobile, cela fonctionne car l'ensemble des acteurs reste suffisamment homogène. Cependant, dans le domaine de la construction, l'hétérogénéité des acteurs entraîne l'utilisation de supports informatiques complètement différents. Même en s'assurant que les IFC sont utilisés par toutes ces applications, le stockage local du modèle pour chaque acteur est potentiellement différent (fichiers, serveurs internes, etc.). C'est pourquoi, les solutions existantes invitent fortement les acteurs à soumettre, à chaque modification, le mo-

dèle auprès du serveur de données centralisé. Il s'agit d'une contrainte supplémentaire pour les acteurs qui n'ont pas encore migré vers la MNC centralisée. Nous prenons donc pour hypothèse que l'indépendance du mécanisme de synchronisation vis-à-vis de la persistance des données est nécessaire, afin d'assurer une transition plus douce vers la MNC. Chaque acteur pourrait ainsi adopter en local le même mécanisme de suivi, via un logiciel client, mais garder sa propre solution locale de stockage des données.

#### Nécessité d'un constructeur d'application

La mise en place d'une solution de versionnement orientée objets est une opération complexe. C'est pourquoi il existe des produits commerciaux de conception collaborative pour la construction offrant une solution déjà configurée. *Active3D* a déjà été cité dans la section 1.2.3, Autodesk propose aussi des solutions autour de Revit© et Buzzsaw© qui supportent les IFC. Si l'utilisateur souhaite une solution plus personnalisée, notamment pour des projets de ponts et supporter ainsi le modèle IFC-Bridge, il peut s'orienter vers un système de versionnement générique. Pour cela, un constructeur d'applications paramètre le système pour l'adapter aux besoins de l'utilisateur [Urtado98]. Cette configuration reste très complexe, longue et donc coûteuse. Notre étude bibliographique n'a pas permis d'identifier des solutions intermédiaires entre outil commercial « clef en main » et des systèmes dont la configuration est entièrement manuelle.

#### Absence d'analyse sémantique

Afin de maximiser la flexibilité de la MNC pour les acteurs de la construction, le modèle IFC propose plusieurs manières de représenter une information. Par exemple, la définition géométrique d'un objet de construction peut se faire grâce à l'extrusion de formes variées, à la définition d'une boite englobante, ou grâce à un ensemble de triangles (B-Rep). Un mur peut donc être construit de ces trois façons différentes. Il existe ainsi une multi-représentation de l'information au sein du modèle IFC. Or, dans un milieu hétérogène d'acteurs et de logiciels, une même information peut être créée et transformée différemment, tout en respectant le standard IFC. Une approche purement structurelle de la comparaison et de la synchronisation des données ne permet pas de prendre en compte la multi-représentation de l'information. Pour contourner le problème, les solutions existantes utilisent des transformations de modèle pour effectuer des vues et des simplifications sémantiques, comme cela été vu à la section 1.3.3 et illustré sur la figure 1.21. Nous estimons qu'il s'agit d'une solution qui alourdit le processus global de conception, illustré sur la figure 1.20. Notre étude bibliographique n'a pas pu mettre en évidence des méthodes de comparaison, prenant en compte la sémantique, sans utiliser de transformations de modèles.

Les limites de toutes les approches existantes mettent en relief certaines problématiques et aident ainsi à définir le cadre d'étude des travaux de cette thèse.

## 1.5 Problématiques soulevées et cadre d'étude choisi

Les problématiques soulevées par l'étude bibliographique concernent moins les modèles de données supportés par les MNC que leur gestion au sein d'un projet. En effet, dans un monde parfait, sans conflit entre acteurs de projet différents, les MNC apporteraient une solution performante. Ce n'est malheureusement pas le cas, le processus de conception est donc complexe. Alors l'utilisation directe de la MNC via des serveurs centralisés n'apporte pas une flexibilité suffisante pour répondre aux besoins des projeteurs en conception. C'est pourquoi, une analyse globale du processus de conception et des liens avec la MNC (section 1.5.1) permet de définir un processus-type pour l'utilisation de la MNC en conception (section 1.5.2) dans le cadre de cette thèse. L'objectif est alors de définir les objectifs et la démarche de cette dernière (section 1.6).

### 1.5.1 Gestion de la MNC au sein d'un projet

Le cycle de vie d'un projet d'ouvrage de construction se compose de plusieurs phases. Ces dernières sont clairement identifiées comme l'illustre la figure 1.24 pour le bâtiment. Pour les autres projets d'aménagement et de génie civil, les étapes principales restent les mêmes :

- La programmation
- La conception
- La construction
- L'exploitation

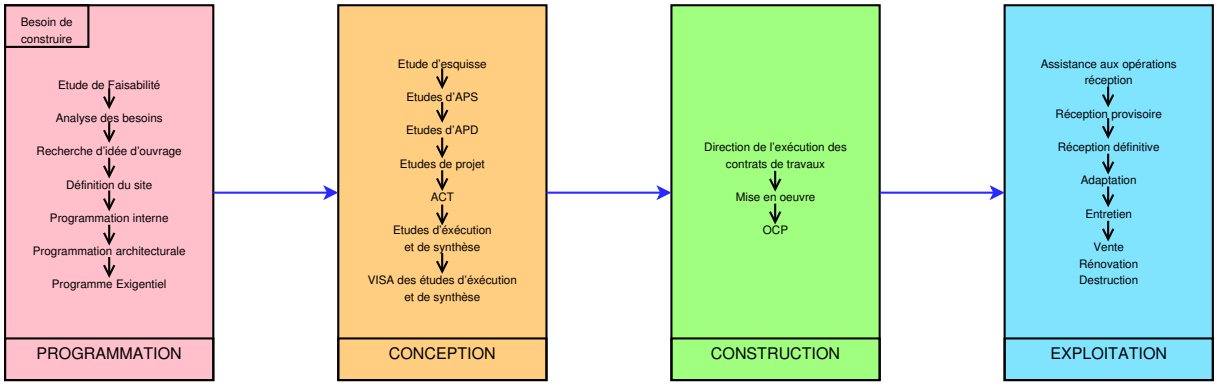


**FIGURE 1.24 :** Le cycle de vie d'un bâtiment (schéma initié par [Guerriero02] modifié par Ah.L).

Nous avons vu que l'approche de la MNC centralise les informations de chaque *métier* impliqué dans le projet autour d'une même base de données. Cette centralisation s'effectue aussi au niveau de deux autres dimensions :

- Les *lots* de projet : en particulier dans le domaine génie civil, les grands projets sont divisés en lots. Chaque lot peut être traité par des acteurs différents, mais la cohérence de l'ensemble a besoin d'être assurée au niveau des interconnexions. La MNC centralise l'information de tous les lots pour répondre à ce besoin.
- le *cycle de vie* du projet : dès l'esquisse, la MNC est créée, puis définie entièrement en

conception, enrichie durant la construction (notes de chantier, études d'exécution), mise à jour durant la maintenance. Elle centralise donc l'information au cours du temps.

La création de la MNC est donc liée au cycle de vie du projet, en particulier lors de la phase de conception. En effet, cette dernière englobe l'activité de réflexion pour répondre à un problème posé en optimisant les coûts, la qualité, et le temps de réalisation. Le résultat de cette réflexion est la création d'un ensemble de données et de documents nommés *artefact* de projet [Simon96]. Dans cette analyse la MNC et les documents dans un format spécifié forment le support logique et physique de l'artefact. Adapter la MNC aux pratiques professionnelles consiste donc à analyser le processus de conception en premier lieu. Ensuite, l'aspect documentaire mérite aussi une analyse propre. En effet, la documentation sert à justifier des choix techniques et à expliciter des informations par écrit. Elle assure notamment la viabilité juridique du projet. Lors de la conception, une analyse des liens entre le cycle de vie du projet, la documentation et la MNC aiderait aussi à l'adaptation de cette dernière aux pratiques professionnelles.

[Hanser03] analyse globalement la communication et la collaboration entre acteurs lors du processus de conception. Il relève notamment l'enchaînement de deux étapes principales :

- La co-conception : lors de cette étape, les acteurs de projet sont ensemble lors d'une réunion de travail. Ils communiquent de manière *synchrone* pour enrichir le projet et résoudre le conflit. A l'issue de cette étape, de nouveaux objectifs sont définis ainsi que la division du travail.
- La conception distribuée : durant cette étape, les acteurs ou groupes d'acteurs travaillent en parallèle sur le projet selon leur métier et le lot de projet. Ils communiquent de façon *asynchrone* par n'importe quel mécanisme de notification : courriers, e-mails, serveur informatique du projet. A la fin de cette étape, une nouvelle réunion de travail synthétise les modifications effectuées, résout les conflits et fixe de nouveaux objectifs.

La figure 1.25 schématise cet enchaînement. Vis-à-vis des modèles de données et des systèmes de gestion existants, nous avons constaté que la problématique se situe au niveau de la synchronisation des données après résolution de conflits. Afin de répondre à cette problématique, nous proposons un processus-type de gestion informatique de projet en conception. Ce processus-type sert de base pour concevoir et implémenter des modèles de synchronisation de la MNC adaptés aux pratiques professionnelles.

### 1.5.2 Proposition d'un processus-type de gestion de projet en conception

Lors de la publication des premiers modèles de données pour supporter la MNC, la priorité a été donnée à la cohérence propre du modèle, c'est-à-dire l'intégration de plusieurs points de vues différents, en fonction de chaque corps de métier, au sein d'un même système d'information. Par extension, les premières solutions de gestion ont aussi proposé un accès uniquement centralisé à la MNC, conformément au principe du modèle de projet partagé, vu à la section 1.2.3 et illustré par la figure 1.7. Ce type d'accès permet de garder la cohérence du projet à tout instant, car chaque modification est effectuée sur le même modèle. Cependant, un tel système ne permet pas à plusieurs acteurs de modifier la même pièce au même instant. Il s'agit d'un accès centralisé *bloquant*. Cela induit un processus de conception contraignant. En effet, deux acteurs ayant des sensibilités et des objectifs différents peuvent souhaiter modifier de façon contradic-

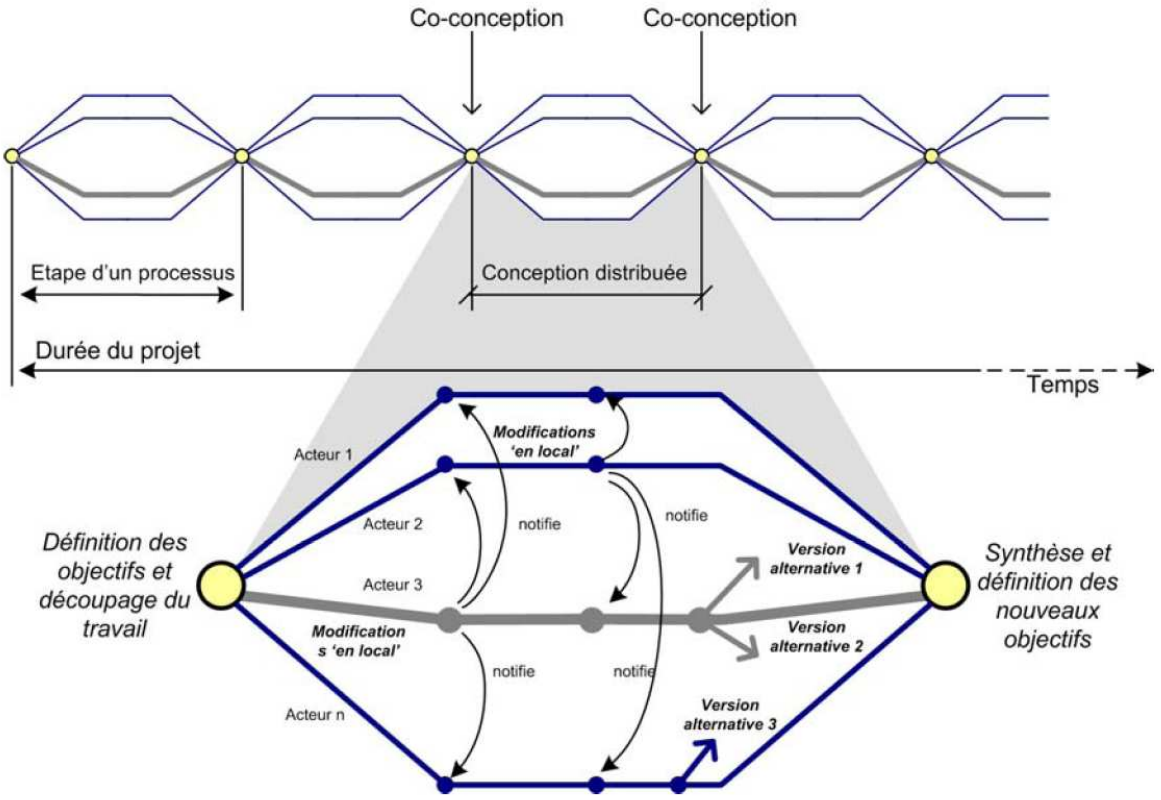


**FIGURE 1.25 :** Processus de conception concourant, conception distribuée et points de synthèse, d'après [Turk97] [Hanser03].

toire la MNC. Par exemple, un architecte souhaite supprimer une colonne pour améliorer la visibilité d'un espace donné, alors que l'ingénieur du bureau d'études applique des propriétés de structures à cette colonne qu'il estime nécessaire à la stabilité de l'ouvrage. Ce conflit ne peut être résolu par un système bloquant car il donnera de manière arbitraire (selon la chronologie des modifications, ou les autorisations par acteur) la priorité à l'architecte ou à l'ingénieur. Selon l'analyse de [Hanser03], la résolution de ce conflit ne peut pas toujours se faire en conception distribuée, mais peut demander une réunion de co-conception. Finalement un accès centralisé bloquant occulte les conflits entre acteurs pour un problème posé. Il n'apporte ainsi aucune aide pour préparer des réunions de travail en co-conception.

C'est pourquoi nous proposons un système de gestion centralisée *non bloquant* : plusieurs acteurs peuvent effectuer des modifications sur le même objet. Pour cela, nous avons repris le schéma de la figure 1.25 pour l'appliquer aux données elles-mêmes, c'est-à-dire la MNC. Nous obtenons alors un système d'accès à la MNC illustré par la figure 1.26.

A partir d'une MNC commune issue d'une réunion de travail entre acteurs, à l'étape $n$ du processus de conception, chaque acteur ou groupe d'acteur peut enrichir la MNC commune, localement. Deux groupes d'acteurs différents peuvent modifier le même objet différemment. De plus, la documentation associée possède des liens avec la MNC. Dans le travail de cette thèse, les

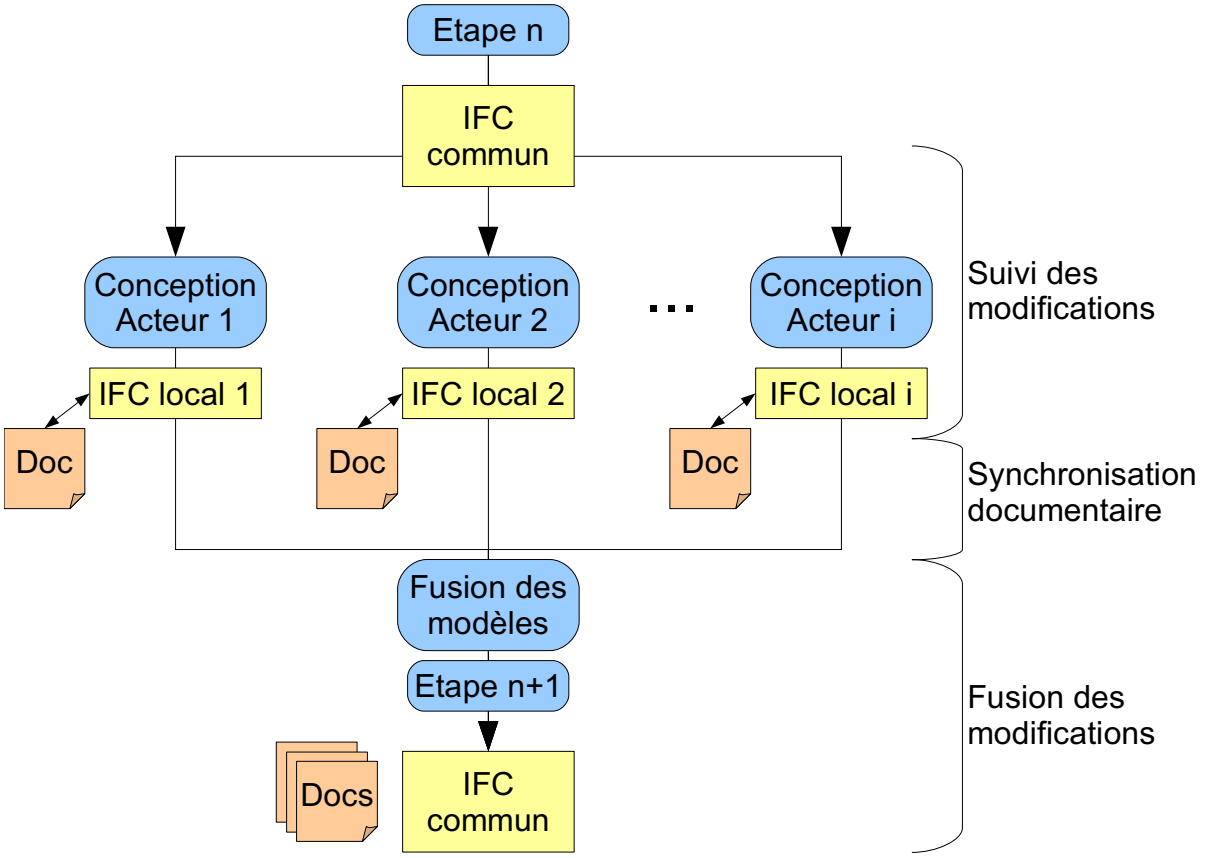


**FIGURE 1.26 :** Proposition d'un système d'accès à la MNC, basé sur le modèle IFC, lors de la conception d'un projet de construction.

liens entre une MNC et une documentation technique seront explicités. Ensuite, pour préparer la réunion de synthèse, étape $n+1$, et lors de celle-ci, les données sont fusionnées. Cette fusion est supervisée par un coordinateur de projet, en particulier pour les modifications contradictoires. Le système fournit ainsi une aide à la décision au coordinateur pour détecter les incohérences lors des modifications en conception distribuée. Une nouvelle MNC commune, enrichie d'un ensemble de documents, est obtenue à cette étape.

Ce processus-type ainsi spécifié sert de base pour définir les objectifs des travaux de cette thèse.

## 1.6 Conclusion : objectifs et démarche des travaux

Les travaux de thèse s'inscrivent dans le cadre d'étude abordé dans la section précédente. Ils visent à adapter la gestion de la MNC, en conception, aux pratiques professionnelles. Pour cela, il s'agit concevoir et développer des algorithmes pour automatiser et assister le processus-type spécifié auparavant. Comme le montre la figure 1.26, nous avons identifié trois niveaux

d'études :

- **Le suivi des modifications en conception distribuée** : au niveau de chaque groupe d'acteur, le travail consiste à détecter automatiquement les changements de la MNC entre deux versions successives. Le choix du logiciel est libre pourvu qu'il utilise le standard commun à tout le projet, c'est-à-dire IFC, ou IFC-Bridge.
- **La synchronisation documentaire** : durant la conception distribuée, au sein de chaque groupe d'acteur lié à un domaine métier spécifique, une documentation technique permet de rapporter l'évolution de la conception, d'expliquer et de justifier les choix qui ont été faits. Nous prenons pour hypothèse qu'il existe des liens entre la MNC et les documents techniques. Ils permettent alors de développer des algorithmes qui automatisent la génération automatique de documents ainsi que la synchronisation de ces derniers en fonction de l'évolution de la MNC.
- **Fusion des modifications** : à partir des différentes branches de conception distribuée, le travail consiste à obtenir une nouvelle MNC commune. Les limites de méthodes existantes et l'analyse précédente montrent que cette étape ne peut être complètement automatique. Les conflits au niveau de la conception se résolvent grâce à une réunion de synthèse et non grâce à un système informatique. Il s'agit donc d'une automatisation supervisée par un coordinateur de projet. En cas de conflit de conception, le système aide le coordinateur à détecter l'incohérence, sans pour autant la résoudre.

Par rapport à ces trois niveaux, nous prenons pour hypothèse que réutiliser les résultats du suivi des modifications comme donnée en entrée pour la synchronisation documentaire et la fusion des modèles permet de faciliter le développement d'un système informatique supportant le processus-type et d'en améliorer les performances. C'est pourquoi la démarche de cette thèse consiste à développer d'abord une méthodologie générale pour le suivi de modifications par comparaison de modèles d'ouvrages. Pour cela, nous avons d'abord choisi de concevoir et implémenter un moteur de comparaison structurel générique (chapitre 2), pour ensuite l'enrichir d'une assistance sémantique (chapitre 3). L'application de cette méthodologie sur le processus-type représente alors les résultats principaux de nos travaux (chapitre 4). Cette application porte non seulement sur le suivi des modifications, mais aussi sur ses apports pour préparer la synchronisation documentaire et la fusion des modifications.

## Résumé

L'objectif de ce chapitre est de décrire les approches existantes de la maquette numérique, dans le secteur du BTP, et de sa synchronisation, en phase de conception. Il vise de plus à montrer les limites des outils et méthodes existantes, notamment à cause de la spécificité du secteur (ouvrage unique, multiplicité des acteurs, imprévus récurrents). Nous définissons pour cela la notion de MNC (Maquette Numérique de Construction) comme étant l'adaptation de la maquette numérique au secteur BTP, depuis sa conception jusqu'à sa maintenance. Elle intègre non seulement le modèle de l'ouvrage qui centralise au mieux toutes les productions des acteurs impliqués, mais aussi la notion de prototype virtuel. Concernant les données elles-mêmes, l'utilisation de standards d'échanges permettent de créer et d'enrichir de manière collaborative le modèle d'ouvrage. Les secteurs de l'aéronautique et de l'automobile sont à l'origine des premiers standards d'échanges dans les années 80. Puis grâce à la norme STEP, une approche universelle a permis à tout secteur industriel de créer son standard d'échange : spécification du modèle de données en langage EXPRESS, échanges de modèles de produits par fichiers ASCII ou XML, ressources génériques standardisées (exemple : entités géométriques). Pour le secteur du bâtiment, les IFC (Industry Foundation Classes) constituent un modèle de support de la MNC, en exploitant les parties fondamentales de STEP. Il permet de définir tous les éléments de construction, les espaces, les processus, etc. La hiérarchisation de ces éléments s'effectue via des entités de relation typées (`IfcRel`) intermédiaires. Concernant les ouvrages hors bâtiment, après plusieurs essais au niveau national, un partenariat franco-japonais SAKURA a permis de proposer une extension aux IFC pour des projets de pont : IFC-BRIDGE. Au-delà de la hiérarchisation des données, la MNC nécessite des mécanismes d'accès et d'enrichissement, au sein d'un environnement collaboratif. Pour cela, nous avons choisi d'étudier la synchronisation des modèles d'ouvrage. Tout d'abord, la comparaison de fichiers ASCII se révèle inadaptée pour la comparaison d'informations structurées, même sauvées dans des fichiers texte. Ensuite, La mise en correspondance de graphes et la comparaison d'arbres apportent respectivement des réponses pour : identifier l'équivalence entre deux structures et déterminer une succession d'étapes élémentaires de transformation de l'une à l'autre. Le versionnement constitue un procédé pour comparer mais aussi stocker une ou plusieurs évolutions d'un modèle d'ouvrage, notamment lors d'une conception collaborative. Enfin, la transformation de modèles et la comparaison sélective permettent d'apporter des réponses adaptés aux acteurs, au type de projet, ou même à la phase à laquelle se trouve un projet. Toutes ces approches existantes ont des limites pour la définition et l'utilisation de la MNC. D'abord, l'exploitation des IFC par les projeteurs reste limitée : investissement pour assurer la transition vers la MNC, décalage entre implémenteurs du modèle et acteurs de BTP, non respect des standards par les applications existantes. Ensuite, les outils de versionnement existants n'apportent pas de solution à la fois adaptée selon le modèle de données et configurée sans l'intervention d'un constructeur d'applications. Enfin, la sémantique des modèles n'est pas prise en compte, ce qui rend le versionnement des modèles IFC ou IFC-BRIDGE délicat, dans la mesure où plusieurs entités peuvent représenter la même information. A partir de ces limitations, le travail consiste à proposer un ensemble d'algorithmes pour assurer la synchronisation de modèles d'ouvrage IFC-BRIDGE. Nous avons donc proposé un processus type de gestion de la MNC, en conception, permettant d'identifier les différentes étapes nécessitant une automatisation.

## Abstract

This chapter aims to describe current approaches concerning digital mock-up (DMU), applied on AEC, and its synchronization during design process. Besides the objective is to raise limitations of current tools and methods, especially due to AEC specificities (unique product, multiplicity of actors, haunting unexpected events). That is why we define the Digital Mock-Up for Construction (DMUC) concept. It relates to the adaptation of digital mock-up for AEC sector, from design to maintenance. It not only refers to product model, which tends to centralize information created by all involved actors, but also to Virtual Prototyping (VP). Concerning product data, the use of exchange standard enables the product model to be created and modified. Aviation and automotive industries were involved in the first exchange standards, during the 80's. Then a universal approach allowed every industry sector to create its own exchange standard, thanks to STEP norm : data model specified in EXPRESS language, product models exchange based on ASCII and XML files, standardized generic resources (for example, geometry definition). For the building industry, IFC (Industry Foundation Classes) refer to a support model of DMUC and uses main parts of STEP. Construction elements, spaces, processes, etc. can be defined. Structure of these elements is completed through typed relation entities (`IfcRel`) in between. Concerning other product than buildings, after several national models, a French-Japanese partnership SAKURA enables an IFC extension for bridge projects to be proposed : IFC-BRIDGE. More than data structuring, DMUC also needs read and write access in a collaborative environment. Therefore we would like to study product model synchronizing. First ASCII file comparison seems not adapted for comparison of structured information, even if they are saved in textual files. Second graph matching and trees comparison respectively solve these problems : structure equivalence and identification of a sequence of elementary steps to transform the first structure into the second one. Versioning not only relates to comparison process but also to storage of DMUC evolution, especially during collaborative design. Last model mapping and selective comparison give adapted results according to actors, project, or current phase in a project. Current approaches have limitations about DMUC definition and use. First IFC are still poorly used by designers : investments to support transition to DMUC, gap between model implementers and AEC actors, non-respect of standards from existing software. Second current versioning tools do not provide adapted solutions depending on any data model and simply configurable without any software engineer. Last model semantics is not considered here, which makes IFC and IFC-BRIDGE models versioning fussy, because several entities relate to equivalent information. From these limitations, this work aims to propose algorithms to support IFC-BRIDGE product models synchronization. Therefore we propose a framework for DMUC handling, during design. It identifies steps which need to be automated.

# Chapitre 2

# Approche générique de la comparaison structurelle de modèles d'ouvrages

## 2.1 Introduction

La MNC est décrite grâce à un modèle d'ouvrage. Ce dernier utilise une approche orientée objets de construction afin de centraliser l'information pour tous les acteurs de projets. Durant la conception distribuée, chaque équipe d'acteurs modifie ce modèle, en ajoutant de nouveaux objets, en supprimant d'autres éléments, ou en changeant certaines propriétés. Afin d'assurer le suivi de ces modifications, une solution consiste à comparer deux à deux les structures des données avant et après modifications. Or n'importe quel modèle d'ouvrages est ***l'instance*** d'un modèle de données commun (IFC, IFC-Bridge, etc.), qui lui-même est défini par un langage standardisé : EXPRESS.

L'idée principale de nos travaux consiste donc à proposer un mécanisme de comparaison structurelle, adapté à n'importe quel modèle d'ouvrage, décrit selon un modèle de données, pourvu que ce dernier soit spécifié dans le même langage : EXPRESS. Nous qualifions cette approche de *générique*. La difficulté sous-jacente est alors d'identifier les limites d'une approche purement générique et de trouver quelles informations, propres au modèle de données étudié, convient-il de spécifier pour concevoir une application de comparaison adaptée.

Les modèles de données étudiés dans le cadre de nos travaux sont d'abord les IFC et l'extension aux ponts IFC-Bridge. Or cette dernière élargit la sémantique du modèle IFC, adapté au départ au cadre bâti, à la définition des ponts (pièces prismatiques, géométrie adaptée, etc.). Cependant, elle ne change pas la structuration globale choisie par le modèle principal IFC. C'est pourquoi, par la suite, notre analyse confondra souvent ces deux modèles.

De plus, la division MOD-EVE du CSTB est amenée à participer à des projets autour de l'échange de données dans le domaine de la thermique spatiale. Ces échanges sont aussi basés sur un standard écrit en langage EXPRESS : TAS-ARM [ESA06]. Afin de valider la généricité de cette partie de nos travaux, ce modèle de données a aussi été étudié pour la comparaison

structurelle. A ce titre, il sera cité plusieurs fois dans ce chapitre.

Afin de concevoir ce mécanisme de comparaison structurelle, une première étape consiste à fixer le cadre d'étude théorique pour la structuration de l'information au sein d'un modèle de données (section 2.2). Plusieurs hypothèses générales seront notamment posées, applicables à tout modèle d'ouvrage. Ensuite, une analyse du langage EXPRESS permet de comprendre comment se structure l'information d'une MNC et de vérifier la compatibilité de ce langage avec certaines hypothèses précédemment posées (section 2.3). Enfin, avant de conclure, l'application du cadre d'étude aux modèles de données IFC et IFC-Bridge consiste à vérifier toutes les hypothèses théoriques et à concevoir, à partir de ces dernières, l'algorithme général de comparaison structurelle (section 2.4).

## 2.2 Cadre d'étude théorique

### 2.2.1 Description des structures de données à comparer

Pour définir la structure des données à comparer, nous avons choisi d'utiliser, à la base, les définitions proposées par [Urtado98] qui ont été vues au premier chapitre, à la section 1.3.2, à propos des entités élémentaires et complexes. Le travail consiste alors à préciser les définitions précédentes dans le cadre de cette thèse.

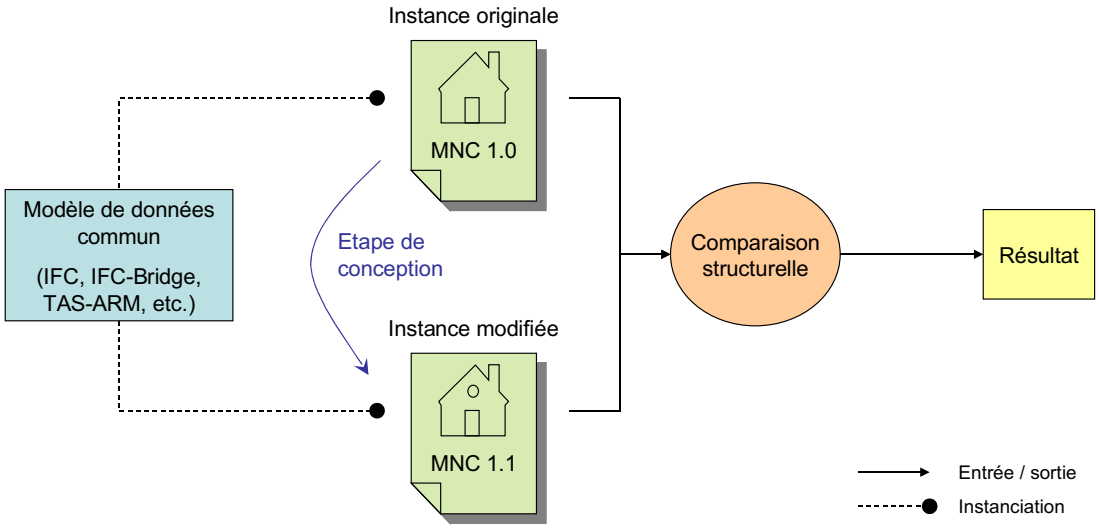


**FIGURE 2.1 :** Mécanisme global de la comparaison structurelle.

Tout d'abord, les données à comparer sont deux *instances* d'un même modèle de données, comme IFC, IFC-Bridge ou TAS-ARM par exemple. Une de ces instances est appelée *instance originale* et l'autre *instance modifiée*. En effet, la synchronisation de MNC suppose de suivre les modifications de cette dernière en cours de conception, et, pour chaque étape, cela se traduit par une comparaison entre une MNC de base et la même MNC après modifications par un

concepteur. La figure 2.1 illustre le procédé. Ensuite, de manière générale, une instance d'un modèle de données est associée à un graphe orienté d'entités complexes, ces dernières étant elles-mêmes des graphes orientés connexes d'entités élémentaires. Donc l'instance est associable à un graphe orienté d'entités élémentaires. Une première hypothèse concerne cette association instance-graphe :

**Hypothèse 1.** *Les liens de dépendance entre entités n'ont pas d'attribut. Ces dépendances sont transformées en arêtes orientées et labellisées durant l'association instance - graphe.*

En effet, dans le premier chapitre, à la section 1.3.2, a été abordée la notion d'entité complexe. Illustrée par la figure 1.18, cette notion suppose l'existence de liens de dépendances entre entités élémentaires. Ces liens sont labellisés et possèdent aussi des attributs. Cette hypothèse suppose l'ensemble des attributs vide. Il convient de préciser dès maintenant que le langage EXPRESS ne permet pas d'affecter des attributs de liens. Les modèles de données étudiés utilisent alors un mécanisme permettant de simuler ces fonctionnalités, grâce à des entités intermédiaires. Une autre hypothèse concerne la structure du graphe associé.

**Hypothèse 2.** *Pour toute instance du modèle, le graphe d'entités élémentaires associé possède une racine, c'est-à-dire qu'il existe une entité $r$ telle que pour toute entité $e_i$ du graphe, il existe un chemin orienté allant de $r$ à $e_i$.*

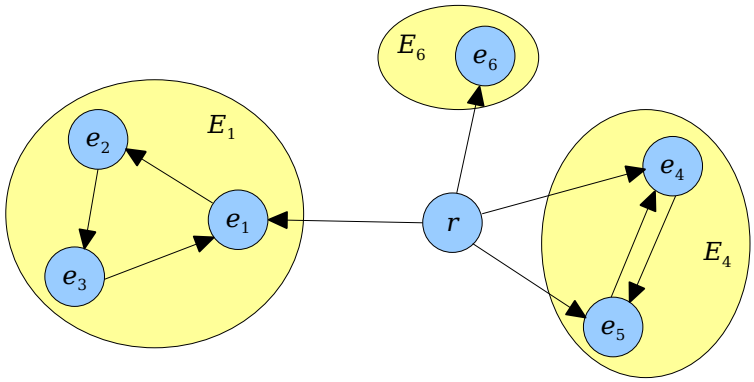


**FIGURE 2.2 :** Exemple de graphe associé à une instance, compatible avec l'hypothèse 2.

Cette hypothèse est nécessaire pour parcourir la structure entière à partir d'une entité. La figure 2.2 montre un exemple de graphe compatible avec l'hypothèse 2. Les entités élémentaires $e_i$ y sont représentées en bleu, et les entités complexes $E_k$ associées aux entités élémentaires $e_k$ en jaune. Concernant la structure du graphe, aucune autre hypothèse n'est faite. Des cycles et des liens multiples entre entités peuvent apparaître, comme l'illustre ainsi la figure 2.2. Après avoir décrit la structure des données au niveau macroscopique, il convient à présent de les décrire au niveau microscopique : les entités élémentaires. Dans le cadre de nos travaux, une telle entité possède trois propriétés fondamentales :

**Hypothèse 3.** *Quelque soit l'entité élémentaire de l'instance, chacun des ses attributs est comparable : il existe une relation d'équivalence dans l'ensemble des valeurs possibles pour cet attribut.*

Le but de la comparaison de deux attributs n'est pas d'ordonner mais seulement de conclure s'ils sont équivalents ou différents. L'utilisation de la relation d'équivalence est donc intuitive. En effet, un attribut est équivalent à lui-même (réflexivité) et cette relation ne dépend pas de l'ordre dans lequel apparaissent les deux attributs en entrée de comparaison (symétrie). De plus, la transitivité de la relation d'équivalence est nécessaire pour assurer une stabilité de la synchronisation, au fur et à mesure de la conception. Rappelons que la transitivité se traduit mathématiquement par : $a \equiv b$ et $b \equiv c \Rightarrow a \equiv c$. Si cette propriété n'est pas vérifiée, la cohérence des résultats de comparaison peut être mise en cause : ils peuvent varier en fonction des différents cheminements de conception, pour un même ouvrage. Par exemple, l'égalité entre deux nombres réels avec tolérance n'est pas transitive : si la tolérance est de $0.01$, $1.687 \approx 1.692$ et $1.692 \approx 1.699$ mais $1.687 \neq 1.699$. Cela signifie que l'algorithme de comparaison n'aurait pas détecté de changement ni entre la première et la deuxième version d'un modèle d'ouvrage, ni entre la deuxième et la troisième version de ce modèle d'ouvrage. Il aurait cependant détecté un changement si la première et la troisième version avaient été comparées directement. Il y a donc une incohérence. En conséquence, l'hypothèse 3 interdit l'utilisation d'égalité avec tolérance. Nous verrons au prochain chapitre qu'il est néanmoins possible d'affaiblir cette hypothèse, sous conditions, pour intégrer l'égalité avec tolérance, relation utile pour un projet de construction.

**Hypothèse 4.** *Chaque entité élémentaire est l'instance d'au moins une entité de modèle de données. Deux entités élémentaires sont comparées si et seulement si elles instancient au moins une entité commune du modèle de données.*

L'hypothèse 4 rappelle que la comparaison s'effectue entre objets de même type, ayant des attributs communs à comparer. Cependant, via le mécanisme ***d'héritage***, une entité peut être l'instance de plusieurs entités du modèle. La figure 2.3 montre divers exemples d'instanciation d'entités utilisant un mécanisme d'héritage : les éléments en jaune représentent les entités du modèle de données. Ce dernier permet de définir des véhicules, en particulier des autos et des motos qui héritent des attributs de l'entité véhicule. De plus, une entité `Remorque` existe ainsi qu'une entité `AutoRemorque` héritant de `Auto` et `Remorque`. Les éléments en bleu représentent des exemples d'instances. Ainsi, la voiture de Jean est une instance de `Auto`, et par héritage, est aussi une instance de `Véhicule`. La moto de Pierre est un exemple analogue. La voiture de Paul, quant à elle, est l'instance de `AutoRemorque`, et par héritage, est aussi l'instance de `Auto`, `Véhicule` et `Remorque`. Enfin, la moto de Jacques instancie à la fois de `Moto` et `Remorque` (il n'existe pas dans le modèle l'entité `MotoRemorque`), et par conséquent instancie `Véhicule`.

Dans ce formalisme, pour chaque paire d'entités élémentaires comparables, nous distinguons alors les entités *totalement* comparables des entités *partiellement* comparables. Dans le premier cas, les entités élémentaires instancient exactement les mêmes entités du modèle de données. Dans le deuxième cas, il existe au moins une entité du modèle de données instanciée par une entité élémentaire et pas par l'autre.

Une dernière hypothèse concerne l'identification des entités élémentaire :

**Hypothèse 5.** *Chaque entité élémentaire possède une identification, c'est-à-dire un label unique parmi toutes les entités de l'instance.*

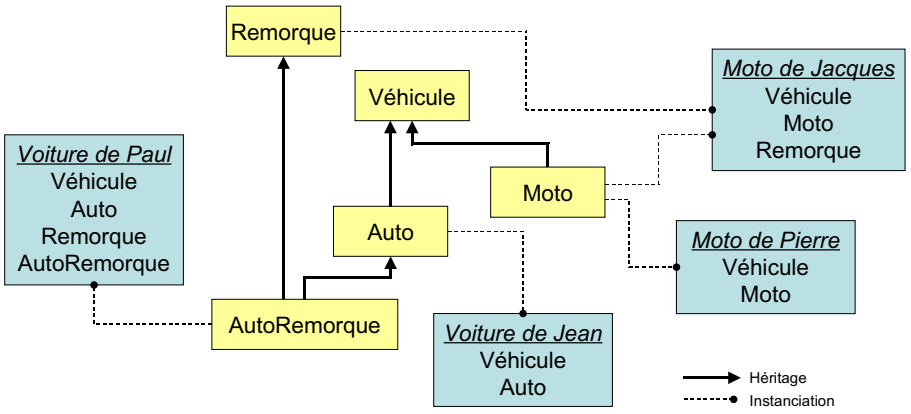


FIGURE 2.3 : Exemple de modèle de données utilisant l'héritage.

Comme l'indique la figure 2.2, chaque entité élémentaire possède un nom. Nous verrons aux sections suivantes que l'identification est nécessaire à la mise en oeuvre de la comparaison. De plus, il convient de préciser certaines définitions pour la suite du raisonnement.

**Définition 12.** *Une identification est dite* ***persistante*** *si elle est conservée durant les différentes étapes de conception.*

Toute identification n'est pas forcément persistante : lors du chargement du modèle, chaque entité peut se voir affecter un index temporaire pour faciliter l'accès aux données en mémoire. L'index est bien une identification mais une même entité peut posséder un index différent d'un chargement à l'autre.

**Définition 13.** *Une entité élémentaire est dite* ***identifiable*** *si elle possède une identification persistante.*

Précisons que l'hypothèse 5 n'impose pas aux entités élémentaires d'être identifiables. De plus, une même entité peut posséder plusieurs identifications : par exemple, une identification temporaire associée à la structure en mémoire et une identification persistante. Par la suite, nous verrons que le déroulement de la comparaison dépend de l'identifiabilité des entités élémentaires.

A partir de ces hypothèses, le travail consiste à présent à définir le déroulement général de la comparaison et en particulier les différents types de résultats possibles.

### 2.2.2 Spécification des résultats de comparaison d'entités identifiables

Dans cette partie, nous supposons que toutes les entités élémentaires d'une instance sont identifiables. Soient deux instances d'un modèle de données commun à comparer : l'instance originale et l'instance modifiée. L'hypothèse 1 a permis d'ignorer partiellement la sémantique des liens. Nous nous appuyons sur cette hypothèse pour négliger la comparaison des structures macroscopiques, en particulier des liens de dépendance. Nous nous focalisons alors sur la comparaison d'objets. En effet, la sémantique d'une MNC est supposée se situer uniquement au

niveau du contenu de ses entités élémentaires, d'après l'hypothèse 1. Cette considération sera néanmoins compensée par une prise en compte de la structuration locale d'une entité élémentaire identifiable.

La comparaison a pour objectif de rapporter les changements qui ont lieu entre l'instance originale et l'instance modifiée. Au vu des explications ci-dessus, cela consiste à rapporter les changements au niveau des entités élémentaires. La question se formule donc de la manière suivante : « quelles transformations peut subir une entité entre deux étapes de conception ? ». Afin d'y répondre, il convient de caractériser l'entité élémentaire identifiable à partir des hypothèses de la section précédente.

Une entité élémentaire identifiable est la donnée :

- d'une *identification persistante*, donnée statique, permettant de répondre à la question « Qui suis-je ? » et de caractériser ainsi sa durée de vie durant l'évolution de la MNC,
- d'un *état*, donnée dynamique, permettant de répondre à la question « Comment suis-je ? », la façon de décrire cet état dépendant directement des entités du modèle dont cet élément est l'instance.

Cette caractérisation permet de décrire de façon microscopique l'entité élémentaire. Cependant, elle ne comporte aucune information sur la localisation au sein de la structure de l'instance. C'est pourquoi nous ajoutons un troisième critère pour la caractérisation d'une entité élémentaire : la position relative de l'entité dans le graphe d'instance. Nous pouvons distinguer deux types de positionnement. La première possibilité consiste à déterminer l'ensemble des chemins allant de la racine du graphe – qui existe conformément à l'hypothèse 2 – jusqu'à l'entité élémentaire. Comme le graphe peut posséder des cycles, cet ensemble peut être infini.[1]. Il s'agit donc d'une approche *globale* puisqu'un changement de structure, indépendant de l'entité considérée, peut modifier son positionnement. La deuxième possibilité consiste à lister l'ensemble des arêtes pointant vers l'entité considérée ainsi que l'ensemble des entités à l'origine de ces liens directs. Cet ensemble est fini. Il s'agit d'une approche *locale* car le positionnement est relatif aux entités parentes. Nous avons choisi cette approche car nous ne souhaitons pas répercuter un changement de structure indépendant de l'entité considérée sur elle-même. Nous verrons plus loin qu'elle est en outre plus simple en termes d'implémentation. Ce nouveau critère de caractérisation permet de prendre en compte la structuration locale de l'instance en chaque entité élémentaire.

A partir de cette caractérisation, l'objectif est alors de définir l'ensemble des transformations possibles pour une entité élémentaire, selon les trois critères précédemment abordés. Tout d'abord, une entité ne peut pas changer d'identification persistante, sinon il ne s'agit plus de la même entité. En effet, trois événements peuvent affecter l'existence même de l'entité :

1. Une ancienne entité existait dans l'instance originale et n'existe plus dans l'instance modifiée. En fait, il n'existe aucune entité dans l'instance modifiée qui possède la même identification que l'ancienne entité. Elle a donc été *supprimée*.

[1] Mais comme il y a nombre fini de noeuds, cet ensemble est générable à partir d'un nombre fini de sous-chemins.

2. Une entité existe dans l'instance modifiée mais n'existait pas dans l'instance originale. Il n'existe donc aucune entité dans l'instance originale qui possède la même identification que la première entité. Elle a donc été *ajoutée*.
3. Une entité existe dans les deux instances. Une correspondance d'identifications a été faite entre deux entités venant de chaque instance.

Dans les deux premiers cas, le résultat de la comparaison correspond à l'affectation du statut 'supprimé' ou 'ajouté' à l'entité concernée. Dans le troisième cas, la comparaison continue entre les deux entités d'identifications identiques, selon les deux derniers critères. Les attributs des deux entités sont comparés, conformément à l'hypothèse 3. Deux types de résultats sont alors distingués : les deux ensembles d'attributs sont identiques ou pas. Concernant le dernier critère, les deux entités sont comparées par rapport à leur position relative. A cette fin, un ensemble de couples est construit pour chacune des deux entités. Chaque couple est composé du type du lien et de l'identification persistante de l'entité à l'origine de ce lien. La comparaison étudie alors l'égalité des deux ensembles de couples.

Les deux comparaisons précédentes étant indépendantes, quatre familles de résultats de comparaison existent. Elles sont répertoriées dans le tableau 2.1.

| CHANGEMENT ATTRIBUTS | CHANGEMENT POSITION | STATUT |
|---|---|---|
| NON | NON | *identique* |
| OUI | NON | *changé* |
| NON | OUI | *déplacé* |
| OUI | OUI | *changé&déplacé* |

**TABLEAU 2.1:** Les statuts de comparaison entre entités de même identification persistante.

Au total, au niveau de l'instance originale et de l'instance modifiée, chaque entité élémentaire identifiable se voit affecter un statut parmi six potentiels : 'ajouté', 'supprimé', 'identique', 'changé', 'déplacé', 'changé&déplacé'. La figure 2.4 illustre ces statuts de comparaison sur un exemple de modèle de pont. Selon l'implémentation choisie, il est possible d'adjoindre au statut des paramètres supplémentaires. Par exemple, si une entité élémentaire est détectée 'changé', un paramètre de ce statut pourrait correspondre à la liste des attributs responsables du changement. L'attribution de ces statuts et de leurs paramètres représente ainsi le résultat de la comparaison structurelle.

Dans cette section, l'hypothèse de l'identifiabilité de toutes les entités élémentaires a été posée. Or, une pré-analyse des modèles de données étudiés : IFC, IFC-Bridge et TAS-ARM, montre qu'il existe des entités non identifiables. Quel comportement convient-il alors d'appliquer à la comparaison structurelle, pour quels résultats ?

### 2.2.3 Etude de la comparaison d'entités non identifiables

Lors de la modélisation de données, l'identifiabilité d'une entité fait l'objet de choix de conception qui dépendent de la sémantique portée par cet élément. Par exemple, est-ce utile

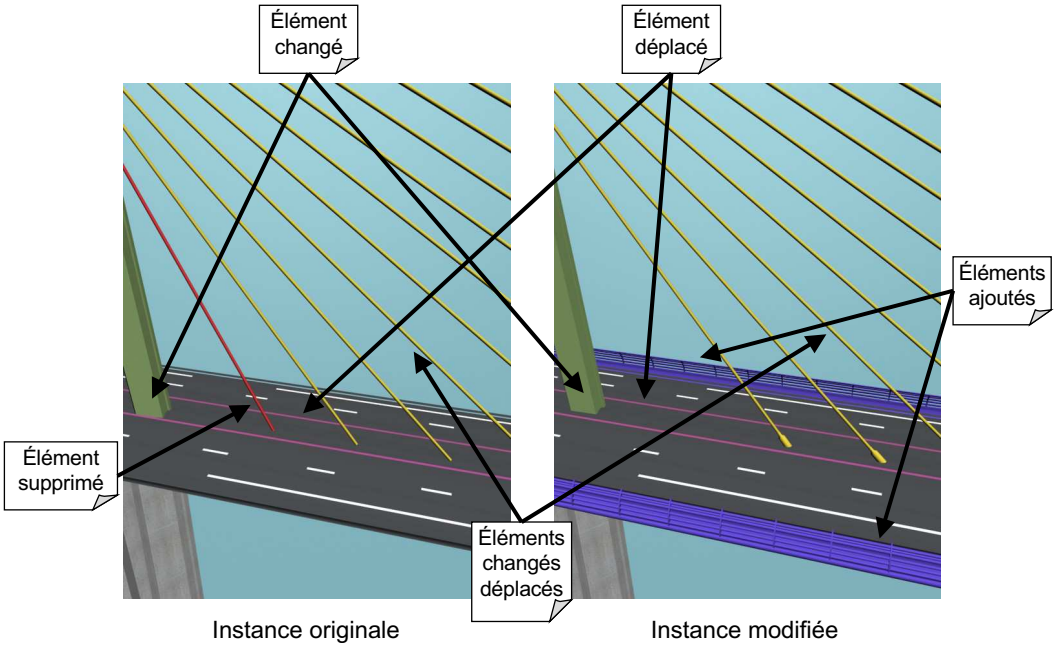


**FIGURE 2.4 :** Résultat de comparaison entre deux modèles de ponts à deux étapes de conception.

de donner une identification à un point 3D exprimable par trois coordonnées, ou suppose-t-on qu'il s'agit seulement d'une structure de données simples servant d'attribut à d'autres entités ? Ainsi pourrait-on aboutir à cette conclusion : une entité a besoin d'une identification persistante si et seulement si elle a une existence propre, indépendamment d'une autre entité. Par exemple, la notion d'espace architectural regroupe des informations qui évoluent tout au long de la conception. Pour étudier cette évolution, cet espace nécessite une identification persistante. Cependant, un point 3D reste une information purement géométrique s'il n'a pas de sémantique propre (comme pourrait avoir un POI, point d'intérêt). Il est donc dépendant d'une entité qui le référence et sa durée de vie est alors incluse dans celle(s) de sa (ses) entité(s) parente(s).

Pour une entité non identifiable, il n'est pas possible de lui affecter un statut parmi les six décrits dans la section précédente. En effet, supposons que l'on souhaite comparer deux points 3D. Si seule la coordonnée $z$ a changé, quelle conclusion peut-on tirer ? Le deuxième point est-il un point différent possédant son cycle de vie propre, ou bien le même point, mais dont l'attribut $z$ a changé ? Sans identification persistante, il n'est pas possible de faire la distinction entre l'affectation des statuts 'supprimé' - 'ajouté' d'une part et celle du statut 'changé' d'autre part. Une première solution consisterait à prendre l'ensemble des attributs comme identification persistante. Dans cet exemple, il s'agit de l'ensemble des coordonnées du point. Mais un tel choix a des conséquences : d'abord une telle entité ne se verrait jamais affecter le statut 'changé' ou 'changé&déplacé' car dès l'instant qu'un attribut change, l'identification change, il ne s'agit plus de la même entité. De plus, l'identification persistante est unique. Concernant le point 3D, cela implique l'impossibilité de définir deux points à la même localisation.

Nous avons donc choisi une autre voie pour gérer les entités non identifiables de manière

générique. Partant de l'hypothèse qu'une entité non identifiable regroupe des informations directement dépendantes des entités qui la référencent, que le cycle de vie d'une telle entité est inclus dans celui de ses entités parentes, nous en déduisons que cette entité élémentaire appartient à l'état de chaque entité parente. Ainsi, l'entité élémentaire référencée devient une valeur comparable d'un nouvel attribut labellisé de chaque entité parente. Nous appelons ce procédé *remontée d'attributs*, puisqu'il s'agit de copier les attributs de l'entité élémentaire non identifiable à chaque entité parente. Si un de ces attributs change, c'est l'entité parente qui est considérée comme changée. L'entité élémentaire non identifiable n'est donc plus vue comme une véritable entité car elle ne possède pas de statut propre de comparaison, mais partage le statut de l'entité parente. Il s'agit d'une démarche qui trouve des points communs avec le versionnement macroscopique [Urtado98].

Comme les entités élémentaires sont reliées entre elles à la manière d'un graphe, il convient d'appliquer la considération précédente de manière récursive, afin de transformer le graphe des données en un graphe ne possédant comme noeuds que des entités élémentaires identifiables. Pour cela, considérons un noeud non identifiable $e$ ayant $n$ entités parentes $p_1, \ldots, p_n$. D'abord il convient de remonter les attributs de $e$ dans chaque parent $p_i$. Ensuite, pour chaque entité non identifiable fille $f_i$ de $e$, les attributs sont remontés au niveau de chaque entité $p_i$, et ainsi de suite. Nous verrons plus loin qu'afin de gérer les cycles, la remontée d'attributs est en fait implémentée en créant un seul nouvel attribut dans l'entité parente. Cet attribut utilise une sérialisation ou une fonction de hachage des entités non identifiables, en considérant les liens de dépendances (et donc les cycles). De plus, lorsqu'il existe un lien partant d'une entité non identifiable à une entité identifiable, les attributs de l'entité identifiables ne sont pas remontés, mais seulement son identification. Le parcours récursif s'arrête alors pour cette entité. Nous reviendrons plus loin sur la définition et l'implémentation de ce mécanisme de sérialisation/hachage. A la fin, les entités non identifiables ne sont plus prises en compte.

La figure 2.5 illustre le procédé. En (a), un graphe associé à une instance est donné avec des entités identifiables et non identifiables. En (b), la remontée des attributs est effectuée pour les entités $e_1$ et $e_2$, les entités $f_1$ et $f_2$ sont alors supprimées du graphe. En (c), toutes les entités non identifiables sont supprimées. Via le lien 'g', ce ne sont pas les attributs de l'entité identifiable $e_2$ qui sont remontés, mais juste son identification persistante. Via le mécanisme de fonction de hachage, le lien 'i', responsable d'un cycle, est aussi pris en compte.

Cette section termine l'approche purement théorique de la comparaison structurelle. Il convient à présent de valider cette approche par son application à n'importe quel modèle de données suivant un même langage de spécification : le langage EXPRESS. Pour cela, la démarche consiste à analyser le langage et ses fonctionnalités, afin de vérifier les hypothèses exprimées dans cette section.

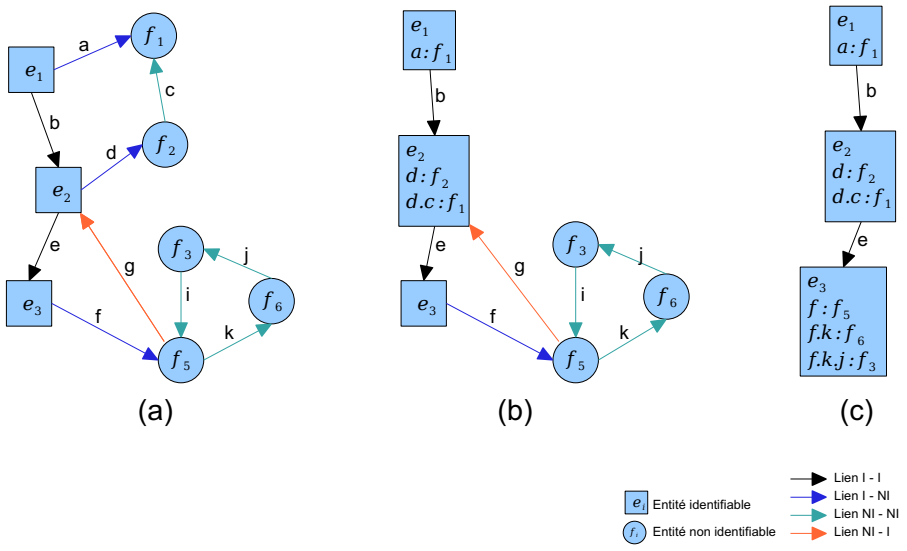


**FIGURE 2.5 :** Elimination des entités non identifiables du graphes d'instance. Dans la légende, 'I' se réfère à une entité identifiable et 'NI' à une entité non identifiable.

## 2.3 Analyse du langage EXPRESS

### 2.3.1 Choix du niveau d'abstraction et d'analyse

Comme il a été vu au premier chapitre, les acteurs d'un projet ont besoin de standards d'échanges pour les données métiers. Le domaine de la modélisation d'information connaît un besoin équivalent de standardisation [Lemesle00]. Ainsi la façon de modéliser un ensemble d'informations suit des standards comme XSD, UML, ou EXPRESS. L'OMG (Object Management Group) est une association d'industriels de l'informatique et de services visant à standardiser les formalismes de modélisation [OMG06]. Elle a défini d'un standard générique, le MOF (Meta Object Facility) proposant une approche pyramidale de la modélisation d'information, illustrée sur la figure 2.6. Le but est de situer le langage EXPRESS par rapport à cette approche, supposée *universelle*.

Dans le formalisme MOF, à la base de la pyramide se trouve le monde réel $M_0$, c'est-à-dire l'ensemble des données d'un projet particulier. Il s'agit du modèle d'ouvrage, instance d'un modèle de données tel que les IFC. Au niveau $M_1$ se trouve le modèle de données définissant leur structure. Le modèle IFC appartient ainsi au niveau $M_1$. Ensuite le méta-modèle est la définition du langage de description, EXPRESS s'intègre donc au niveau $M_2$. Enfin, l'unification des langages de description autour des mêmes mécanismes s'effectue au niveau $M_3$, le méta-méta-modèle, dont l'OMG donne une définition explicite avec le MOF. De manière générale, le passage du niveau $M_{n+1}$ à $M_n$ est une instanciation : le MOF instancie des méta-modèles, qui instancient des modèles de données, qui eux-mêmes instancient des modèles d'ouvrage. Pour éviter un nombre infini de niveaux d'abstraction, le MOF est lui-même une instance du MOF. Cette capacité réflexive permet de clore ce formalisme de modélisation.

Dans ce paradigme, notre analyse se situera au niveau d'abstraction $M_2$, niveau auquel se

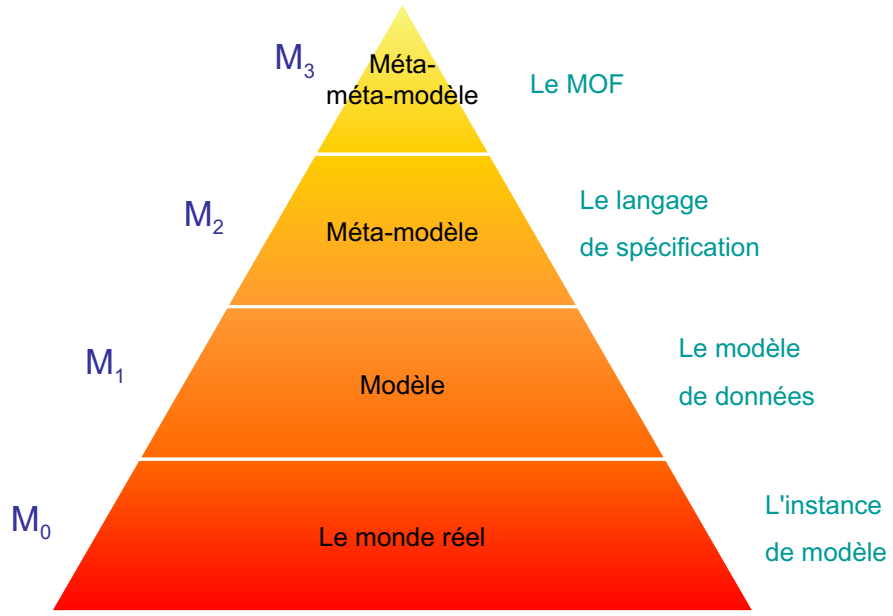


**FIGURE 2.6 :** Les niveaux d'abstraction définis par le MOF, d'après [OMG06].

trouve le langage EXPRESS : d'une part, il n'est pas possible de développer notre analyse au niveau du méta-méta-modèle. En effet, à cause de son mécanisme d'héritage spécifique, le langage EXPRESS n'est en réalité pas compatible avec le méta-méta-modèle MOF. De plus, le formalisme de modélisation basé sur le langage EXPRESS (et donc limité au niveau $M_2$) suffit car EXPRESS est auto-instanciable à l'instar de MOF. D'autre part, nous souhaitons développer un modèle de comparaison générique, c'est-à-dire indépendant du modèle de données. Cela implique donc une analyse à un niveau d'abstraction strictement supérieur à $M_1$.

C'est pourquoi nous allons analyser non pas les concepts d'un modèle de données particulier, comme IFC ou IFC-Bridge, mais la façon dont EXPRESS traite les différentes notions de modélisation orientée objets.

### 2.3.2 Description des concepts du langage EXPRESS

Le langage EXPRESS [ISO94a] permet de spécifier des données. Contrairement à UML, il ne permet pas de définir des comportements sur les objets, il s'agit seulement de décrire leur état.

Pour cela, ce langage dispose de plusieurs types de données. Tout d'abord les types *simples* permettent de stocker des états élémentaires. Il s'agit des nombres entiers, nombres réels, structures booléennes, chaînes de caractères, types binaires. Les types *agrégation* sont des ensembles d'états de même type, avec différents modes de stockage : les listes ordonnées, les ensembles non ordonnés avec ou sans répétition, etc. Les *énumérations* modélisent des états avec une valeur parmi plusieurs possibles. Par exemple, la notion de `saison` peut être vue comme un état pouvant prendre comme valeur `printemps`, `été`, `automne` ou `hiver`.

Il existe aussi des types définis par le concepteur. Tout d'abord, le type *défini* permet de déclarer un pseudonyme pour un type. Par exemple, il est possible de déclarer le type `jour`, pseudonyme d'un entier. Ensuite le type *sélection* permet de définir des objets dont le type appartient à un ensemble prédéfini. On pourrait par exemple définir un objet de type `numérique` c'est-à-dire un objet de type nombre entier, nombre réel, binaire ou structure booléenne :

```
TYPE Numérique = SELECT
        (INTEGER, REAL, BINARY, BOOLEAN);
END_TYPE;
```

Enfin le type *entité* est le plus important. Il permet de définir des objets avec plusieurs attributs, qui peuvent être eux-mêmes de type entité. Voici un premier exemple :

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

Il s'agit d'une entité dont le nom est `point` et possède trois attributs *explicites* de type simple (nombre réel), les trois coordonnées `x`, `y` et `z`. Un attribut explicite possède un nom et un type. Toute instance d'une entité doit donner une valeur à chacun de ses attributs explicites, sauf si cet attribut est déclaré optionnel. Voici un autre exemple :

```
TYPE vector = LIST [3:3] OF REAL;
END_TYPE;

ENTITY circle;
  centre : point;
  radius : REAL;
  axis   : vector;
DERIVE
  area      : REAL := PI*radius**2;
  perimeter : REAL := 2.0*PI*radius;
END_ENTITY;
```

Il permet de définir un objet de type cercle. Tout d'abord un type défini `vector` est déclaré, comme étant une liste de trois éléments. En effet, dans `LIST [3:3]`, le premier chiffre indique le nombre minimum d'éléments dans la liste et le deuxième le nombre maximum. Ensuite, l'entité cercle est déclarée. Il possède trois attributs explicites : le centre de type `point` défini dans l'exemple précédent, un rayon de type réel, et un axe, de type `vector`. Il possède en outre des attributs dérivés, c'est-à-dire des attributs dont la valeur est calculable à partir de la définition même de l'entité. Ainsi, à partir de la définition d'un cercle, il est possible d'en déduire sa surface et son périmètre. Le langage EXPRESS possède donc un moteur d'expression avec des opérateurs arithmétiques. Il existe enfin une dernière famille d'attributs, les attributs `inverse`, illustrée par l'exemple suivant :

```
ENTITY DiscreteCurve;
  controlPoints : LIST [0:?] OF ControlPoint;
```

```
END_ENTITY;

ENTITY ControlPoint;
  coordinates : point;
INVERSE
  fromCurve : SET [0:?] OF DiscreteCurve FOR controlPoints;
END_ENTITY;
```

Ici sont déclarées deux entités. La première correspond à la définition d'une courbe discrète à partir d'une liste de points de contrôle. Cette liste n'a pas de taille prédéfinie (`[0:?]` signifie de taille plus grande ou égale à 0). La deuxième entité est le point de contrôle qui contient un point géométrique pour les coordonnées et un attribut (ou lien) inverse. Ce dernier permet de rappeler quelles courbes discrètes utilisent ce point de contrôle.

En outre, dans un modèle de données EXPRESS, il est possible de définir des règles et des contraintes permettant de renforcer le typage de la structure. Nous retiendrons ici deux méthodes pour contraindre le modèle : les règles d'unicité et les clauses WHERE. Les autres méthodes ne concernent pas les entités directement. Pour expliquer les règles d'unicité, voici un autre exemple de modèle EXPRESS :

```
ENTITY PersonName;
  last     : STRING;
  first    : STRING;
  middle   : STRING;
  nickname : STRING;
UNIQUE
  ur1 : nickname
END_ENTITY;

ENTITY Employee;
   name : PersonName;
   officeNumber : INTEGER;
UNIQUE
  ur2 : name, officeNumber;
END_ENTITY;
```

La première entité `PersonName` définit l'identification d'une personne. La règle `ur1` indique que *toutes* les instances de l'entité PersonName doivent avoir un `nickname` différent. Dans la deuxième entité, la règle `ur2` indique que toutes les instances doivent avoir un couple (`name`, `officeNumber`). Cependant, deux employés peuvent être affectés au même numéro de bureau (attribut `officeNumber`). L'autre contrainte, nommée *clause WHERE*, permet de fixer un domaine de validité sur des attributs :

```
ENTITY UnitVector;
  x : REAL;
  y : REAL;
  z : REAL;
WHERE
  length1 : x**2 + y**2 + z**2 = 1.0;
END_ENTITY;
```

Dans cette entité, une contrainte est posée sur la longueur du vecteur représenté par l'entité `UnitVector`. En effet, la contrainte `length1` impose que sa longueur soit égale à 1 (l'opérateur `**` est celui de l'exposant).

Enfin, le dernier concept qui nécessite une attention particulière pour la comparaison structurelle est celui de ***l'héritage***. Le langage EXPRESS propose pour cela un mécanisme plus complexe que celui des langages de programmation orientés objets comme C++, java, C#, etc. En effet, il permet de concevoir un *héritage simple*, comme dans les langages de programmation, mais en suggère d'autres plus complexes. Voici un premier exemple d'héritage simple conçu par EXPRESS :

```
ENTITY animal
  ABSTRACT SUPERTYPE OF (ONEOF(cat, horse));
  name : STRING;
END_ENTITY;

TYPE sex = ENUMERATION OF (male, female);
END_TYPE;

ENTITY human
  gender : sex;
END_ENTITY;

ENTITY cat
  SUBTYPE OF (animal);
...
END_ENTITY;

ENTITY horse
  SUBTYPE OF (animal);
...
END_ENTITY;

ENTITY centaur
  SUBTYPE OF (horse, human);
...
END_ENTITY;
```

A partir d'une entité mère `animal`, on obtient deux entités filles correspondant à deux types d'animaux. Chacune de ces entités filles possède tous les attributs de l'entité mère, à savoir `name` dans cet exemple. Dans le formalisme EXPRESS, il s'agit de l'héritage `ONEOF`. Comme en C++, il est aussi possible de créer des *héritages multiples*. Ainsi l'entité `centaur` hérite, comme son nom l'indique, des attributs des entités `human` et `horse`. Cependant, EXPRESS propose deux autres mécanismes d'héritage. Continuons l'analyse à travers un exemple :

```
ENTITY person
  SUPERTYPE OF (employee ANDOR student);
...
END_ENTITY;
ENTITY employee
```

```
  SUBTYPE OF (person);
...
END_ENTITY;
ENTITY student
  SUBTYPE OF (person);
...
END_ENTITY;
```

Ici, une personne est soit un employé, soit un étudiant, soit les deux ! On peut ainsi définir un objet qui est instance de l'entité `employee&student`. Ce mécanisme est assimilable à un héritage multiple implicite. Enfin, le dernier mécanisme d'héritage est `AND`. Si on l'utilise dans l'exemple précédent, il ne serait pas possible de créer une instance de `employee` ou de `student`. On pourrait seulement définir un objet de type `employee&student`.

Ce panorama des fonctionnalités du méta-modèle EXPRESS nécessite une classification. Cette dernière permettra en effet d'identifier quels concepts ont un lien avec la comparaison structurelle.

### 2.3.3 Catégorisation des fonctionnalités du langage EXPRESS

Nous souhaitons orienter l'analyse du langage EXPRESS autour de la notion d'entité complexe (cf. section 1.3.2). A cette fin, il convient de distinguer d'une part l'état de l'entité, d'autre part ses liens avec les autres entités. De plus, le langage EXPRESS donne la possibilité de définir un domaine de validité et des valeurs calculées à partir des données existantes. C'est pourquoi nous proposons trois catégories de fonctionnalités :

- **La définition d'entité élémentaire (DE)** : elle regroupe les fonctionnalités permettant de définir les entités d'un modèle de données, sans prendre en compte ses liens avec les autres entités du modèle, conformément à la définition de [Urtado98].
- **Les propriétés calculées (PC)** : elles correspondent à des caractéristiques de l'entité calculables automatiquement, à partir du modèle de données, quelle que soit l'instance.
- **La structuration de l'entité complexe (SE)** : elle englobe tous les moyens permettant de hiérarchiser et lier les entités définies.

Parmi les fonctionnalités abordées précédemment, les suivantes appartiennent à la catégorie DE :

- type simple, type défini, type *ENUMERATION*,
- type *SELECT*,
- agrégation d'éléments non-entités (`ARRAY`, `LIST`, `BAG` et `SET`),
- attributs explicites non-entités,
- spécialisation d'attributs non-entités.

En effet, l'état d'une entité élémentaire est la donnée de ses attributs qui ne sont pas des références vers des entités. Si, à la suite d'un héritage, des attributs non-entités se spécialisent (passage d'un type NUMBER à un type REAL par exemple), il s'agit bien d'un changement de définition de l'entité. Voici à présent les fonctionnalités de la catégorie PC :

- attributs `DERIVE`,
- contraintes locales : règles d'unicité et clause `WHERE`,

- redéclaration d'un attribut explicite en attribut `DERIVE`.

En effet, les attributs DERIVE sont calculables, quelle que soit l'instance, à partir d'un algorithme donné par le modèle de données. Les contraintes sont aussi des propriétés calculables pour restreindre le domaine de validité d'une valeur ou en assurer son unicité parmi un ensemble. Enfin, lors d'un héritage, si un attribut explicite devient `DERIVE`, il devient calculable à partir du modèle de données. Enfin, la catégorie SE se compose des concepts suivants :

- agrégation d'entités,
- attributs explicites d'entités,
- spécialisation d'attributs entités,
- attributs inverse,
- techniques d'héritage.

Les attributs explicites référençant des entités, directement ou via des agrégations, créent des liens entre entités élémentaires pour former des entités complexes. De plus, les attributs rendent bidirectionels les liens entre entités. La spécialisation d'attributs entités modifie les liens. Enfin les différentes techniques d'héritage structurent les entités en regroupant des propriétés communes au sein d'entités parents.

Les diverses fonctionnalités ont été catégorisées. Ceci permet à présent d'identifier quelles fonctionnalités EXPRESS doit exploiter notre méthode de comparaison structurelle.

### 2.3.4 Support du langage EXPRESS pour la comparaison structurelle

L'objectif est ici de restreindre le cadre d'étude pour simplifier la problématique de comparaison et de synchronisation. Pour cela, revenons dans un premier temps à la catégorisation précédente. La définition d'entités (DE) et la structuration d'entités (SE) sont nécessaires à la comparaison structurelle qui compare les états des entités complexes du modèle, en prenant en compte les liens de dépendance entre entités. Les propriétés calculées (PC), quant à elles, participent aussi à la description des états d'une entité. Cependant, cette description est automatiquement calculable à partir de la définition modèle de données. Or, dans un objectif de versionnement orienté objets, l'information de changement d'une propriété calculée n'a pas besoin d'être stockée. En effet, quelle que soit la version d'une instance de modèle, il est possible de calculer cette propriété automatiquement, elle n'est donc pas stockée[2]. C'est pourquoi nous pouvons en déduire qu'il est inutile de prendre en compte des éléments appartenant à la catégorie PC pour la comparaison structurelle.

Dans un deuxième temps, une analyse des modèles de données, sur lesquels se basent nos travaux, permet d'identifier quelles fonctionnalités du langage EXPRESS ne sont jamais utilisées. Rappelons les modèles de données analysés : IFC, IFC-Bridge et Tas-Arm. IFC-Bridge contient en fait le modèle IFC mais n'utilise pas de fonctionnalités supplémentaires EXPRESS. C'est pourquoi seuls les résultats de IFC-Bridge et de Tas-Arm sont mentionnés. Tout d'abord, les deux modèles n'exploitent pas tous les types d'agrégation et TasArm ne possède pas d'objets de type `SELECT`.

[2]Sauf à des fins d'optimisation, pour éviter de recalculer une valeur si elle est nécessaire plusieurs fois à l'application.

Être capable de gérer tous les types de données ne rend pas le travail de comparaison plus complexe. Par conséquent toutes les fonctionnalités associées aux types ou plus généralement à la catégorie **DE** sont pris en compte dans notre travail.

Ensuite, les deux modèles utilisent seulement des techniques d'héritage simple ONEOF. Ce dernier point est fondamental pour la suite : nous choisissons de restreindre le champ d'application de notre méthode de comparaison structurelle à des modèles de données n'utilisant que de l'héritage simple. Cette restriction n'affaiblit cependant pas la portée de nos travaux. En effet, nous avons développé une méthode pour transformer un modèle utilisant de l'héritage multiple, ANDOR ou AND en un modèle n'exploitant que de l'héritage simple. Les détails de cet algorithme sont rapportés à l'annexe B.

Le tableau 2.2 résume les résultats de notre analyse. A partir des fonctionnalités principales du langage EXPRESS, nous les avons catégorisées et pour chaque modèle de données, nous avons vérifié si cette fonctionnalité est utilisée. Enfin, nous en déduisons le cadre théorique pour la méthode de comparaison structurelle à concevoir.

### 2.3.5 Application du cadre d'étude au langage EXPRESS

Cette partie vise à vérifier si les hypothèses, exprimées dans la section 2.2.1, sont valides dans le cadre de la modélisation EXPRESS. Tout d'abord, l'hypothèse 1 sur la restriction de la sémantique des liens est vérifiée pour n'importe quel modèle écrit dans ce langage. Voici un exemple représentatif du cas général :

```
ENTITY Personne;
  nom : STRING;
  coordonnées : Adresse;
END_ENTITY;
ENTITY Adresse;
  rue : STRING;
  cp : INTEGER;
  ville : STRING;
END_ENTITY;
```

Une entité élémentaire, instance de l'entité `Personne`, a un lien vers une autre entité élémentaire, instance de l'entité `Adresse`. Ce lien est orienté de la personne vers l'adresse, et son nom est `coordonnées`.Il s'agit donc d'un lien labellisé, mais il n'a pas d'attributs propres, conformément à l'hypothèse 1.

Ensuite l'hypothèse 2 suppose l'existence d'une racine de parcours. Or rien ne garantit l'existence d'une telle entité au sein d'une instance de n'importe quel modèle EXPRESS. Cette hypothèse ne peut donc être vérifiée qu'en descendant au niveau d'abstraction $M_1$, c'est-à-dire en analysant chaque modèle de données utilisé dans le cadre de nos travaux.

L'hypothèse 3, quant à elle, dépend uniquement du langage EXPRESS. En effet, tous les attributs du langage EXPRESS possèdent un type défini par ce langage. Il est donc possible de définir des opérations de comparaison génériques, c'est-à-dire valides pour n'importe quel modèle de

| FONCTIONNALITÉ | CATÉGORIE | IFC-BRIDGE | TAS-ARM | COMPARAISON STRUCTURELLE |
|---|---|---|---|---|
| Types simples, définis, ENUMERATION | **DE** | * | * | √ |
| Type SELECT | **DE** | * | | √ |
| Agrégations typées | | | | |
| Array | **DE/SE** | | | √ |
| List | **DE/SE** | * | * | √ |
| Bag | **DE/SE** | | | √ |
| Set | **DE/SE** | * | * | √ |
| Attributs | | | | |
| Explicites | **DE/SE** | * | * | √ |
| Dérives | **PC** | * | * | |
| Inverses | **SE** | * | * | √ |
| Contraintes | | | | |
| Règles d'unicité | **PC** | * | * | |
| Clauses WHERE | **PC** | * | * | |
| Technique d'héritage | | | | |
| ONEOF (simple) | **SE** | * | * | √ |
| multiple | **SE** | | | |
| ANDOR | **SE** | | | |
| AND | **SE** | | | |
| Héritage d'attributs | | | | |
| Spécialisation | **DE/SE** | * | * | √ |
| Redéclaration en DERIVE | **PC** | * | * | |

**TABLEAU 2.2:** Cadre d'étude de la comparaison structurelle à partir des fonctionnalités EXPRESS supportées par les modèles de données étudiés. Certaines fonctionnalités appartiennent à deux catégories DE/SE car si la fonctionnalité concerne une entité, il s'agit d'une structuration SE, sinon il s'agit d'une définition DE.

données EXPRESS. A cette fin la section suivante définit ces opérations de comparaison pour chaque type possible.

L'hypothèse 4 tient d'abord à rappeler que l'on compare des instances (les données elles-mêmes) et non des modèles de données. Pour cela, chaque entité élémentaire du modèle d'ouvrage est l'instance d'une ou plusieurs entités EXPRESS du modèle de données. Via l'utilisation de l'héritage simple (seul mécanisme d'héritage retenu pour nos travaux, cf. section précédente), les entités peuvent être partiellement ou totalement comparables. Dans tous les cas, cette hypothèse est vérifiée.

Enfin l'hypothèse 5 est elle aussi vérifiée, grâce notamment aux protocoles d'échanges de données standardisés (STEP-21), vu à la section 1.2.2. Cependant, nous verrons dans la section

suivante que ces identifications ne sont pas toujours persistantes.

A partir de cette analyse du langage EXPRESS, le travail consiste à présent à concevoir l'algorithme de comparaison structurelle et de l'appliquer aux modèles de données étudiés IFC et IFC-Bridge.

## 2.4 Application au modèle IFC-Bridge et conception de la comparaison structurelle

### 2.4.1 Application du cadre d'étude au modèle IFC-Bridge

Suite à la vérification d'hypothèses d'une modélisation EXPRESS générale, vue à la section précédente, certaines d'entre elles nécessitent une analyse au niveau du modèle de données. Pour cela, l'objectif consiste à vérifier sur le modèle IFC-Bridge deux points :

- L'hypothèse 2 sur l'existence d'une racine de parcours.
- Distinguer les entités identifiables des entités non identifiables, suite à l'hypothèse 5.

Le modèle IFC 2x3 possède 653 entités, et l'extension IFC-Bridge ajoute environ 15 entités au modèle IFC (le nombre exact sera formulé lors de l'intégration complète de IFC-Bridge dans le modèle IFC, en 2008). [Liebich04] permet d'identifier une racine potentielle du graphe de données. Il s'agit de l'entité `IfcProject` qui représente la racine des éléments de structure spatiale. Pour rappel, un projet décrit en IFC est divisé de manière macroscopique en éléments de structure spatiale : projet, site, bâtiment, étage, espace. Ces éléments sont structurés en arbre dont la racine est un projet, qui contient un ou plusieurs sites, composé lui-même d'un ou plusieurs bâtiments, etc. Du point de vue de la spécification logique, ces entités sont en fait reliées par des entités de relation, comme cela a été vu au premier chapitre, sur la figure 1.10.

L'objectif consiste donc d'abord à démontrer que l'entité *IfcProject* est racine du graphe des éléments de structure spatiale. La figure 2.7 présente un extrait des entités du modèle de données IFC avec les liens d'héritage associés. Dans cet extrait, les entités de structure spatiale sont visibles en jaune (nous considérons dans ces travaux que l'entité `IfcProject` est aussi un élément de structure spatiale, celle de niveau le plus élevé). Les entités en orange regroupent des propriétés communes à d'autres entités. Parmi ces dernières, on remarque que toutes les entités de structure spatiale héritent de `IfcObjectDefinition`. Cette entité possède en particulier deux attributs permettant de hiérarchiser les éléments de structure spatiale. A cette fin, ces attributs font appel à une entité de relation, `IfcRelDecomposes`. L'implémentation de ce procédé est alors le suivant : l'entité `IfcRelDecomposes` possède deux attributs reliant l'entité parente (via l'attribut `RelatingObject`) et les entités enfants (via l'attribut `RelatedObjects`). Or ces attributs correspondent à des arêtes orientées dans le graphe de données. Le modèle IFC rend bidirectionnelles ces arêtes grâce aux attributs *inverse* de l'entité *IfcObjectDefinition* (l'explication des liens inverses a été vue à la section 2.3.2). D'après [Liebich04] et [IAI06], les éléments de structure spatiale sont toujours reliés via une entité de relation qui hérite de `IfcRelDecomposes` : `IfcRelAggregates`. Au niveau des instances, la hiérarchisation de ces éléments possède une structure-type illustrée par la figure 2.8.

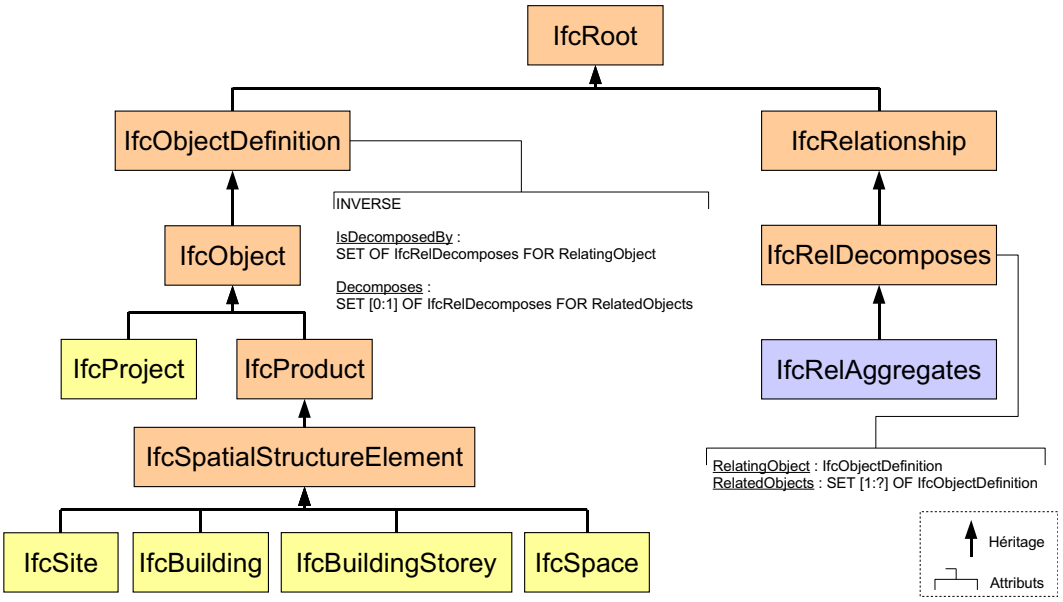


**FIGURE 2.7 :** Extrait du diagramme d'héritage des entités du modèle IFC 2x3.

Cette figure ne détaille pas la multiplicité des éléments site, bâtiment, étage et espace. En fait, chaque entité parente peut être reliée (via l'entité `IfcRelAggregates`) à plusieurs entités enfants. Dans tous les cas, grâce aux liens inverses, l'entité *IfcProject* est bien racine du graphe des éléments de structure spatiale. Ce n'est évidemment pas la seule racine, n'importe quelle entité de la figure 2.8 pourrait servir de racine. Cependant, la première raison de ce choix est d'ordre sémantique : un projet représente le conteneur de tous les autres éléments de structure spatiale. Mais la raison la plus importante est la suivante : le modèle IFC 2x3 garantit qu'il existe au plus une seule instance de l'entité `IfcProject`, contrairement aux autres entités, comme le montre l'extrait du code EXPRESS ci-dessous :

```
RULE IfcSingleProjectInstance FOR (IfcProject);
    WHERE
      WR1 : SIZEOF(IfcProject) <= 1;
END_RULE;
```

Nous verrons plus loin que cette propriété est très utile pour la conception de l'algorithme de comparaison structurelle, qui sera vue à la section 2.4.3.

Il reste cependant impossible de montrer que l'instance de l'entité *IfcProject* est racine de n'importe quel modèle d'ouvrage. En effet, rien n'empêche de définir une information (par exemple de matériau) sans qu'elle soit attachée au projet. Cependant, [Liebich04] et [IAI06] imposent que toutes les informations utilisées dans le projet soient reliées à ce dernier. Dans ce but, les éléments de construction sont reliés aux éléments de structure spatiale via l'entité de relations `IfcRelContainedInSpatialStructure`. Ainsi chaque entité de structure spatiale est racine du graphe de tous les éléments le concernant. Nous pouvons alors supposer que l'instance de l'entité `IfcProject` est racine du graphe de données associé à ce projet. Nous vérifierons ce

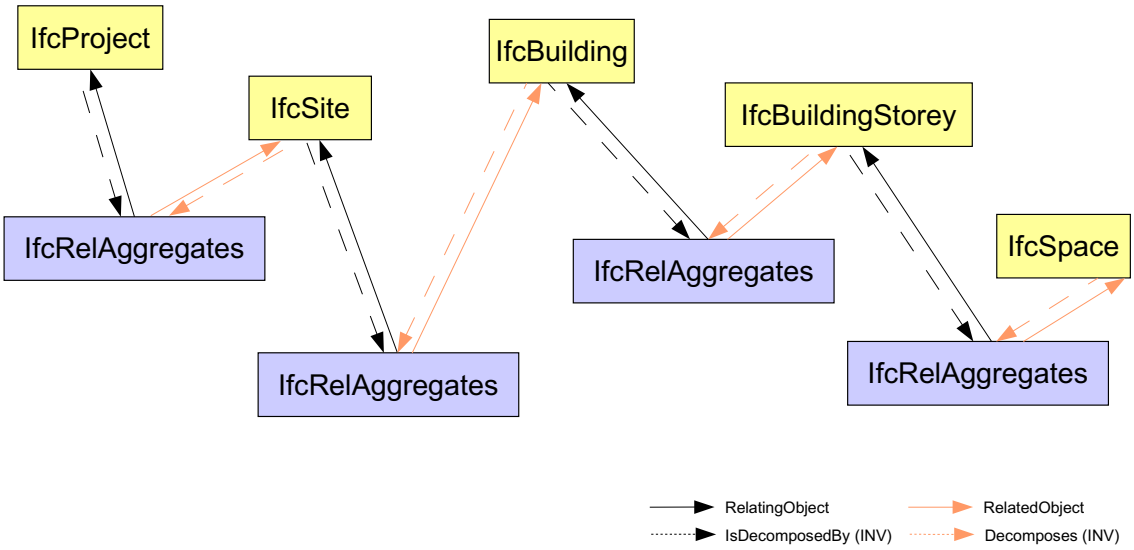


**FIGURE 2.8 :** Graphe de dépendance des éléments de structure spatiale.

point lors de l'implémentation de la comparaison sur plusieurs modèles IFC.

Le modèle IFC vérifie ainsi l'hypothèse 2. A présent, concernant l'hypothèse 5, rappelons que le langage EXPRESS ne donne pas de moyen direct pour identifier les instances d'entités EXPRESS. C'est pourquoi le protocole STEP propose un mécanisme d'identification, qui a été étudié à la section 1.2.2 du premier chapitre, pour la persistance d'instances de modèles EXPRESS. Pour rappel, lors de la sérialisation des données en un fichier STEP-21, chaque entité se voit attribuer un nombre entier, ce dernier sert de référence pour les attributs explicites. Voici un rappel de la figure 1.4 :

```
#1=IFCSIUNIT(*,.TIMEUNIT.,$,.SECOND.);
#2=IFCSIUNIT(*,.MASSUNIT.,$,.GRAM.);
#3=IFCDIMENSIONALEXPONENTS(1,0,0,0,0,0,0);
#10=IFCCARTESIANPOINT((0.,0.,0.));
#11=IFCDIRECTION((0.,0.,1.));
#12=IFCDIRECTION((1.,0.,0.));
#13=IFCAXIS2PLACEMENT3D(#10,#11,#12);
```

`IfcAxis2Placement3d` possède trois attributs explicites qui référencent un objet de type `IfcCartesianPoint` et deux objets de type `IfcDirection`. Cependant, dans la pratique, la persistance de ces identifications à travers des cycles sauvegarde/chargement successifs n'est pas garantie. Nous avons ainsi observé des changements complets d'identification lors d'un simple transfert d'un modèle IFC entre deux logiciels commerciaux de CAO architecte, entre *ArchiCAD 9* et *Revit Building* par exemple. Comme dans le problème de racine de parcours, l'approche générique pure, se basant sur la seule définition du langage EXPRESS, ne suffit pas. Le travail consiste donc à distinguer les entités identifiables des autres dans le cas du modèle IFC et IFC-Bridge. Le modèle IFC-Bridge ne proposant pas de nouvelles façons d'identifier les objets, nous portons cette analyse sur le modèle IFC. Les résultats de cette recherche sont cependant direc-

tement applicables à l'extension IFC-Bridge.

Parmi les 653 entités EXPRESS contenues dans ce modèle, nous distinguons les entités identifiables des entités non identifiables en partant d'abord d'une propriété de l'identification : elle donne une valeur unique à chaque objet d'un ensemble. Nous listons donc les entités du modèle possédant des règles d'unicité. Ces dernières permettent de donner des valeurs uniques parmi un ensemble d'objets *de même type*. 15 entités possèdent ainsi un attribut soumis à une contrainte d'unicité. Parmi elles, 8 entités concernent des propriétés de matériau, comme le montre la figure 2.9.

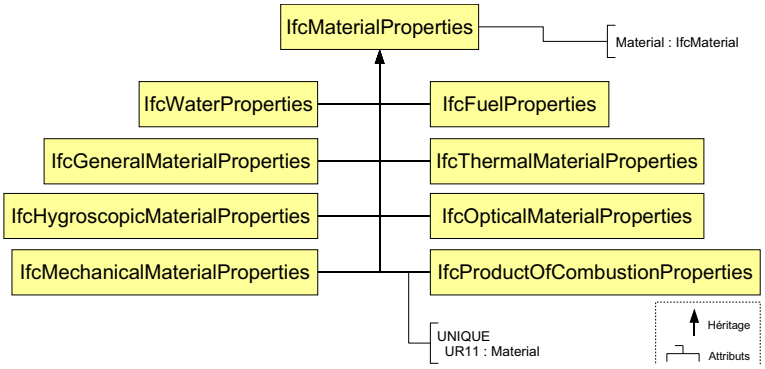


**FIGURE 2.9 :** Entités de propriétés de matériau avec contraintes d'unicité.

L'entité `IfcMaterialProperties` définit un attribut `Material`, et dans chaque entité héritant de `IfcMaterialProperties` (sauf `IfcExtendedMaterialProperties` non représenté sur la figure), une contrainte d'unicité est fixée. Sémantiquement, cela signifie qu'on ne peut affecter qu'une seule propriété de matériau de chaque famille (mécanique, hydraulique, optique, etc.) à un matériau donné. Il ne s'agit donc pas d'un véritable moyen d'identification. Parmi les sept restantes, les attributs concernés ont un label associable à une identification : `ID`, `ActionID`, `Name`, `GlobalId`, etc. La figure 2.10 montre les entités ayant de tels attributs (en orange) dans le diagramme d'héritage du modèle IFC 2x3.

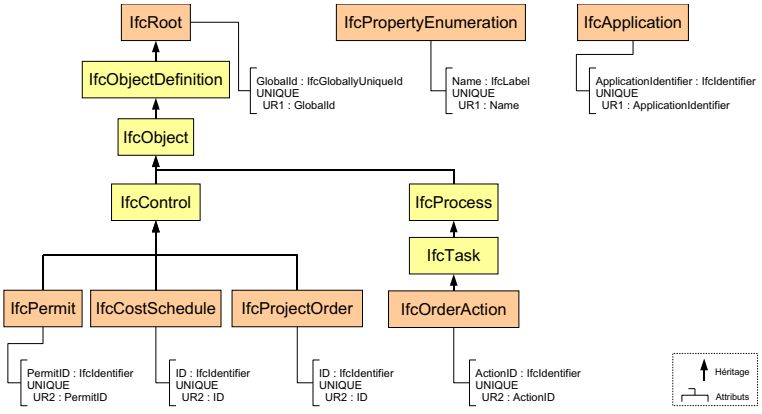


**FIGURE 2.10 :** Diagramme d'héritage des entités potentiellement identifiables.

Tout d'abord, l'entité IfcApplication pourrait être considérée comme une entité identifiable grâce à l'attribut ApplicationIdentifier. Cependant, nous observons que cette entité est utilisée seulement par l'entité IfcOwnerHistory, comme le montre le code ci-dessous :

```
ENTITY IfcOwnerHistory;
        OwningUser : IfcPersonAndOrganization;
        OwningApplication : IfcApplication;
        State : OPTIONAL IfcStateEnum;
        ChangeAction : IfcChangeActionEnum;
        LastModifiedDate : OPTIONAL IfcTimeStamp;
        LastModifyingUser : OPTIONAL IfcPersonAndOrganization;
        LastModifyingApplication : OPTIONAL IfcApplication;
        CreationDate : IfcTimeStamp;
END_ENTITY;
```

Cela signifie que quelle que soit l'instance du modèle IFC, pour atteindre une instance de IfcApplication depuis l'instance de IfcProject, l'algorithme de parcours passe par une instance de IfcOwnerHistory. Or cette dernière ne possède aucun attribut potentiel d'identification. D'après la section 2.2.3, les liens d'une entité non identifiable vers une entité identifiable ne sont pas parcourus. Donc si IfcApplication est considérée comme identifiable, il ne sera jamais parcouru. C'est pourquoi nous considérons par la suite IfcApplication comme non identifiable. Le problème est similaire avec IfcPropertyEnumeration. Ce dernier n'est utilisé que par l'entité IfcPropertyEnumeratedValue :

```
ENTITY IfcPropertyEnumeratedValue
 SUBTYPE OF (IfcSimpleProperty);
        EnumerationValues : LIST [1:?] OF IfcValue;
        EnumerationReference : OPTIONAL IfcPropertyEnumeration;
 WHERE
        WR1 : NOT(EXISTS(EnumerationReference)) OR
               (SIZEOF(QUERY(temp <* EnumerationValues |
                temp IN EnumerationReference.EnumerationValues))
                = SIZEOF(EnumerationValues));
END_ENTITY;
```

De plus, IfcPropertyEnumeratedValue hérite de IfcSimpleProperty qui hérite de IfcProperty. Or, aucune de ces trois entités n'est potentiellement identifiable. C'est pourquoi nous ignorons aussi l'identifiabilité de IfcPropertyEnumeration. Les cinq dernières entités potentiellement identifiables appartiennent au même graphe d'héritage. L'entité IfcRoot est fondamentale pour le modèle IFC, puisque 300 entités sur les 653 du modèle héritent directement ou indirectement de IfcRoot. L'attribut correspond bien à une identification globale (d'après le label de l'attribut) et persistant (d'après le type de l'attribut). On peut donc considérer que IfcRoot est une entité identifiable, ainsi que les 300 autres entités qui en héritent. Quant aux quatre dernières entités IfcPermit, IfcCostSchedule, IfcProjectOrder et IfcOrderAction, elles héritent toutes de IfcRoot. Elles ont donc chacune deux possibilités d'identification. Pour assurer une meilleure cohérence et une meilleure flexibilité de comparaison, seul l'identifiant de IfcRoot est pris en compte. En effet, considérer les identifiants des quatre entités citées précédemment ne permettrait d'effectuer des comparaisons qu'entre éléments de même type exactement (deux instances de IfcPermit ou deux instances de IfcProjectOrder par exemple). Donc les attributs

`ID`, `ActionID`, `PermitID` sont vus dans la suite comme des attributs de l'état de l'entité.

Les modèles de données IFC et IFC-Bridge sont donc compatibles avec les hypothèses du cadre théorique. Il convient alors de concevoir l'algorithme de comparaison structurelle. Cette conception commence par l'examen en détail de la comparaison entre deux entités élémentaires EXPRESS.

### 2.4.2 Définition et comparabilité des entités élémentaires EXPRESS

Au premier chapitre les notions d'entités élémentaires et d'entités complexes ont été définies, cf. section 1.3.2. Cette partie vise donc à spécifier ce que représente une entité élémentaire par rapport à la notion d'entité en langage EXPRESS. Plus précisément, il s'agit de différencier la représentation de l'état d'un objet de celle de ses liens et ses dépendances avec d'autres objets. L'entité élémentaire est alors la donnée de son état seulement. Cette distinction est un problème fondamental car elle fixe implicitement la granularité de la comparaison. En effet, un élément qui fait partie de l'état d'une entité n'est plus considéré lui-même comme une entité. Le résultat de la comparaison ne sera donc pas particulier à cet élément mais global à l'entité élémentaire qui le contient.

Une entité élémentaire est une instance d'entité EXPRESS munie d'une partie de l'ensemble de ses attributs. Les éléments de la catégorie **DE** des fonctionnalités EXPRESS, vues à la section 2.3.3, représentent une première base. A cela, il convient d'ajouter les entités non identifiables, d'après la section 2.2.3. Voici les différentes familles d'attributs d'une entité élémentaire. D'abord il s'agit des attributs de types simples, ou énumération, synthétisés dans l'exemple suivant :

```
TYPE Saison = ENUMERATION OF (printemps, été, automne, hiver);
END_TYPE;
TYPE Label = STRING;
END_TYPE;
ENTITY AttributsSimple;
  unEntier : INTEGER;
  unRéel : REAL;
  uneChaineCaractere : STRING;
  unTitre : Label;
  uneSaison : Saison;
END_ENTITY;
```

Si l'attribut est de type défini pointant vers un type simple, comme `Label`, il est aussi pris en compte. Les algorithmes de comparaison pour déterminer l'équivalence ou non de ces attributs sont évidents. Ensuite les entités non identifiables font aussi partie de l'état d'une entité élémentaire :

```
ENTITY Point3D;
  x : REAL;
  y : REAL;
  z : REAL;
END_ENTITY;
```

```
TYPE Genre = ENUMERATION OF {Madame, Mademoiselle, Monsieur}
ENTITY Personne;
  id : STRING;
  nom : STRING;
  titre : Genre
  localisation : Point3D;
END_ENTITY;
```

L'entité élémentaire `Personne`, identifiable via `id`, possède l'attribut `localisation` qui référence une entité `Point3D`. Cette dernière n'est pas identifiable. Conformément à la section 2.2.3, la remontée des attributs `x`, `y` et `z` dans l'entité `Personne` ne s'effectue pas de manière individuelle, mais en regroupant tous ces attributs en un seul, via une sérialisation ou une fonction de hachage. La sérialisation d'une instance de `Point3D` serait par exemple : `Point3D(32.5,-8.96,1.03)`. Pour assurer la comparabilité des attributs, la fonction de sérialisation conserve la relation d'équivalence (cf. hypothèse 3) : soit $s$ la fonction de sérialisation, soient $a$ et $b$ deux attributs à comparer, la relation suivante doit être vérifiée : $a \equiv b \Rightarrow s(a) = s(b)$. De plus, cette fonction doit être injective : $s(a) = s(b) \Rightarrow a \equiv b$. Nous avons identifié des sérialisations vérifiant ces propriétés quelle que soit l'entité non identifiable EXPRESS. La fonction de hachage est une sérialisation particulière. Cette méthode transforme n'importe quelle structure en un nombre entier. La désérialisation est très difficile voire impossible. Mais la désérialisation n'est pas nécessaire ici, car la structure associée reste toujours en mémoire. Vis-à-vis de la sérialisation, les principaux avantages de la fonction de hachage sont les suivants :

- Performance de comparaison : un entier est une structure de longueur fixe en informatique (codé sur un certain nombre de bits), contrairement à une chaîne de caractère. La comparaison de clés de hachage est donc plus rapide.
- Économie de ressources : pour la même raison que celle exposée au point précédent, une clé de hachage demande peu de mémoire.
- Utilisation reconnue en informatique : il existe de nombreux algorithmes robustes proposant des fonctions de hachage. Cette technique est en effet utilisée aussi en cryptographie ou pour l'optimisation à l'accès de bases de données.

La fonction de hachage doit respecter les mêmes propriétés que pour la sérialisation : si $h$ est une fonction de hachage, $a$ et $b$ deux structures à comparer, on a : $a \equiv b \Rightarrow h(a) = h(b)$ (morphisme) et $h(a) = h(b) \Rightarrow a \equiv b$ (injectif). La dernière propriété n'est cependant pas vérifiée de manière générale, seulement de manière statistique [Knuth98]. Dans ces travaux nous avons néanmoins privilégié la fonction de hachage à la sérialisation pour les avantages sus-cités.

Les sélections représentent une autre catégorie pour définir des attributs d'une entité élémentaire. Nous rappelons qu'elles permettent à l'attribut de prendre une valeur venant de types différents parmi un ensemble défini par cette sélection. L'ambiguïté vient de la possibilité de définir un type sélection regroupant une entité identifiable et un autre type (type simple ou entité non identifiable). En réutilisant l'entité non identifiable `Point3D` de l'exemple précédent simplifié :

```
ENTITY Rue;
```

```
  id : STRING;
  nom : STRING;
  départ : Point3D
  arrivée : Point3D
END_ENTITY;

TYPE Lieu = SELECT
(Point3D, Rue);

ENTITY Personne;
  id : STRING;
  nom : STRING;
  localisation : Lieu;
END_ENTITY;
ENTITY
```

L'attribut `localisation` n'est plus un point 3D mais un élément de type Lieu qui est soit un point 3D, soit une rue. Dans le premier cas, il s'agit d'un attribut de l'entité élémentaire, alors que dans le deuxième, il s'agit d'un lien vers une autre entité identifiable. L'approche générique que nous avons choisie ne nous permet pas de changer la structuration des données en fonction d'une instance particulière.

Pour résoudre ce problème, nous avons analysé le modèle IFC 2x3 pour vérifier si ce cas se présente. Parmi les 46 types SELECT définis dans le modèle IFC 2x3, un seul fait intervenir des entités identifiables, c'est-à-dire des éléments héritant de l'entité `IfcRoot`, d'après la section 2.4.1 :

```
TYPE IfcStructuralActivityAssignmentSelect = SELECT
(IfcStructuralItem, IfcElement);
END_TYPE;
```

Or, `IfcStructuralItem` et `IfcElement` héritent tous les deux de `IfcRoot`, ils sont à ce titre identifiables. Donc, au sein du modèle IFC 2x3, il n'existe pas de types SELECT mélangeant des entités identifiables avec d'autres types d'éléments. Le cas d'ambiguïté ne se présente pas pour ce modèle. Si un attribut est de type SELECT non associé à des entités identifiables, la comparaison de deux instances s'effectue en deux temps :

1. Vérifier si les deux instances de l'attribut ont choisi le même type parmi l'ensemble proposé par le SELECT. Si ce n'est pas le cas, les instances sont différentes et la comparaison s'arrête là.
2. Comparer les instances en utilisant l'algorithme usuel associé au type commun.

Après les types simples, les entités non identifiables et les sélections, les types agrégations peuvent également définir des attributs d'une entité élémentaire :

```
TYPE Label = STRING;
END_TYPE;
ENTITY Point3D;
  x : REAL;
```

```
  y : REAL;
  z : REAL;
END_ENTITY;
ENTITY AttributsAggregations
  id : STRING;
  listeTitres : ARRAY [1:10] OF Label;
  ensembleRéels : BAG [0:?] OF REAL;
  suiteEntiers : LIST [0:?] OF INTEGER;
  ensemblePoints : SET [0:?] OF Point3D;
END_ENTITY;
```

Les algorithmes de comparaison de tels attributs dépendent alors du type d'agrégation, en particulier de leur propriété intrinsèque : l'ordre. Dans le cas `ARRAY` et `LIST`, l'ordre des éléments dans l'agrégation est pris en compte, donc la comparaison se fait deux à deux suivant un parcours simultané des deux agrégations à comparer, en complexité $O(n)$. Dans le cas `BAG` et `SET`, l'ordre des éléments n'a pas d'importance, donc si les deux agrégations possèdent les mêmes éléments dans un ordre différent, ils sont considérés comme équivalents. Une comparaison serait donc d'une complexité $O(n^2)$ dans le cas général. Néanmoins cette complexité est améliorable, en particulier dans le cas où les éléments des agrégations sont auparavant ordonnés. La comparaison redevient d'une complexité linéaire, et l'ordonnancement des éléments a une complexité de $O(n \log_2(n))$. Cet ordonnancement n'est effectué qu'une seule fois, même si la structure est appelée à être comparée plusieurs fois. Tous les objets EXPRESS nécessitent en contre partie une relation d'ordre. Pour les types simples, il suffit d'utiliser les relations d'ordre usuelles. Concernant les entités non identifiables et les sélections, les fonctions de hachage associent à ces structures des valeurs numériques qui permettent de les ordonner. De la même manière les agrégations sont aussi ordonnables via l'utilisation de fonctions de hachage (lorsque l'on définit un ensemble de listes d'objets par exemple).

Cette partie a permis de définir sans ambiguïté la notion d'entité élémentaire en langage EXPRESS, en particulier dans le cas des modèles IFC et IFC-Bridge. Les autres attributs des entités EXPRESS servent alors à créer des liens entre entités élémentaires, pour former des entités complexes, conformément au paradigme de [Urtado98]. L'objectif du travail consiste alors à concevoir l'algorithme de comparaison structurelle, sachant que les hypothèses du cadre d'étude théorique ont été validées.

### 2.4.3 Conception de l'algorithme de comparaison structurelle

L'objectif de la comparaison structurelle est d'affecter des statuts de comparaison aux entités élémentaires identifiables, comme cela a été vu à la section 2.2.2. Pour cela, nous avons choisi une approche consistant à effectuer un *parcours simultané* des deux graphes, associés respectivement à l'instance originale et à l'instance modifiée. L'avantage d'une telle méthode est de gagner en complexité algorithmique dans certains cas : une comparaison deux à deux de toutes les entités du modèle aurait une complexité de $O(n^2)$, au mieux $O(n \log_2(n))$ si les éléments sont triés. Dans le meilleur des cas, un parcours simultané permet de tendre vers une complexité linéaire $O(n)$. Cependant, aucune garantie théorique ne peut être donnée. En effet, si une entité

référence, via un attribut, un ensemble non ordonné d'entités élémentaires, la comparaison de cet attribut revient à comparer les éléments deux à deux. Nous supposons néanmoins que le parcours simultané permet d'augmenter les performances de l'algorithme dans le cas où le modèle d'ouvrage a peu changé entre deux étapes de conception.

A partir de l'instance originale et de l'instance modifiée, la comparaison s'effectue entre les deux graphes associés, après avoir traité les entités non identifiables (cf. section 2.2.3). La comparaison commence par l'entité racine de chaque instance. L'hypothèse 2 assure l'existence d'un tel élément. Dans le cas des modèles IFC et IFC-Bridge, l'unicité est aussi garantie. En effet, il existe une règle imposant l'existence d'au plus une entité `IfcProject` (cf. section 2.4.1). Le déroulement de la comparaison suit alors l'organigramme de la figure 2.11.

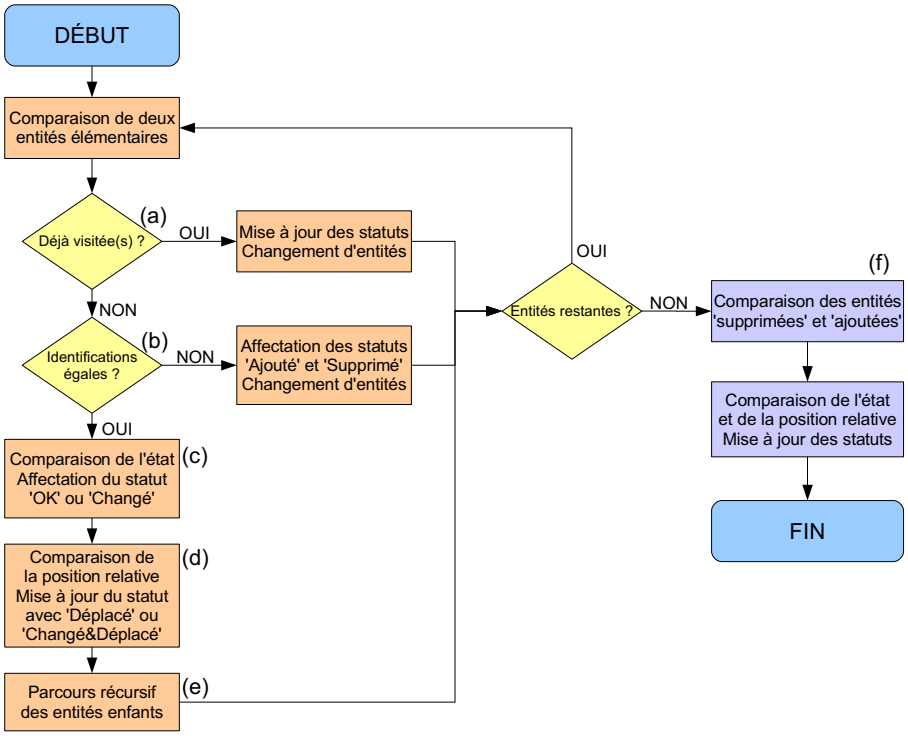


**FIGURE 2.11 :** Organigramme général de l'algorithme de comparaison structurelle.

Lors de la comparaison de deux entités élémentaires, le travail consiste d'abord à vérifier si une au moins des deux entités a déjà été visitée, c'est-à-dire qu'elle a été affectée d'un statut de comparaison (en (a) sur la figure 2.11). Si les deux entités sont concernées ou si l'une d'elles seulement est affectée des statuts 'Identique', 'Changé', 'Déplacé', 'Changé&Déplacé', alors la comparaison ne va pas plus loin pour ces entités. L'entité qui n'avait pas de statut (ainsi que ses entités descendantes) , devient 'Supprimé' (resp. 'Ajouté') si elle appartient à l'instance originale (resp. modifiée). La comparaison reprend avec d'autres entités en suivant le parcours récursif du graphe, sans visiter les enfants. Si aucune des deux entités n'a de statut, la comparaison continue. Si seule une des deux entités possède le statut 'Ajouté' ou 'Supprimé', ce dernier est ignoré et la comparaison continue.

La deuxième étape consiste à comparer les identifications persistantes des entités (en (b)). Si elles sont égales, alors la comparaison continue. Sinon, l'entité de l'instance originale (resp. modifiée) devient 'Supprimée' (resp. 'Ajoutée'), ainsi que toute sa descendance. Dans le graphe, les liens vers les enfants ne sont pas parcourus. La comparaison reprend avec d'autres entités selon le parcours récursif du graphe.

Ensuite, les états des entités sont comparés (en (c)). Cela revient à comparer chaque attribut des entités élémentaires (cf. section 2.4.2). Si au moins une valeur d'attribut diffère, alors les deux entités sont affectées du statut 'Changé'. Sinon, elles reçoivent le statut 'Identique'. En (d), la comparaison porte sur la position relative des entités dans leur graphe respectif, c'est-à-dire l'ensemble des identifications des entités parents. Si cet ensemble diffère, les entités reçoivent le statut 'Déplacé' ou 'Changé&Déplacé' selon le résultat de l'opération précédente.

Enfin, la comparaison recommence avec les entités enfants, via les attributs qui les référencent (en (e)). Lorsque le parcours récursif prend fin (tous les liens du graphe ont été parcourus) alors une dernière étape vise à comparer les entités supposées supprimées et celles supposées ajoutés (en (f)). Une correspondance d'identification est faite, avec une complexité de $O(n \log_2 n)$ si ces deux listes sont triées, puis une comparaison d'état et de position relative pour les entités qui ont la même identification persistante. Ce second parcours permet surtout de détecter des entités déplacées dans le graphe qui n'ont pas été détectées comme telles dans le premier parcours.

## 2.5 Conclusion : apports et limites de la comparaison structurelle

A partir d'hypothèses d'étude, un mécanisme global de comparaison structurelle a été conçu. Si un modèle de données, écrit en langage EXPRESS, respecte ces hypothèses, une application de comparaison structurelle, adaptée à des projets exploitant ce modèle de données, est implémentable. En particulier, ce modèle de comparaison structurelle est adapté aux MNC exprimées selon le modèle IFC et IFC-Bridge.

Cependant, la généricité pure de l'approche n'est pas possible. Il convient de spécifier plusieurs informations propres au modèle : distinction des entités identifiables et non identifiables en précisant quel(s) attribut(s) sert d'identification persistante. Dans le cas du modèle IFC-Bridge, l'entité `IfcRoot` et toutes les entités qui en héritent sont identifiables grâce à l'attribut `GlobalId`. L'autre information à spécifier est la racine des graphes d'instances. Il s'agit ici de l'instance de l'entité `IfcProject` qui est unique dans n'importe quel modèle d'ouvrage.

Cependant, peu de sémantique associée aux modèles de données a été prise en compte dans cette partie. Outre les problèmes géométriques soulevés par [Pazlar06] et [Amor06], des tests ont été menés dans le cadre de nos travaux. En utilisant le logiciel ArchiCad 9©, de la société Graphisoft, les cycles de chargement/sauvegarde d'un même modèle d'ouvrage IFC provoquent des changements structuraux, notamment au niveau des entités de relation (cf. chapitre 3).

De plus, l'architecture du modèle de données IFC est telle qu'elle permet de définir une même information de plusieurs manières, afin de s'adapter à des pratiques professionnelles ou des méthodes d'acquisition de données différentes. Si le modèle de données devient accessible à un plus grand nombre d'acteurs de projets, son exploitation par les logiciels de CAO se complexifie. Un logiciel de CAO certifié est certes capable d'importer les différents modes de représentation de données. Cependant, via un mécanisme de transformation de données vers un modèle de données propriétaire, ce logiciel privilégie un mode de représentation plutôt qu'un autre. Entre plusieurs logiciels de CAO, le mode « privilégié » peut différer. Donc des cycles de chargement/sauvegarde via plusieurs logiciels provoquent aussi des changaments structuraux, sans altérer la sémantique sous-jacente du modèle d'ouvrage. Malheureusement, le système de comparaison structurelle conçu dans ce chapitre est incapable de donner des résultats corrects dans cette situation, pourtant courante dans des projets de construction (environnement hétérogène de logiciels CAO).

Comme cela a été vue au premier chapitre, prendre en compte la sémantique suppose de transformer les données vers un format pivot. Nous souhaitons trouver une autre méthode qui réutilise le système de comparaison structurelle. C'est l'objectif du troisième chapitre qui vise à enrichir les méthodes précédentes, par une analyse sémantique du modèle de données IFC-Bridge. Le but est alors de trouver comment aider la comparaison structurelle, sans en modifier sa généricité à la base.

## Résumé

Dans ce chapitre, l'objectif consiste à proposer une méthodologie générique pour comparer des modèles d'ouvrages, dérivant d'un modèle de données, exprimé en langage EXPRESS. Cette méthodologie s'accompagne d'algorithmes de comparaison de graphes d'objets fortement typés. Pour cela, une étude théorique pose cinq hypothèses nécessaires sur la structure des données à comparer : les liens entre entités sont labellisés mais ne possèdent pas d'attributs, le graphe d'objets issu du modèle d'ouvrage possède une racine, tout attribut est comparable, deux entités sont comparées seulement si elles instancient au moins une entité commune du modèle de données, chaque entité élementaire possède une identification. De plus, une entité est dite identifiable si elle possède une identification persistante, c'est-à-dire invariante durant tout le cycle de vie de l'ouvrage. Ensuite, les résultats possibles de comparaison sont spécifiés de manière générale. Suite aux hypothèses précédentes, ils correspondent à un ensemble de statuts de comparaison affectés aux entités élémentaires du modèle d'ouvrage. Les liens entre entités élémentaires servent donc au parcours et à la localisation des entités mais ne sont pas comparés. Dès lors, la distinction des entités identifiables des autres devient fondamentale. En effet, pour une entité identifiable, il s'agit d'affecter un statut de comparaison parmi 6 potentiels : *identique*, *ajouté*, *supprimé*, *changé*, *déplacé*, *changé&déplacé*. Mais lors de la comparaison d'entités non identifiables, si un attribut diffère, est-ce dû à la comparaison de deux entités distinctes avec leur propre cycle de vie, ou bien est-ce dû à un changement des propriétés d'une même entité ? Sans identification persistante, l'affectation d'un statut se révèle donc impossible. C'est pourquoi les entités non identifiables sont encapsulées par les entités parentes identifiables, grâce à un processus récursif que nous avons appelé « remontée d'attributs ». Ensuite, une analyse du langage EXPRESS permet de vérifier la validité des hypothèses dans ce formalisme, et d'établir des postulats complémentaires concernant la définition d'un modèle de données en langage EXPRESS : utilisation d'héritage simple seulement, élimination des attributs DERIVE, des clauses WHERE et des règles d'unicité. Enfin l'application du cadre théorique au modèle IFC et IFC-BRIDGE permet de compléter la vérification de l'hypothèse 2 sur l'existence d'une racine de parcours, l'instance unique de l'entité `IfcProject`. De plus, l'ensemble des entités identifiables IFC-BRIDGE est spécifié, notamment grâce à l'entité `IfcRoot`. L'algorithme de comparaison structurelle est alors complètement conçu, de la comparaison des attributs de type élémentaire au parcours simultané des deux graphes de données. Finalement, cet l'algorithme répond à la problématique d'une comparaison générique de modèles d'ouvrages, issus d'un modèle commun écrit en langage EXPRESS. La sémantique des données n'est cependant pas pris en compte, ce qui affablit très nettement la pertinence des résultats. Il convient donc de compléter cette approche par une analyse sémantique.

## Abstract

In this chapter the objective is to propose a generic methodology for product models comparison, which derive from an EXPRESS data model. This methodology comes with comparison algorithms of strongly typed objects graph. To do so, a theoretical approach sets five necessary hypotheses concerning the data structure to be compared : links between entities are labelled but they do not have any attribute, the mapped object graph from product model has a root node, every attribute can be compared, two entities are compared only if they instantiate at least one common entity from data model, every entity has an identification. Besides potential results of a comparison are specified. Because of previous hypotheses, they relate to a set of comparison statuses assigned to elementary entities from the product model. Links between elementary entities are only used for traversal and localisation of entities but they are not compared. Therefore distinction between identifiable entities and non-identifiable ones becomes fundamental. Actually each identifiable entity is assigned a status among 6 potential : *identical*, *added*, *removed*, *changed*, *moved*, *changed&moved*. But during a comparison of non identifiable entities, if an attribute is different, what does that mean ? Are there two different entities with their own lifecycle, or is this an evolution of one entity ? Without any persistent identification, status cannot be assigned. Consequently non identifiable entities are encapsulated into parent identifiable entities, thanks to a recursive process we call "“attributes flattening"”. Then checking validity of theoretical hypotheses is done by analysing EXPRESS language. Complementary postulates relate to data model definition using EXPRESS language : simple inheritance only, ignoring DERIVE attributes, WHERE clauses and uniqueness rules. Last applying theoretical approach on IFC and IFC-BRIDGE completes check of hypothesis 2 concerning root node existence, thanks to the unique instance of `IfcProject`. Besides all the identifiable entities from IFC-BRIDGE are specified, thanks to `IfcRoot` entity. The algorithm of structural comparison is then fully designed, from attributes comparison to simultaneous traversal of both product data. This algorithm eventually solves the problem of generic comparison of product models, coming from a common EXPRESS data model. However semantics is not considered here, which weakens results relevance. So this approach needs to be completed with a semantic analysis.

# Chapitre 3

# Analyses sémantiques pour la comparaison structurelle générique

## 3.1 Introduction

De l'approche basée sur des transformations de modèles [Katranuschkov06], l'analyse sémantique conduite dans ce chapitre reprend deux procédés essentiels : l'extraction d'information et les équivalents sémantiques. La première vise à simplifier le traitement des données pour une application particulière. Simplifier la structure des données permet en outre de rendre le modèle plus accessible aux acteurs de projets ou aux développeurs d'applications. Le projet SABLE [BLIS-Project05] résout cette problématique via la création de « vues » d'un même modèle dans chaque métier concerné. A ce titre, il convient de rappeler que le modèle IFC est très modulaire (cf. chapitre 1 , figure 1.8). Ceci permet de faciliter l'extraction d'informations.

La deuxième notion permet de simplifier la structure de l'information, le temps d'un traitement particulier. Cette information simplifiée serait moins accessible aux acteurs, car moins flexible (un seul mode de représentation pour une même information), mais autorise certaines opérations, comme la comparaison de données.

Dans le but de réutiliser le système de comparaison structurelle, conçu précédemment, ce chapitre vise à définir une méthodologie pour l'analyse sémantique, implémentable en tant que module additionnel du système de comparaison structurelle. Ainsi certaines fonctions de comparaison sont-elles remplacées par des mécanismes spécifiques à certaines entités, et des méthodes de prétraitement sont-elles définies.

A cette fin, plusieurs méthodes d'extraction d'information seront abordées (section 3.2), puis un cadre de définition d'équivalents sémantiques sera spécifié (section 3.3). Enfin, ces deux méthodes seront appliquées sur le modèle IFC-Bridge à travers plusieurs exemples (section 3.4).

## 3.2 Extraction sélective d'information

La notion de transformation de modèles (ou mapping en anglais) a été abordée au premier chapitre, à la section 1.3.3. Dans cette partie, l'objectif consiste à sélectionner l'information à comparer, sans devoir créer un autre véritable modèle de données, mais uniquement en modifiant légèrement le modèle de départ. La fonction de transformation est alors construite en même temps que le nouveau modèle de données. Voici un exemple illustré par la figure 3.1. Soit $A$ un modèle de données dont on souhaite sélectionner l'information. Grâce à un ensemble de règles applicables à ce modèle, un modèle de données $A'$ est obtenu, contenant seulement l'information à comparer. La fonction de transformation est conférée explicitement par l'ensemble des règles. Ce dernier représente un ensemble de d'informations fournie par l'expert ou le créateur d'applications pour un projet de construction. Ensuite, n'importe quelle instance du modèle de donnée $A$ est transformable automatiquement en une instance du modèle de données $A'$.

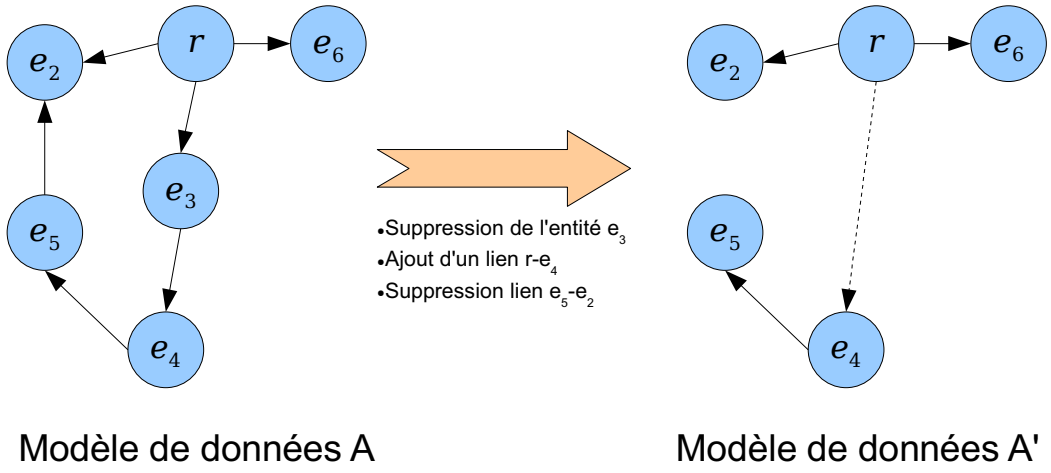


**FIGURE 3.1 :** Exemple de transformation d'un modèle de données à partir de règles.

D'après les travaux de [Broad03], l'approche est dite *active*, par opposition à une approche passive correspondant à une sélection par l'algorithme de comparaison lui-même. Une approche passive ne prendrait donc pas en compte la sémantique du modèle de données. Le but de cette section consiste alors à proposer un cadre théorique pour les différentes règles de transformation de modèles. Ces dernières ne modifient pas le déroulement global de l'algorithme de comparaison structurelle, car il suffit que ce dernier s'applique au modèle de données $A'$ au lieu du modèle $A$.

A cet effet, nous avons choisi trois méthodes : la sélection des éléments selon les entités dont ils sont l'instance (section 3.2.1), la transformation et la sélection d'attributs, afin de conserver la connexité du graphe de données et de simplifier ses liens (section 3.2.2) et la gestion des entités qui, par leur sémantique, peuvent transformer la structure du graphe de données (section 3.2.3).

### 3.2.1 Sélection par type des entités à comparer

La première méthode consiste d'abord à identifier les entités du modèle de données qui ne serviront pas à la comparaison, ou qui peuvent rendre ses résultats incorrects, comme nous le verrons plus loin. Il s'agit bien d'une analyse sémantique, car chaque entité du modèle de données porte une sémantique spécifique : entité géométrique, de matériau, d'objet de construction, de coût, etc. De plus, la sélection permet d'extraire l'information vraiment utile à un projet particulier, ou à un corps de métier spécifique.

Ensuite, il s'agit d'ignorer toutes les entités du modèle d'ouvrage qui instancient les entités précédentes, lors de la comparaison. Dans le graphe associé, cela correspond à supprimer tous les noeuds correspondant à ces instances, ainsi que toutes les arêtes concernées. Ce mécanisme consiste donc en une sélection d'entités élémentaires. S'il existe des entités du modèle de données, dépendantes entièrement d'entités qui ne sont pas sélectionnées, alors elles seront supprimées implicitement, car il n'existe plus de lien les atteignant, après traitement. Il s'agit donc d'une sélection d'entités complexes. La figure 3.2 illustre ces deux procédés de sélection d'entités élémentaires et complexes.

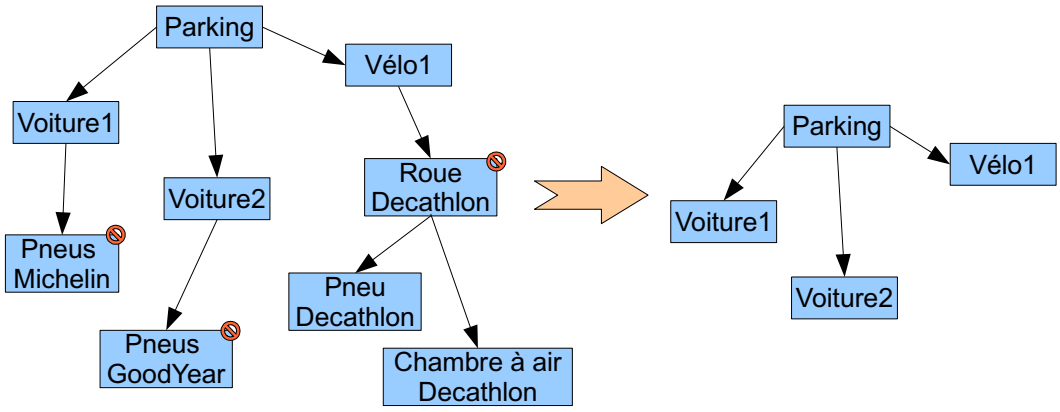


**FIGURE 3.2 :** Sélection d'entités élémentaires et complexes par type. Tous les objets de type *pneu* et de type *roue de vélo* ont été supprimés explicitement (symbole orangée sur chaque instance concernée). Or, l'entité complexe *roue de vélo* est liée à un pneu et une chambre à air, ils sont supprimés implicitement.

Cependant, lorsque le constructeur d'application ne souhaite pas effectuer des sélections d'entités complexes, le graphe obtenu risque de ne plus respecter l'hypothèse 2, c'est-à-dire que la racine présupposée du graphe d'origine ne peut plus atteindre toutes les entités du modèle d'ouvrage. Une solution consiste alors à introduire de nouvelles arêtes dans le graphe, en transformant les attributs d'entité.

### 3.2.2 Transformation d'attributs d'entités

Dans le cadre de cette thèse, transformer des attributs d'entités signifie supprimer ou ajouter des attributs à certaines entités du modèle d'ouvrage selon les entités du modèle de données

qu'elles instancient. La suppression d'attributs permet d'enlever des arêtes au graphe associé, sans enlever de noeuds. Cela permet de faciliter le parcours de l'algorithme de comparaison. La figure 3.3 propose un exemple de suppression d'attributs d'entités. En fait, cette technique représente surtout une amélioration de performance par rapport à une extraction d'information. En effet, nous rappelons que l'hypothèse 1 implique que la sémantique d'un modèle de données soit contenue surtout dans les entités (les noeuds du graphe), plutôt que dans les liens. Néanmoins le constructeur d'application a besoin de connaître sémantiquement le modèle afin de choisir les attributs à supprimer de la comparaison structurelle.

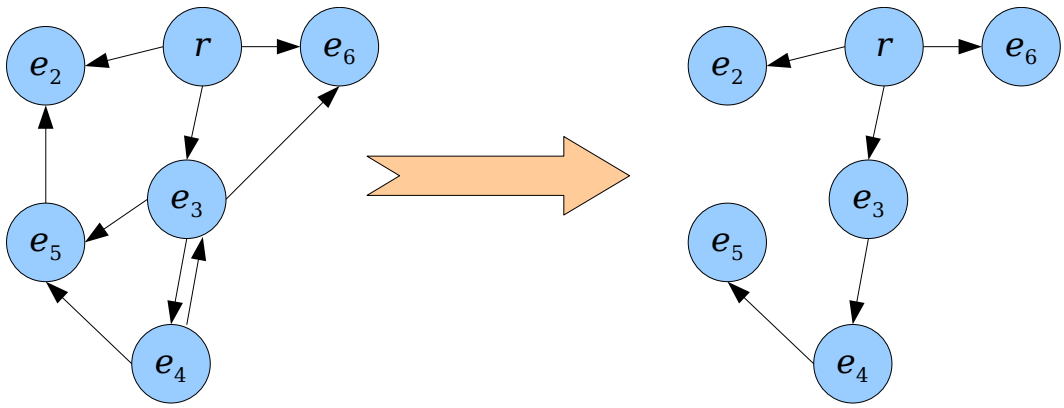


**FIGURE 3.3 :** Suppression d'attributs sur un modèle de données. Le modèle obtenu devient plus simple à parcourir après suppression d'arêtes correspondant à des attributs d'entités.

L'ajout d'attributs permet, quant à lui, d'insérer de nouveaux liens entre entités du modèle d'ouvrage. Cette technique se révèle donc particulièrement utile suite à une sélection d'entités élémentaires par type, comme expliqué à la section précédente. Le constructeur d'application peut ainsi relier *virtuellement* des entités séparées après la suppression d'une entité intermédiaire. Pour ajouter des liens, il suffit donc de créer des attributs *virtuels* pointant vers une autre entité du modèle de données. Chaque modèle d'ouvrage sera donc concerné par l'insertion de ce nouvel attribut virtuel. La figure 3.4 suggère un exemple d'une telle transformation.

Cependant, l'ajout d'attribut virtuel n'est possible que si cette transformation se révèle capable de donner automatiquement une valeur à cet attribut pour n'importe quel modèle d'ouvrage. Contrairement aux transformations précédentes où il suffisait de supprimer des entités ou des attributs clairement identifiés, il s'agit ici de spécifier un algorithme pour transformer n'importe quel modèle d'ouvrage. Dans le cadre de nos travaux de thèse, nous utiliserons l'introduction d'attributs virtuels seulement dans le cas où la suppression d'entités par type entraîne la coupure d'un chemin entre deux entités. Le paragraphe suivant explique comment donner une valeur aux attributs virtuels dans ce cas précis.

Soient trois entités $A$, $B$, et $C$. Via un attribut `aVERSb`, $A$ pointe vers $B$ et via l'attribut `bVERSc`, $B$ pointe vers $C$. Si l'entité $B$ est supprimée à cause de son type, $A$ et $C$ ne sont plus reliés. Une solution consiste donc à introduire un attribut virtuel `aVERSc` en $A$. Sa valeur est

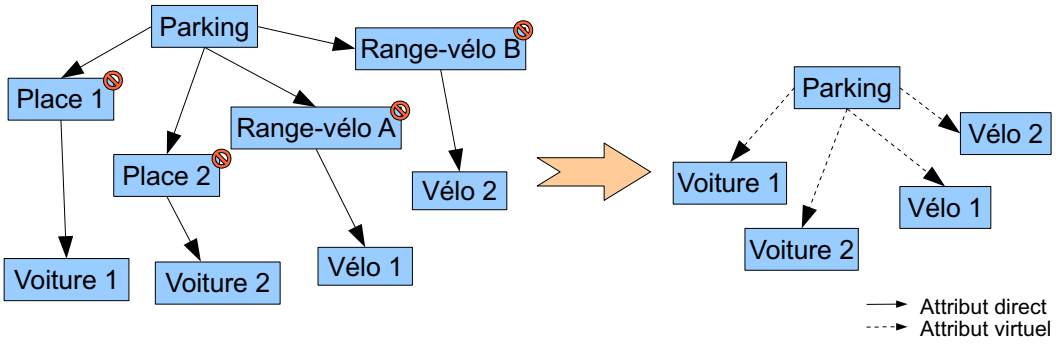


**FIGURE 3.4 :** Sélection d'entités par type et ajout d'attributs virtuels. Les entités représentant des emplacements de parking auto ou vélo sont supprimées. Pour assurer la connexité du graphe de données, les véhicules sont reliés directement à l'instance de l'entité parking.

alors `B.bVERSc`, $B$ étant connue par $A$ par le lien `aVERSb`. La démarche est similaire si $B$ reliait plus de deux entités entre elles, comme l'illustre la figure 3.5.

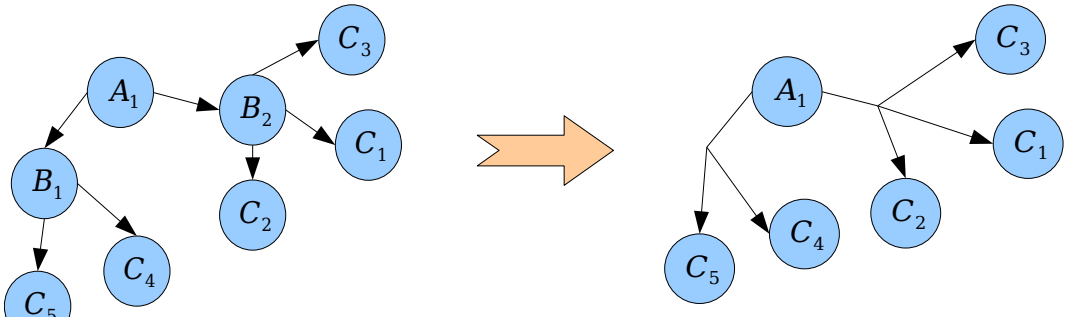


**FIGURE 3.5 :** Exemple d'introduction d'attribut virtuel. Gestion du cas où l'entité intermédiaire référence une liste d'objets.

La sélection d'entités par type et la transformation d'attributs sont deux techniques dont la finalité consiste principalement à extraire les données utiles à un projet ou à un corps de métier. Elles permettent aussi d'ignorer des données altérées par des conversions entre logiciels de CAO et qui ainsi entravent le déroulement de la comparaison structurelle. Nous verrons plus loin que ce cas se produit dans l'utilisation des IFC avec ArchiCAD 9. Nous avons choisi d'aborder une autre catégorie d'extraction d'information, potentiellement utile pour la gestion du modèle IFC-Bridge : analyser et appliquer explicitement des transformations du modèle d'ouvrage, induites par sa propre sémantique.

### 3.2.3 Gestion des transformations implicites d'un modèle d'ouvrage

Quelque soit le domaine industriel, la modélisation de l'information s'accompagne de solutions de stockage de cette information tout au long du projet. En effet, les projeteurs ont besoin de constituer une mémoire sur le projet, revêtant la forme d'armoire à plans, électronique ou non. Dès la conception, ils souhaitent donc connaître les différentes modifications effectuées sur le projet, par quel auteur, etc. A cette fin, certains standards d'échanges définissent des modèles de données munis de mécanismes traçant l'évolution du modèle d'ouvrage. Un tel modèle d'ouvrage contient par conséquent des informations sur ses propres changements (changements de propriétés, ajouts et suppressions d'entités). Nous verrons ainsi à la section 3.4.2 que les IFC implémentent partiellement ce mécanisme.

Par rapport à la comparaison structurelle, il s'agit d'identifier les informations permettant l'ajout, la suppression, le changement et le déplacement dans le graphe de données d'une entité. Dans le cadre théorique de nos travaux, nous supposons qu'une telle information fait partie de l'état de l'entité concernée, c'est-à-dire un attribut de type simple ou référençant une entité non identifiable. Voici un exemple simple :

```
ENTITY Object;
  id : STRING;
  name : STRING;
  subObjects : LIST [0:?] OF Object;
  deleted : BOOLEAN;
  moved : BOOLEAN;
  movedTo : OPTIONAL Object;
END_ENTITY;
```

Un objet est défini par une identification (supposée persistante), un nom, et un ensemble de sous-objets. Cet objet possède de plus un attribut booléen `deleted` précisant si l'entité considérée a été en fait supprimée. L'attribut `moved`, quant à lui, montre si cette entité a été déplacée, et, le cas échéant, son nouveau parent est indiqué par l'attribut `movedTo`. Pour la comparaison structurelle, il s'agit de prendre en compte ces derniers attributs, en enlevant ou en déplaçant l'entité concernée, avant d'effectuer la comparaison. Cela peut se faire grâce à un prétraitement de la structure des données, qui s'effectue en temps constant pour chaque entité, grâce à l'hypothèse d'étude précédemment énoncée.

La sélection et le réarrangement de l'information à comparer facilitent le travail à mener par la comparaison structurelle. Cependant, cette analyse sémantique a besoin d'être complétée, car il peut exister plusieurs entités permettant de définir des concepts équivalents (exemple : un point 3D en coordonnées cartésiennes, cylindriques et sphériques). C'est pourquoi l'intégration d'un mécanisme d'équivalence sémantique se révèle nécessaire pour enrichir le moteur de comparaison structurelle.

## 3.3 Définition des équivalences sémantiques

### 3.3.1 Equivalence entre entités élémentaires

D'après l'hypothèse 2 sur l'existence d'une racine du graphe de données, nous pouvons en déduire le corollaire suivant :

> *Soit $r$ la racine du graphe de données. Quelle que soit l'entité $e$ du modèle d'ouvrage différente de $r$, il existe une entité $e_i$ pointant vers $e$ via un attribut.*

En effet, cet attribut représente la sémantique de l'arête orientée pointant vers l'entité $e$, l'existence de cette arête étant une conséquence directe de l'hypothèse 2.

Pour étudier l'équivalence entre entités élémentaires, nous avons donc choisi une approche orientée *attribut d'entité* : deux entités sont dites équivalentes ou non *relativement à un attribut d'entité*. D'après les hypothèses 3 et 4, l'équivalence entre deux entités suppose donc qu'elles soient comparables, c'est-à-dire, qu'elles instancient au moins une entité en commun. En effet, un attribut d'entité est typé, et si ce type est une entité du modèle de données, alors sa valeur doit être une entité du modèle d'ouvrage instanciant le type précédent. Par exemple, voici un modèle de données EXPRESS :

```
ENTITY Object;
  name : STRING;
  position : Point;
END_ENTITY;

ENTITY Point
ABSTRACT SUPERTYPE OF (ONEOF(CartesianPoint, SphericPoint));
END_ENTITY;

ENTITY CartesianPoint
SUBTYPE OF (Point);
  x : REAL;
  y : REAL;
  z : REAL;
END_ENTITY;

ENTITY SphericPoint
SUBTYPE OF (Point);
  r : REAL;
  theta : REAL;
  phi : REAL;
END_ENTITY;
```

L'entité `Object` possède un attribut `position`, de type `Point`. Or l'entité `Point` est une entité vide, mais les entités `CartesianPoint` et `SphericPoint` héritent de `Point`. Vis-à-vis de l'attribut `position` de l'entité `Object`, deux instances des entités respectives `CartesianPoint` et `SphericPoint` sont comparables et donc potentiellement équivalentes. Cependant, si l'attribut était de type `CartesianPoint`, alors les deux instances précédentes n'auraient pas pu être com-

parées car celle de `SphericPoint` n'aurait pas pu être une valeur pour l'attribut `position`.

A partir du pré-requis précédent, il devient à présent possible de donner une définition de l'équivalence sémantique d'entités élémentaires :

**Définition 14.** *Soit $e_i$ et $e_j$ deux entités du modèle d'ouvrage, et `attr` un attribut de type $T$ de l'entité $E$ du modèle de données. $e_i$ et $e_j$ sont dites équivalentes par rapport à l'attribut `attr` si et seulement si :*

- *$e_i$ et $e_j$ instancient tous les deux $T$.*
- *Il existe une relation d'équivalence $\equiv_T$ dans l'ensemble des entités qui instancient $T$.*
- *$e_i \equiv_T e_j$.*

Nous remarquons que cette définition fait surtout intervenir le type $T$ plutôt que l'attribut en tant que tel. C'est pourquoi il est facile d'enrichir le moteur de comparaison structurelle sans modifier en profondeur son fonctionnement : le constructeur d'application doit définir les relations d'équivalence pour établir des équivalences sémantiques. En référence au modèle de données précédent, cela revient à définir une relation d'équivalence par rapport à l'entité `Point`. Pour cela, une solution consisterait à convertir les deux points en coordonnées cartésiennes (si le point est une instance de CartesianPoint, le travail est déjà fait) et à comparer les valeurs obtenues. Contrairement aux solutions existantes basées sur des transformations de modèles [Katranuschkov06], la définition de relations d'équivalences permet de ne pas avoir à effectuer un mapping sur la totalité du modèle. Le moteur de comparaison utilise alors les relations d'équivalences définies par le constructeur d'applications, au lieu de la comparaison des attributs deux à deux si l'entité possède un type géré par ces relations.

L'équivalence sémantique peut aussi porter sur des types simples au lieu d'entités. il suffit alors de remplacer les relations d'équivalence usuelles par la relation choisie. Par exemple, la comparaison de chaînes de caractères sans prendre en compte la casse : 'viaduc', 'Viaduc' et 'VIADUC' seraient donc équivalents. Autre exemple, la comparaison de valeurs réels jusqu'à $n$ décimales : 1.239, 1.234 et 1.231 sont équivalents si $n = 2$.

Lors de notre réflexion autour des équivalents sémantiques, nous avons remarqué que ce procédé se révèle incompatible avec les méthodes de hachage. En effet, lorsque les attributs d'une entité non identifiable sont remontés à l'entité identifiable parente, le hachage permet de remonter toute l'entité en transformant tous les attributs à remonter en un seul nombre entier. Or, deux hachages sont égaux seulement si l'ensemble des attributs sont deux à deux égaux (l'unicité du hachage est assuré de manière probabiliste). Si l'on introduit des équivalences sémantiques, il se peut que des attributs soient différents deux à deux. Donc le hachage sera différent malgré l'équivalence sémantique. C'est pourquoi, au prochain chapitre, nous verrons comment conserver des méthodes de hachage avec des équivalents sémantiques.

Spécifier et traiter les équivalences sémantiques permet d'enrichir la comparaison structurelle, c'est-à-dire de l'adapter à des modèles de données spécifiques pour lesquels il existe plusieurs manières de définir une même information. Cependant, dans certaines situations, en

particulier dans le domaine de la construction, se trouvent des équivalences qui ne possèdent pas toutes les propriétés requises pour une relation d'équivalence (au sens des mathématiques). C'est le cas des relations d'égalité avec tolérance.

### 3.3.2 Tolérance et correction du modèle d'ouvrage

La notion de tolérance permet de supposer équivalentes deux valeurs, en principe numériques, qui ne sont pas égales. Par exemple, avec une tolérance de $0.01$, on a $2.569 \approx 2.574$ car $2.574 - 2.569 = 0.005$. Il s'agit d'une relation réflexive : $a \approx a$, symétrique : $a \approx b \Rightarrow b \approx a$. Elle n'est cependant pas transitive : $2.569 \approx 2.574$ et $2.574 \approx 2.581$ mais $2.569 \neq 2.581$.

La notion de tolérance étant utilisée de manière intensive dans le monde de la construction, il convient d'adapter le travail de la partie précédente à des relations non transitives. La principale conséquence de l'utilisation d'une telle relation, pour la comparaison structurelle, est de rendre cette dernière incohérente en fonction des données entrées. Lors de l'énoncé de l'hypothèse 3, à la section 2.2.1 du deuxième chapitre, ce problème a été en effet soulevé : les résultats de comparaison peuvent varier en fonction des différents cheminements de conception, pour un même ouvrage. Reprenons l'exemple précédent : $2.569 \approx 2.574$ et $2.574 \approx 2.581$ mais $2.569 \neq 2.581$. Cela signifie que l'algorithme n'aurait pas détecté de changement ni entre la première et la deuxième version d'un modèle d'ouvrage, ni entre la deuxième et la troisième version de ce modèle d'ouvrage. Il aurait cependant détecté un changement si la première et la troisième version avaient été comparées directement. Il y a donc une incohérence.

Une solution consiste à effectuer une comparaison qui modifie le modèle d'ouvrage. En effet, d'après [Pazlar06] et [Amor06], un modèle d'ouvrage subit des altérations lors de conversions et d'éditions successives. Ces altérations restent minimes : par exemple, les altérations des valeurs numériques restent inférieures aux tolérances usuelles : $1.25$ peut devenir $1.249998$. La gestion de la tolérance par correction du modèle permet ainsi de rectifier ce bruit qui pourrait altérer de manière visible le modèle d'ouvrage. Non seulement la comparaison structurelle détecte qu'il s'agit en réalité de la même valeur (tolérance), mais elle élimine aussi le bruit généré par la conversion (correction du modèle d'ouvrage).

A cette fin, nous supposons que, lors de la comparaison de deux instances (originale et modifiée), l'instance originale possède la valeur correcte, et l'instance modifiée possède la valeur susceptible d'être bruitée. C'est pourquoi, la correction du modèle d'ouvrage s'effectue sur l'instance modifiée. Ce procédé permet de rétablir la cohérence de la comparaison structurelle, comme l'illustre la figure 3.6.

On a ainsi $2.569 \approx 2.574$. Sur l'instance modifiée, $2.574$ est remplacée par $2.569$. A la deuxième comparaison, on a alors $2.569 \neq 2.581$ au lieu de $2.574 \approx 2.581$. L'utilisation de la tolérance suppose donc qu'aucun changement de conception ne modifie de valeurs dans l'intervalle fixé par la tolérance. En effet, cette modification serait prise pour une altération de données et donc corrigée à l'ancienne valeur. En utilisant ce procédé, l'utilisation de relations de tolérance ne se différencie pas des relations d'équivalence, par conséquent les conclusions de la

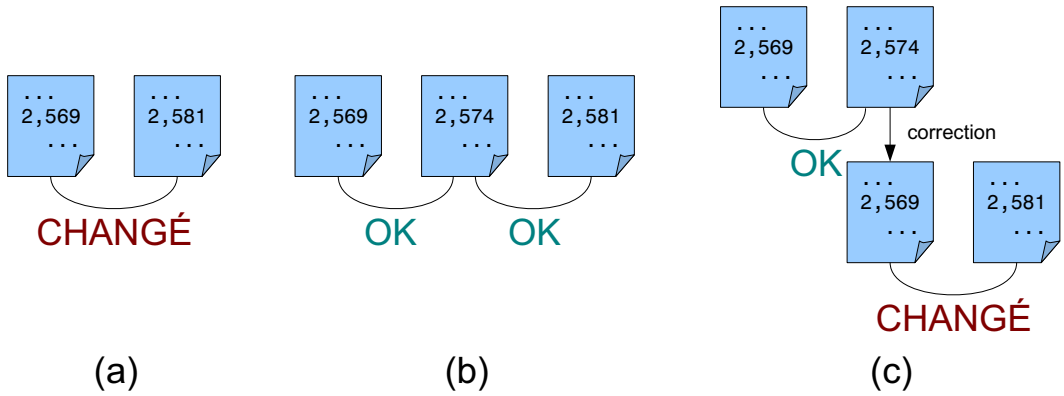


**FIGURE 3.6 :** Comparaison de valeurs avec tolérance. En (a), les deux valeurs sont différentes, même avec une tolérance de 0.01. En (b), via un modèle d'ouvrage intermédiaire, le système ne détecte pas de changement entre le modèle initial et le modèle final. Il y a donc incohérence avec le scénario (a). En (c), lorsque le système ne détecte pas de changement selon la tolérance, la valeur est corrigée à sa valeur initiale. La deuxième comparaison aboutit à un changement. Ce résultat est cohérent avec le scénario (a).

section précédente s'appliquent aussi maintenant à ces relations non transitives.

Cependant, la correction de l'instance modifiée en cours de comparaison influe sur le déroulement de cette dernière. En effet, si une entité de l'instance modifiée est comparée avec plusieurs entités de l'instance originale qui ont chacune des valeurs d'attributs différentes, alors comment corriger l'entité de l'instance modifiée ? En particulier, si ces différences sont inférieures à l'intervalle de tolérance, comme l'illustre la figure 3.7. Nous avons choisi dans ce cas de ne pas corriger la valeur de l'entité de l'instance modifiée, et de détecter à chaque fois un changement lors de la comparaison. La tolérance n'est pas pris en compte dans ce cas.

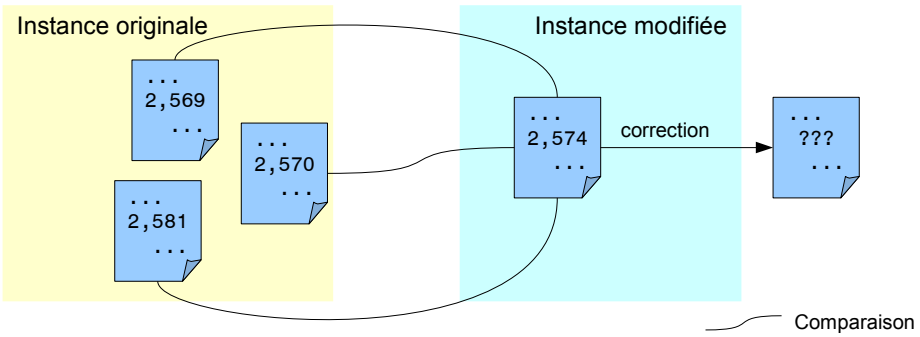


**FIGURE 3.7 :** Comparaison d'une entité de l'instance modifiée avec plusieurs entités de l'instance originale. L'application d'une relation de tolérance dans ce cas pose problème.

Cette partie clôt l'approche théorique de l'analyse sémantique. Nous supposons en effet que, grâce à des méthodes d'extraction d'information et à la définition d'équivalents sémantiques, la méthode de comparaison structurelle, enrichie de cette analyse, répond aux besoins des

constructeurs d'application et des projeteurs pour suivre un modèle d'ouvrage. Cette analyse évite ainsi le recours à des transformations de modèles. Pour vérifier la portée de ces méthodes d'analyse sémantique, il convient de les appliquer au cadre du modèle de données IFC-Bridge.

## 3.4 Application au sein du modèle IFC-Bridge

### 3.4.1 Traitement des entités de relation

Lors de premiers tests d'édition et de conversion de modèles IFC sur ArchiCad 9© de Graphisoft, en utilisant le plugin IFC 2x3, des modifications sur la structure du modèle ont lieu, même sans vraiment l'éditer entre deux sauvegardes. Nous retiendrons en particulier le changement des entités de relation. Pour cela, un bref rappel des entités IFC de relation est nécessaire.

Au deuxième chapitre, à la section 2.4.1, les entités de structure spatiale ont été abordées, ainsi que les liens entre elles (cf. figure 2.7 et 2.8). Plus généralement, le modèle IFC définit des entités de relation, héritant de l'élément de base `IfcRelationship` (cf. figure 3.8).

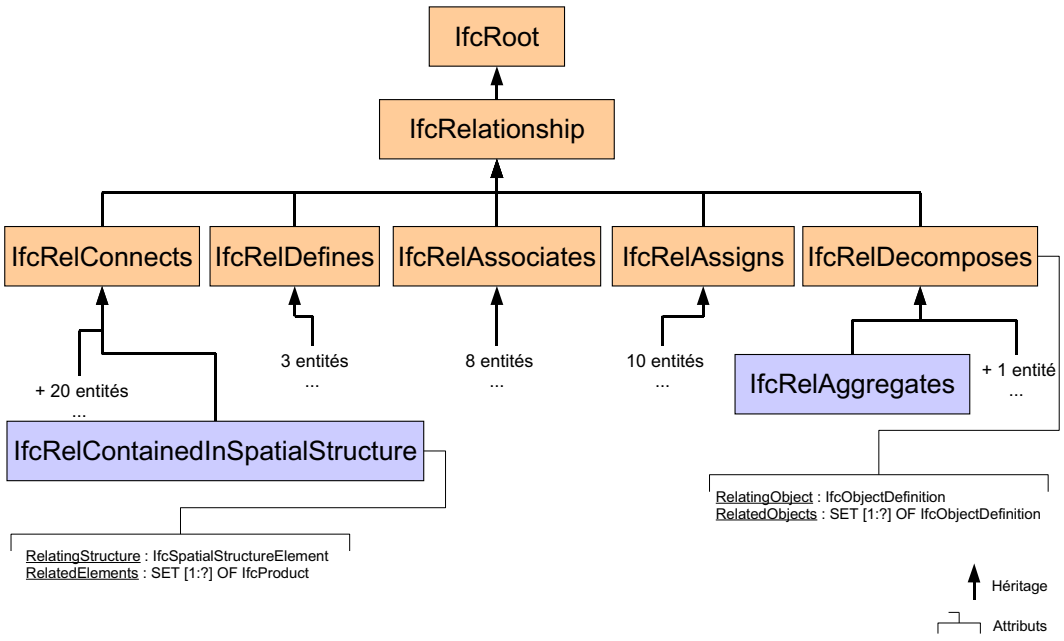


**FIGURE 3.8 :** Diagramme d'héritage partiel des entités de relations IFC.

A cause de son héritage de l'entité `IfcRoot`, il s'agit d'une entité identifiable. Elle permet de simuler les liens munis d'attributs, grâce à une entité intermédiaire : l'entité de relation. En effet, les liens directs entre entités sont labellisés (label de l'attribut) mais ne possèdent pas d'attributs d'après l'hypothèse 1 du deuxième chapitre. Ces relations sont dites de type 1-à-1 ou bien 1-à-plusieurs, selon le cas. Par exemple, l'entité `IfcRelContainedInSpatialStructure` permet de relier un élément de structure spatiale (via l'attribut `RelatingStructure`) et plusieurs éléments de construction (via RelatedElements). L'entité `IfcRelAggreagates` relie un élément `conteneur` (via `RelatingObject`) à plusieurs éléments *contenus* (via `RelatedObjects`).

Après ouverture et sauvegarde sous un autre nom d'un modèle d'ouvrage IFC, avec ArchiCad 9©, la comparaison entre le modèle original et la sauvegarde montre que toutes les entités de relation ont changé d'identifiant, comme l'illustre la figure 3.9.

```
#4052= IFCRELAGGREGATES('1KfxnqpT94GgqzsdI2iUED',#13,'BuildingStoreyContainer',...
#4848= IFCRELAGGREGATES('1Ro$ivThTF1RRh2DhU_tHx',#13,'BuildingContainer',...
#4852= IFCRELAGGREGATES('2GEArdJxv3agospNFwoDPO',#13,'SiteContainer',...
#4854= IFCRELAGGREGATES('1SkUzwvzj45QEX1VSQvWPq',#13,'ProjectContainer',...
```

**Instance originale**

```
#4352= IFCRELAGGREGATES('0igC_T1BH3Ph5xprF8w4W9',#13,'BuildingStoreyContainer',...
#5797= IFCRELAGGREGATES('1b6O77L1v3FBsM6sy8MSsl',#13,'BuildingContainer',...
#5801= IFCRELAGGREGATES('0IMu3UNuXAV8MHsxagEY2o',#13,'SiteContainer',...
#5803= IFCRELAGGREGATES('22O8KZHvzBdQt85ofgUN3p',#13,'ProjectContainer',...
```

**Instance modifiée**

**FIGURE 3.9 :** Comparaison des entités `IfcRelAggregates` durant un simple chargement/sauvegarde dans ArchiCad 9©. Les identifications de la sérialisation ont changé mais cela n'est pas pis en compte par l'algorithme de comparaison. Cependant, les identifications persistantes (premier paramètre de chaque entité) ont aussi changé alors qu'aucune modification sur le modèle n'a été effectuée.

Or, comme il a été vu à la section 2.4.1 du deuxième chapitre, les entités de relation sont nécessaires au parcours de l'entité `IfcProject` aux entités de structure spatiale, et de ces dernières aux éléments de construction. Un changement d'identifiants perturbe le parcours de la comparaison structurelle : les entités de relation se voient attribuer le couple de statuts 'supprimé' / 'ajouté', et les entités liées à ces dernières seront toutes détectées comme 'déplacé'.

Une première solution consiste à ignorer l'identifiant des entités de relation durant la comparaison. Cela signifie qu'elles ne seraient plus identifiables. Or, les entités non identifiables ne peuvent pas servir à parcourir le modèle d'ouvrage entre entités identifiables (cf. section 2.2.3 du chapitre 2). Une autre solution consiste donc à ignorer toutes les entités de relation et à les remplacer par des attributs virtuels, conformément au procédé décrit dans la section 3.2. Dans cette partie, nous détaillons cette transformation pour les entités `IfcRelAggregates` et `IfcRelContainedInSpatialStructure`. Concernant les autres entités de relation, le chapitre 4 donnera plus de détails. La figure 3.10 détaille le procédé pour les deux entités précédentes.

`IfcRelAggregates` permet de relier deux éléments héritant de `IfcObjectDefition`. Dans cet exemple, il s'agit d'un bâtiment (instance de `IfcBuilding`) et d'étages (instances de `IfcBuildingStorey`). L'entité de relation possède deux attributs `RelatingObject` et `RelatedObjects`. Ces derniers sont respectivement associés aux attributs inverses `IsDecomposedBy` et `Decomposes` de l'entité `IfcObjectDefinition`. La multiplicité des liens entraîne qu'un seul objet conteneur – lié à `RelatingObject` – est relié à un ou plusieurs objets contenus (attribut `RelatedObjects`). La manipulation se décompose selon les étapes suivantes :

- Supprimer l'attribut inverse `IsDecomposedBy` liant le bâtiment aux entités de relation.

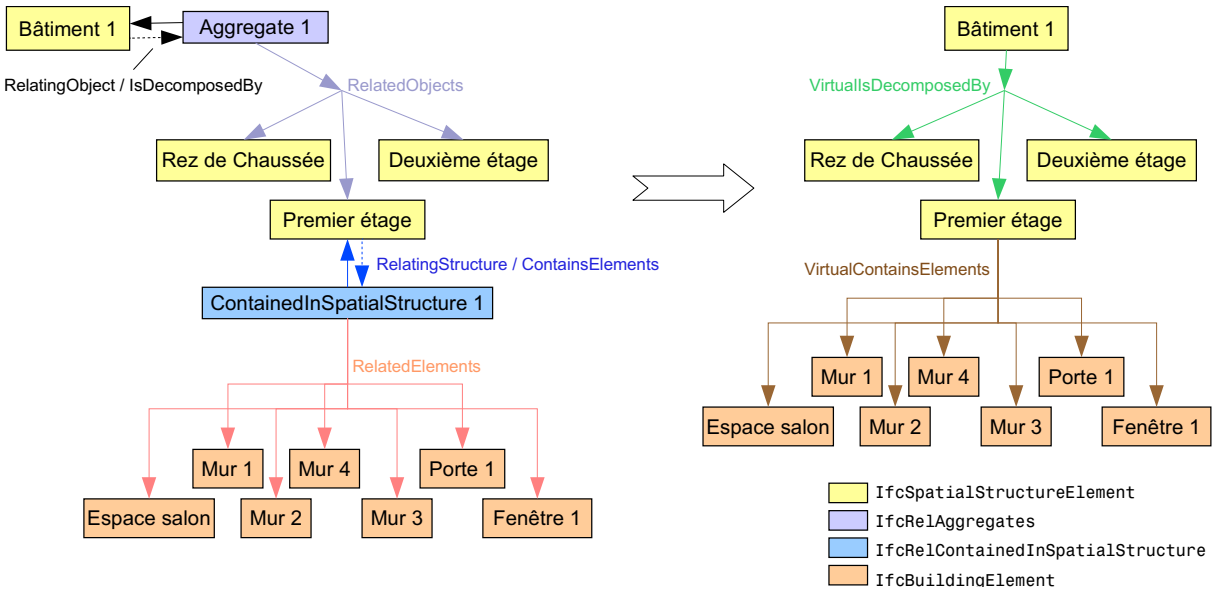


**FIGURE 3.10 :** Application des attributs virtuels aux entités de relation IFC.

- Supprimer chaque attribut direct `RelatedObjects` liant l'entité de relation à un ensemble d'étages.
- Ajouter un attribut virtuel `VirtualIsDecomposedBy` qui constitue un ensemble référençant directement chaque groupe d'étages. La valeur de cet attribut est remplie grâce aux valeurs des attributs supprimés précédemment.
- Supprimer l'attribut inverse `Decomposes` liant chaque étage à son entité de relation.
- Supprimer l'attribut direct `RelatingObject` liant l'entité de relation au bâtiment
- Ajouter un attribut virtuel `VirtualDecomposes` référençant son entité conteneur, c'est-à-dire le bâtiment. La valeur de cet attribut est déduite des deux attributs précédents.

La démarche est analogue pour l'entité `IfcRelContainedInSpatialStructure`, illustrée par la figure 3.10. La principale conséquence d'une telle transformation c'est d'ignorer les attributs décrivant les entités de relation. Dans le cadre de ces travaux, nous n'avons pas besoin de ces informations pour comparer des modèles d'ouvrage. Néanmoins, dans l'exemple précédent, il reste toujours possible de créer des attributs virtuels au niveau du site pour récupérer les attributs des entités `IfcRelAggregates`. Dans tous les cas, il s'agit d'un exemple pour montrer l'utilité de l'extraction de l'information et particulier de la suppression d'entités par type, compensée par l'introduction d'attributs virtuels. Il est en effet fort probable que le changement d'identification des entités de relation soit une spécificité du logiciel ArchiCad© ou seulement de la version testée (version 9).

Une autre application à Ifc-Bridge de l'extraction d'information correspond aux transformations implicites du modèle d'ouvrage : l'entité `IfcOwnerHistory`.

### 3.4.2 Gestion de l'entité IfcOwnerHistory

Toutes les entités identifiables héritent de `IfcRoot`. Cette dernière possède un attribut `OwnerHistory` de type `IfcOwnerHistory`. La figure 3.11 rappelle la structure de cette entité. Elle permet de stocker des méta-informations sur l'entité considérée, c'est-à-dire la nature de la dernière modification, l'auteur de ce changement, la protection de cette entité contre les modifications, etc. Vis-à-vis de la comparaison structurelle, il est nécessaire de prendre en compte ces informations.

```
ENTITY IfcOwnerHistory;
 OwningUser : IfcPersonAndOrganization;
 OwningApplication : IfcApplication;
 State : OPTIONAL IfcStateEnum;
 ChangeAction : IfcChangeActionEnum;
 LastModifiedDate : OPTIONAL IfcTimeStamp;
 LastModifyingUser : OPTIONAL IfcPersonAndOrganization;
 LastModifyingApplication : OPTIONAL IfcApplication;
 CreationDate : IfcTimeStamp;
END_ENTITY;

TYPE IfcChangeActionEnum = ENUMERATION OF
( NOCHANGE,
MODIFIED,
ADDED,
DELETED,
MODIFIEDADDED,
MODIFIEDDELETED);
END_TYPE;
```

**FIGURE 3.11 :** Description EXPRESS de l'entité `IfcOwnerHistory`

Cette information permettrait par exemple de confirmer le résultat de la comparaison structurelle : une entité a été détectée 'changée' par l'algorithme, et ceci est confirmé par l'attribut `ChangeAction` de `IfcOwnerHistory`. Toutefois cette vérification ne se révèle pas nécessaire, et ce d'autant qu'un logiciel d'édition de modèles pourrait « oublier » de mettre à jour cet attribut. Lors de nos premiers tests avec ArchiCad 9©, nous avons cependant noté un autre fait fondamental : une entité identifiable, supprimée lors de l'édition du modèle d'ouvrage par ArchiCad, n'est pas vraiment enlevée du modèle lui-même. Seul son attribut `ChangeAction` s'est vu attribuer la valeur 'DELETED'. Une comparaison structurelle de base aurait détecté l'entité concernée comme 'changée', car elle existe toujours dans l'instance modifiée, avec l'attribut `OwnerHistory.ChangeAction` modifié.

Une solution consiste donc à ignorer toute entité identifiable dont l'attribut `OwnerHistory.ChangeAction` vaut 'DELETED'. Cela signifie que si une entité de l'instance modifiée possède un tel attribut, contrairement à son homologue de l'instance originale, le fait d'ignorer l'entité marquée supprimée implique l'attribution du statut 'supprimée' à l'entité de l'instance originale,

ce qui est sémantiquement correct. Il s'agit donc ici d'appliquer la gestion des transformations implicites, abordée à la section 3.2.3. Pour cela, le prétraitement des données consiste à vérifier l'attribut `OwnerHistory.ChangeAction`. Si la valeur est 'DELETED', alors il convient de supprimer temporairement l'entité du graphe de données, pendant la comparaison, ainsi que tous les liens pointant vers elle.

Plus généralement, il serait possible d'extraire d'autres informations à partir du modèle IFC-Bridge, en utilisant les techniques abordées : la suppression d'entités par type, l'ajout d'attributs virtuels et la gestion des transformations implicites. Par exemple, rappelons que l'architecture des IFC est très modulaire (cf. figure 1.8 du premier chapitre), et qu'il serait possible d'extraire uniquement les modules utiles à une équipe projet, grâce à une suppression d'entités par types. Après avoir appliqué les techniques d'extraction d'information sur le modèle IFC-Bridge, notre analyse a aussi conduit à établir des équivalences sémantiques, en particulier sur la définition géométrique des pièces et leur localisation spatiale.

### 3.4.3 Equivalences sémantiques des entités géométriques

Parmi les différents modules qui forment le modèle de données IFC et IFC-Bridge, certains permettent de définir des formes géométriques. Il s'agit de *Geometric Model Resource* et de *Geometric Resource*. Chaque élément de construction IFC possède une ou plusieurs représentations géométriques, grâce à l'attribut `Representation`. Conformément aux définitions de [IAI06] concernant l'entité `IfcShapeRepresentation` (qui hérite de l'entité `IfcRepresentation`), il existe plusieurs manières de définir une géométrie :

- Des ensembles géométriques basés sur des points, des lignes des courbes, et des surfaces : **Curve2D**, **GeometricSet**, **GeometricCurveSet**.
- Un modèle entièrement surfacique : **SurfaceModel**.
- Un modèle volumique : **SolidModel**. Ce modèle possède lui-même plusieurs variantes :
  - **SweptSolid** : extrusions et surfaces de révolutions.
  - **Brep** : utilisations de surfaces frontières.
  - **CSG** et **Clipping** : utilisations d'opérations booléennes entre volumes.
- Une boite englobante pour une géométrie simplifiée : **BoundingBox**.
- Une courbe de référence munie de sections : **SectionedSpine**. Cette notion a été étendue dans le cadre du modèle IFC-Bridge étant donné son utilité pour la définition de pièces prismatiques (tablier, piles, etc.) en fonction d'un axe de référence (axe rouge par exemple).

Parmi ces différentes représentations, plusieurs peuvent donner le même résultat visuel et géométrique. Dans le cadre de nos travaux, nous avons donc cherché à appliquer la définition d'équivalents sémantiques sur une partie des entités géométriques du modèle IFC-Bridge : les courbes.

La figure 3.12 présente le diagramme d'héritage de l'entité `IfcCurve`. Dans le paradigme IFC, une courbe peut être :

- Une courbe finie : des B-Splines, en particulier les courbes de Bézier, des courbes compo-

sées, des polylignes, et des courbes infinies tronquées.
- Une conique : cercle ou ellipse.
- Une droite.
- Une clothoïde. Cette courbe, issue de l'extension Ifc-Bridge, permet de définir une transition entre une droite et un cercle.
- Une courbe parallèle d'une distance fixe par rapport à une autre.

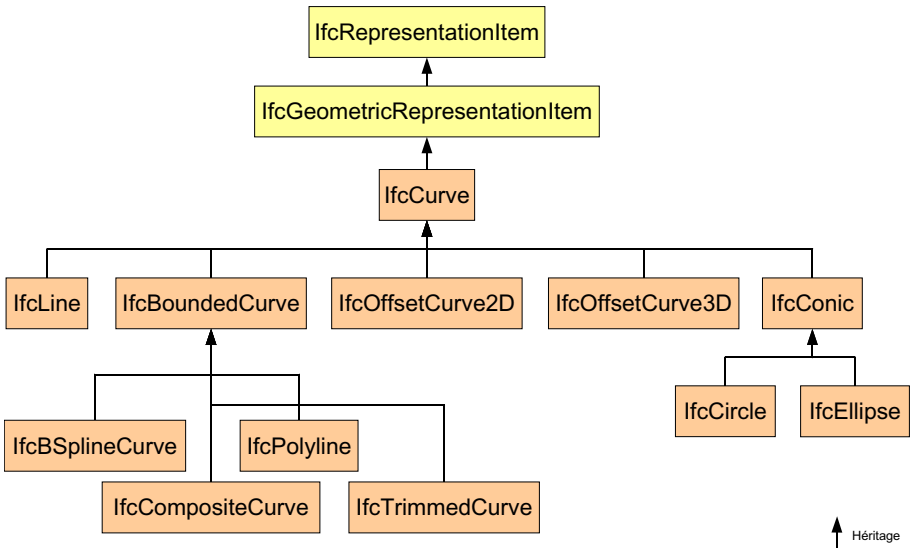


**FIGURE 3.12 :** Diagramme d'héritage de l'entité `IfcCurve`. Les entités en orange correspondent aux entités

Parmi ces représentations, certaines peuvent être équivalentes et d'autres non. Par exemple, une ligne ou une polyligne n'est jamais comparable à une ellipse[1]. De plus, il existe des cas extrêmes où certains modes de représentations deviennent équivalents. Par exemple, une polyligne avec un seul segment est comparable à une courbe tronquée dont la base est une droite. C'est pourquoi notre analyse s'effectue en deux temps.

D'abord, certains modes de représentations sont prétraités, pour simplifier certaines structures composées comme les polylignes et les courbes composées. Dans certains cas, ces structures sont remplacées par d'autres modes de représentations. Les quatre modes concernés sont les suivants :

1. **Les courbes infinies tronquées** – `IfcTrimmedCurve`. Cette entité possède un paramètre `BasisCurve`, de type `IfcCurve`. Il existe une contrainte qui interdit l'utilisation de courbes finies comme courbe de base. Seules les coniques et les droites sont donc troncables. Les deux lieux de troncature (`Trim1` et `Trim2`) sont exprimés soit par deux points 3D, soit par deux abscisses curvilignes. Or, il existe un cas limite où la courbe tronquée est équivalente à sa courbe de base : lorsqu'il s'agit d'une troncature d'une courbe dont la fonction paramétrique est périodique. Dans notre cas, cela correspond aux courbes fermées, les coniques (cercles ou ellipses). Dans le cas où la courbe de base est une ellipse ou un cercle,

[1] Une ellipse discrétisée est néanmoins associable à une polyligne, mais cela dépasse le cadre de ces travaux

le prétraitement consiste donc à remplacer la courbe tronquée par sa courbe de base si les abscisses curvilignes correspondant à Trim1 et Trim2 ont une distance réciproque égale à la période de la courbe, ici 360°. En effet, la troncature n'a pas d'utilité car la courbe entière est dans ce cas décrite.

2. **Les polylignes** – `IfcPolyline`. Une telle entité possède un attribut `Points` qui représente une liste de points 3D correspondant aux sommets de la polyligne. Le prétraitement consiste d'abord à éliminer des sommets inutiles, c'est-à-dire des sommets qui forment un angle plat de 180°(alignement de ce point avec le point précédent et le point suivant). Ensuite, si le résultat donne une polyligne avec un seul segment, celle-ci est équivalente à une entité `TrimmedCurve` dont la base est une droite. Cependant, lors de la suppression d'un sommet, il convient de prendre en compte le changement d'abscisse curviligne.

3. **Les courbes composites** – `IfcCompositeCurve`. Cette entité possède un attribut `Segments` représentant l'ensemble des courbes composantes avec des conventions de transition. Ces courbes composantes appartiennent à la famille des courbes finies. Le prétraitement est une généralisation de celles du point précédent : il consiste à fusionner des courbes composantes à condition qu'elles soient de même type et que leur paramètre permette une fusion pour former une autre courbe. Par exemple, deux quarts de cercle de même centre et dont les limites permettent de former un seul demi-cercle. Si ce prétraitement est aisé dans le cas de courbes tronquées de type droites ou coniques, la fusion de B-Splines exige plus de rigueur : compatibilité des points de contrôle, ainsi que les poids associés dans le cas de courbes de Bézier rationnelles. Ensuite, si le résultat donne un seul segment, la courbe composite est remplacée par son seul segment associé.

4. **Les B-Splines** – `IfcBSplineCurve`. Ces types de courbes sont très utilisés en CAO. Il existe certains cas triviaux, où ces courbes sont comparables à des segments. Par exemple, dans le cas d'une simple courbe de Bézier, lorsque les quatre points de contrôle sont alignés. D'autres B-Splines peuvent être aussi équivalentes à des arcs de cercles [Bangert97]. Dans le cadre de ces travaux, seule l'équivalence avec des segments sera étudiée, consistant en un prétraitement détectant l'alignement des quatre points de contrôle.

Ce prétraitement permet de gérer des cas triviaux d'équivalences et simplifie l'analyse. Ainsi, une courbe composite n'est à présent comparable qu'avec une autre courbe de même type ou de type polyligne (si toutes les courbes segments sont des droites tronquées). Le tableau 3.1 montre la comparabilité des courbes selon leur type.

Dans un deuxième temps, il reste peu de types de courbes comparables à d'autres, grâce au prétraitement :

- Composite-Polyline : si chaque segment de la courbe composite est un segment, alors la comparaison des sommets de ces segments permet d'établir l'équivalence avec une polyligne.
- Circle-Ellipse : si les deux demi-rayons d'une ellipse sont égaux entre eux et au rayon du cercle à comparer, ainsi que leur centre, les deux courbes sont équivalentes.
- Circle-Offset : une courbe parallèle (de distance algébrique $d$) à un cercle de rayon $r$ est elle-même un cercle de rayon $r + d$.

| | B-Spline | Composite | Polyline | Trimmed | Circle | Ellipse | Line | Clothoid | Offset |
|---|---|---|---|---|---|---|---|---|---|
| **B-Spline** | √ | | | | | | | | |
| **Composite** | | √ | √ | | | | | | |
| **Polyline** | | | √ | | | | | | |
| **Trimmed** | | | | √ | | | | | |
| **Circle** | | | | | √ | √ | | | √ |
| **Ellipse** | | | | | | √ | | | |
| **Line** | | | | | | | √ | | √ |
| **Clothoid** | | | | | | | | √ | |
| **Offset** | | | | | | | | | √ |

**TABLEAU 3.1:** Comparabilité réciproque des types de courbes, après prétraitement.

- Line-Offset : une courbe parallèle à une droite est une droite de même vecteur de direction et dont la position est calculable à partir du vecteur unitaire orthogonal à la première droite.

Il existe d'autres cas où les courbes parallèles sont équivalentes à des courbes plus complexes (polylignes, B-Splines), mais ces équivalences dépassent le cadre de cette thèse.

Il convient à présent d'intégrer cette analyse au sein du comparateur structurel. Pour cela, il est nécessaire d'apporter des algorithmes de comparaison, en fonction du type commun des deux entités (d'après la définition 14, le type commun existe). Cet algorithme remplace alors l'algorithme usuel qui consiste à comparer tous les attributs deux à deux.

Une première fonction donne l'équivalence entre n'importe quelle conique (`IfcConic`) et permet ainsi de traiter l'équivalence Ellipse-Cercle. Ensuite, une fonction propre aux B-Spline donne la possibilité de traiter certaines familles de B-Splines (courbes de Bézier, courbes de Bézier rationnelles), et de trouver des équivalences dans certains cas limites. Une autre fonction de comparaison s'occupe de toutes les courbes finies (`IfcBoundedCurve`), en appelant au besoin les fonctions précédentes, et traitant, de plus, le cas de comparaison Composite-Polyline. Enfin, la fonction de comparaison de toutes les courbes (`IfcCurve`) traite aussi les cas de comparaison entre les courbes parallèles d'une part et les droites ou cercles d'autre part.

### 3.4.4 Equivalences sémantiques pour la localisation spatiale

Dans le paradigme IFC, toute entité héritant de l'entité `IfcProduct` peut posséder une représentation et une localisation. L'attribut `ObjectPlacement` de cette entité est de type `IfcObjectPlacement`. Ce dernier peut revêtir une des trois formes suivantes, par héritage :

1. **Repère local** – `IfcLocalPlacement` : il s'agit de définir un système d'axes 2D ou 3D dans le repère monde ou local à un autre repère (ce dernier étant lui-même un objet de type `IfcObjectPlacement`).
2. **Placement sur une grille** – `IfcGridPlacement` : il s'agit de définir le système d'axes en

fonction d'une grille globale de conception.

3. **Placement par rapport à un objet de référence** – `IfcReferencePlacement` : introduit dans l'extension IFC-Bridge, il s'agit de placer un objet par rapport à un autre de référence. La seule implémentation d'un tel placement consiste en l'utilisation d'une courbe de référence : `IfcReferenceCurvePlacement` (cf. annexe A).

En fonction du logiciel utilisé, certains modes représentations peuvent être mélangés. Le fait d'utiliser un autre système de placement ne conduit pas à modifier un objet de construction si la localisation finale est identique. Pour traiter les équivalences de localisation géométrique, une solution consiste à définir une fonction de comparaison pour les entités de type `IfcObjectPlacement`. Dans cette fonction, quelque soit le mode de représentation choisi, la localisation est convertie en un système d'axes dans le repère monde. Pour cela, nous rappelons qu'il est possible d'associer à ce système d'axes une matrice de passage $4 \times 4$ en coordonnées homogènes [Weisstein99] [Graustein30]. L'utilisation des coordonnées homogènes, plutôt que des coordonnées cartésiennes d'un espace vectoriel, est due au fait que les translations deviennent des applications linéaires, ce qui autorise à effectuer des changements de repère entre systèmes ayant des origines différentes du point $(0, 0, 0)$. L'objectif de cette partie consiste donc à indiquer la démarche globale de conversion de chaque mode au système d'axes souhaité.

La définition d'un repère local s'effectue par l'intermédiaire de l'entité `IfcLocalPlacement` :

```
ENTITY IfcLocalPlacement
SUBTYPE OF (IfcObjectPlacement);
 PlacementRelTo :  OPTIONAL IfcObjectPlacement;
 RelativePlacement :  IfcAxis2Placement;
[...]
END_ENTITY;
```

Cette entité possède deux attributs : `RelativePlacement` possède le système d'axes orthogonal et `PlacementRelTo` référence un autre placement. Si ce dernier est omis, alors la localisation s'effectue dans le repère monde et le travail est terminé. Précisons la structure de `IfcAxis2Placement` :

```
TYPE IfcAxis2Placement = SELECT
(IfcAxis2Placement2D,
IfcAxis2Placement3D);
END_TYPE;

ENTITY IfcPlacement
[...]
Location :  IfcCartesianPoint;
[...]
END_ENTITY;

ENTITY IfcAxis2Placement3D
SUBTYPE OF (IfcPlacement);
 Axis :  OPTIONAL IfcDirection;
 RefDirection :  OPTIONAL IfcDirection;
```

```
DERIVE
 P :  LIST [3:3] OF IfcDirection :=  IfcBuildAxes(Axis, RefDirection);
 [...]
END_ENTITY;
```

Dans le cas 3D, le système d'axes se compose de la localisation d'un point (attribut `Location` de l'entité `IfcPlacement`), de l'axe Z qui définit aussi le plan XY (orthogonalité du repère), via l'attribut `Axis`, et de l'axe X, via l'attribut `RefDirection`, qu'il convient de projeter sur le plan XY si nécessaire. Si ces attributs sont omis, le repère canonique est utilisé. L'obtention d'un système d'axes dans le repère monde s'effectue alors de manière récursive : à ce système d'axes est associée une matrice de passage, multipliée à la matrice de passage du repère spécifié par l'attribut `PlacementRelTo`. Si ce dernier référence une entité `IfcLocalPlacement`, la récursion continue.

L'utilisation d'une grille de conception, pratique courante en CAO, est référencée par l'entité `IfcGridPlacement` :

```
ENTITY IfcGridPlacement;
SUBTYPE OF (IfcObjectPlacement);
 PlacementLocation :  IfcVirtualGridIntersection;
 PlacementRefDirection :  OPTIONAL IfcVirtualGridIntersection;
END_ENTITY;

ENTITY IfcVirtualGridIntersection;
 IntersectingAxes :  LIST [2:2] OF UNIQUE IfcGridAxis;
 OffsetDistances :  LIST [2:3] OF IfcLengthMeasure;
END_ENTITY;

ENTITY IfcGridAxis;
 AxisTag :  OPTIONAL IfcLabel;
 AxisCurve :  IfcCurve;
 SameSense :  IfcBoolean;
[...]
END_ENTITY
```

Cette méthode spécifie d'abord la localisation de l'origine $O$ par intersections d'axes de grilles. Cet axe est une courbe parallèle à une courbe prédéfinie par l'entité `IfcGridAxis`, la distance de cette parallèle étant donnée par l'attribut `OffsetDistances`. Un autre point $P$ (optionnel) permet de donner une direction particulière à l'axe X, portée par le segment $[OP]$. Les autres axes sont construits de manière canonique en fonction du repère monde. La difficulté de cette partie réside dans le calcul du ou des point(s) d'intersections entre courbes. Les détails de calculs dépassent le cadre d'étude de cet exemple. Le calcul des autres axes demeure cependant aisée et permet d'obtenir la matrice de passage $4 \times 4$.

Enfin, l'extension aux ponts IFC-Bridge définit un autre mode de localisation, en fonction d'une courbe de référence (implémentation au 01/01/2007, avant intégration au modèle IFC) :

```
ENTITY IfcReferenceCurvePlacement;
SUBTYPE OF (IfcReferencePlacement);
```

```
 CurvilinearAbscissa : IfcLengthMeasure;
 Axis : IfcDirection;
 RefDirection : IfcDirection;
 RelativeTo : IfcReferenceCurvePlacementSystem;
END_ENTITY;

ENTITY IfcReferenceCurvePlacementSystem
SUBTYPE OF (IfcReferencePlacement);
 Label : STRING;
 BasedOn : IfcReferenceCurve;
END_ENTITY;

ENTITY IfcReferenceCurve3D
SUBTYPEãOFä(IfcReferenceCurve);
 Curve3D : IfcCurve;
END_ENTITY;
```

La définition des axes ressemble à celle de l'entité `IfcAxis2Placement3D`, seule la position de l'origine n'est pas explicite mais calculée en fonction d'une abscisse curviligne d'une courbe donnée par l'entité `IfcReferenceCurve`. La difficulté consiste donc à calculer cette coordonnée explicitement, le reste du travail étant similaire à celui effectué sur `IfcLocalPlacement`, sans récursion.

## 3.5 Conclusion

Dans cette partie, nous avons pu proposer plusieurs méthodes pour enrichir sémantiquement un système de comparaison purement structurelle à la base. Des mécanismes simples d'extraction d'information ont été proposés, afin de simplifier les données à comparer et à traiter des informations susceptibles d'être mal traitées par le système de comparaison structurelle. La définition d'équivalents sémantiques augmente pour sa part la pertinence des résultats de comparaison : deux informations sémantiquement équivalentes malgré une structuration différente auront plus de chance d'être détectées comme équivalentes, contrairement à une approche structurelle pure.

Les limitations d'une telle approche résident surtout dans la mise en oeuvre de l'analyse sémantique : le système en lui-même est incapable de définir des équivalents sémantiques ou d'extraire l'information « judicieuse ». Une personne connaissant finement le modèle IFC et IFC-Bridge fournit manuellement un ensemble de règles pour l'extraction d'informations et des fonctions de comparaison pour les équivalents sémantiques. Dans le cadre de ces travaux, nous avons proposé quelques exemples d'applications pour assurer une comparaison correcte sur les modèles d'ouvrages que nous avons testés, mais ces exemples ne correspondent pas à une analyse sémantique exhaustive du modèle IFC.

De plus, nous avons pour l'instant détaillé seulement l'approche théorique et l'application logique aux modèles IFC et IFC-Bridge. Afin de tester la validité de ces approches, il convient de proposer des méthodes d'implémentation. Il s'agit par conséquent d'implémenter le système

de comparaison structurelle, ainsi que son enrichissement sémantique. A partir de cette implémentation, il conviendra alors de vérifier si l'ensemble est intégrable dans le système d'accès à la MNC, compatible avec le processus-type pour la gestion d'un projet en conception (cf. section 1.5.2 du premier chapitre).

## Résumé

L'objectif de ce chapitre est d'enrichir le système de comparaison conçu au chapitre précédent par une prise en compte de la sémantique du modèle de données. Inspirée de l'approche basée sur les transformations de modèles [Katranuschkov06], notre analyse reprend deux procédés essentiels : l'extraction d'information et les équivalents sémantiques. Le premier consiste à sélectionner l'information à comparer, sans devoir créer un autre modèle de données complètement différent, mais en modifiant légèrement le modèle de données initial. Dans ce procédé, nous identifions trois types d'extraction : la sélection par type des entités à comparer, la transformation d'attributs d'entités (elle consiste à couper des liens entre entités ou à introduire des liens virtuels entre entités, suite à une sélection par type par exemple) et enfin la gestion des transformations implicites d'un modèle d'ouvrage. Ces dernières ont lieu lorsque le modèle de données prévoit lui-même des mécanismes de versionnement. La gestion de ces transformations s'effectue alors par un prétraitement, en temps constant par entité, qui transforme le graphe de données durant la comparaison. Le deuxième procédé vise à définir des équivalences sémantiques entre deux entités *a priori* différentes dans le modèle de données initial. D'abord, l'équivalence entre entités élémentaires n'a lieu que si elles instancient au moins une entité en commun (hypothèse 4). Par exemple, définir une équivalence entre trois modes de représentation d'un point 3D (cartésien, cylindrique et sphérique) suppose qu'elles héritent toutes les trois d'une entité `Point`. Ensuite, il suffit de définir une relation d'équivalence dans l'ensemble des instances possibles de l'entité commune, dans cet exemple l'entité `Point`. Des équivalences sémantiques peuvent aussi être définies pour des types simples (exemple : VIADUC $\equiv$ Viaduc $\equiv$ viaduc) Cependant, cette démarche ne fonctionne pas avec des relations de tolérance, notamment à cause de la non-transitivité de ces relations, ce qui rend incohérent les résultats de comparaison en cas de dérive lente d'une valeur. Comme ce type de relation est courante dans le monde de la construction, une méthode est élaborée pour traiter ce cas : lors de la comparaison entre une instance originale et une instance modifiée, la comparaison réajuste une valeur jugée équivalente à une autre au niveau de l'instance modifiée, la différence étant à l'origine d'une éventuelle dérive lente. A partir de cette analyse théorique, l'application de ces méthodes au modèle IFC-BRIDGE augmente la pertinence des résultats du système. Les entités de relations, traitées de manière incorrectes par le logiciel ArchiCad© 9, sont ignorées suite à une sélection par type et l'ajout d'attributs virtuels. De plus, l'entité `IfcOwnerHistory`, indiquant le dernier changement effectué sur cette entité (ajout, suppression, modification), est traitée comme une transformation implicite du modèle d'ouvrage. Enfin, deux applications des équivalences sémantiques au modèle IFC-BRIDGE sont effectuées sur les courbes géométriques et sur la localisation spatiale.

## Abstract

This chapter aims to extend the comparison system (previously designed) by considering semantics within the data model. Inspired by model mapping approach [Katranuschkov06], our analysis reuses two essential processes : extraction of information and semantic equivalences. The first one relates to a selection of information to be compared, without any creation of a really new data model. The initial data model is only slightly modified. We identify three kinds of extraction : entities selection by types, entities attributes transform (remove links or add virtual links between entities) and handle of implicit transforms of a product model. They occur when the data model itself provides versioning mechanisms. Preprocessing the product model is our solution to solve this problem. The second process aims to define semantic equivalences between two different entities in the initial data model. First equivalence between entities occurs only when they instantiate at least one common entity (hypothesis 4). For example, defining an equivalence between three spatial representation of a 3D point (cartesian, cylindrical and spherical) needs a common entity `Point` to be inherited. Then defining an equivalence relation within all instances of this common entity is sufficient to solve the problem. Besides semantic equivalences could be defined for simple types (example : BRIDGE $\equiv$ Bridge $\equiv$ bridge). However, it does not work with tolerance relations, especially because these relations are not transitive. Directly using them would make the system unstable if a value is drifting. Since this kind of relation is largely used in AEC sector, this methodology is applied : during the comparison between an original instance and a modified one, the system corrects the value in the modified instance, because it supposes that the difference is a noise responsible of drifting values. From this theoretical analysis, applying this methodology on IFC-BRIDGE data model improves system results relevance. Relation entities are badly handled by ArchiCad© 9. Therefore, they are ignored thanks to an entities selection by type and the addition of virtual attributes. Besides, `IfcOwnerHistory` entity, which describes the last change of an entity (add, remove, modify), is handled as an implicit transform of the product model. Last two examples of semantic equivalences are given through geometric curves and spatial localisation.

# Chapitre 4

# Implémentations et application à la synchronisation de la MNC

## 4.1 Introduction

Les deux premiers chapitres ont permis la conception d'un système de comparaison structurelle, enrichi d'une analyse sémantique. Un système de comparaison sémantique a donc été élaboré. Dans le cadre de nos travaux, l'implémentation apporte alors une solution informatique afin de démontrer la faisabilité de notre démarche théorique. L'application à la synchronisation de la MNC permet, de son côté, d'illustrer « l'utilisabilité » de nos travaux à travers plusieurs exemples.

Dans un souci de cohérence au sein de notre démarche et afin d'utiliser les outils et les méthodes usuels de l'informatique actuelle, nous avons choisi l'implémentation dans un langage orienté objets (qui supporte la POO) : le C++. Ce langage, très reconnu dans tous les domaines industriels, reste très exploité pour des logiciels clients, techniques et scientifiques. Ses performances par rapport aux autres langages orientés objets représentent la raison principale de notre choix. Nous verrons au prochain chapitre dans quelle mesure nos travaux pourraient être implémentés dans d'autres langages (comme Java), qui seraient plus adaptés à la communication client/serveur et aux liens avec la persistance des données.

Ce chapitre se décompose de la manière suivante : d'une part l'implémentation du comparateur structurel (section 4.2) et l'analyse sémantique (section 4.3), d'autre part plusieurs intégrations à des sytèmes de gestion de MNC (section 4.4).

## 4.2 Implémentation du système de comparaison structurelle

Rappelons l'objectif du comparateur structurel : soient deux modèles d'ouvrages, instances du même modèle de données. L'un est supposé original et l'autre est supposé modifié. Le système effectue une comparaison de ces instances et donne un résultat. L'implémentation nécessite la spécification de la structure du système ainsi que des données en entrée et en sortie :

- Le modèle de données est spécifié en langage EXPRESS, conformément aux développements du deuxième chapitre.
- Les modèles d'ouvrages sont spécifiés selon la norme STEP-21 (cf. section 1.2.2 du premier chapitre).
- Le système de comparaison est une bibliothèque en langage C++
- Le résultat est un document XML (sans présentation ou mise en page)

La figure 4.1 rappelle le schéma de la figure 2.1 du deuxième chapitre, en spécifiant cette fois la structure de chaque élément.

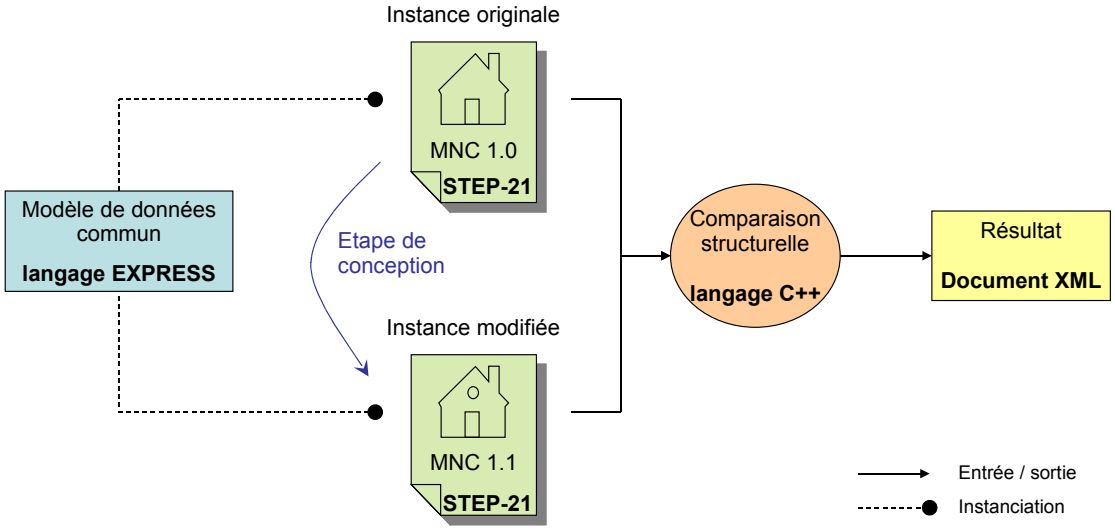


**FIGURE 4.1 :** Spécification de la comparaison structurelle.

Afin d'apporter une méthodologie d'implémentation du comparateur structurel, il est proposé une technique de génération automatique (section 4.2.1), avant de développer un diagramme des classes de comparaison (section 4.2.2). Ensuite, l'implémentation des informations de base nécessaires au comparateur s'effectue grâce à un assistant structurel (section 4.2.3). Enfin, la structure des résultats de comparaison est détaillée à la section 4.2.4.

### 4.2.1 Génération automatique du moteur de comparaison

Pour pouvoir implémenter un système de comparaison de modèles d'ouvrages, spécifiés suivant le standard STEP, nous avons besoin d'une bibliothèque permettant de gérer des modèles de données écrits en langage EXPRESS. En effet, une telle bibliothèque permet de charger un modèle d'ouvrage, d'en lire les propriétés, éventuellement de les éditer, et de sauver le modèle modifié. Il existe des solutions commerciales, comme ST-Developer© de STEP-Tools [STEP-Tools06b], et EXPRESS Data Manager© de EPM Technology [EPM06]. Ces outils possèdent des fonctionnalités très avancées (transformation de modèles, validation de modèles) mais se révèlent aussi très onéreux. C'est pourquoi il existe une solution développée par l'Université de Manchester, sous contrat de l'Agence Spatiale Européenne (ESA) : *Expressik*. Il s'agit d'un produit Open-source (non encore publié à ce jour) pour générer automatiquement des

bibliothèques en langage C++, capables de gérer un modèle de données EXPRESS. La bibliothèque générée respecte la norme ISO 10303-23 (STEP-23) en termes de fonctionnalités.

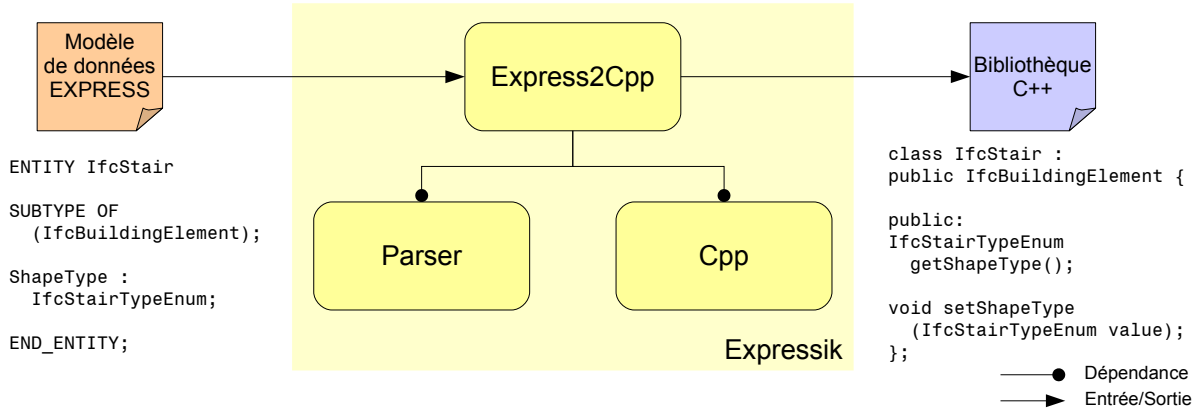


**FIGURE 4.2 :** Architecture globale de l'application Expressik.

La figure 4.2 représente l'organisation globale de cette application. Cette dernière est divisée en trois paquetages distincts :

- Paquetage **Parser** : il transforme le fichier d'un modèle de données écrit suivant le langage EXPRESS en une structure hiérarchisée en mémoire, où toutes les informations sont accessibles (y compris les commentaires du fichier EXPRESS).
- Paquetage **Cpp** : il transforme des concepts de programmation orientée objets en des lignes de code C++, sous la forme usuelle de fichiers d'inclusion '.h' et de sources '.cpp'.
- Paquetage **Express2Cpp** : il s'agit du coeur de l'application. Il prend en entrée le fichier d'un modèle de données (par exemple, la description EXPRESS du modèle IFC ou IFC-Bridge), et génère les sources d'une bibliothèque C++. Pour cela, ce paquetage utilise naturellement les deux précédents.

L'approche choisie par Expressik est dite *Early-binding* : la bibliothèque générée n'est capable de gérer qu'un seul modèle de données EXPRESS, mais toutes les méthodes de gestion sont compilées, par conséquent les performances d'accès aux données sont optimisées. Cette approche s'oppose à celle dite *Late-binding* : la bibliothèque est générique, donc capable de gérer n'importe quel modèle de données, mais les méthodes de gestion sont créées dynamiquement, au chargement du modèle d'ouvrage. Les performances sont donc moins bonnes que dans l'approche Early-binding.

Dans le cadre de nos travaux, et même au-delà, la division MODEVE du CSTB a développé une version simplifiée du paquetage Express2Cpp : *Express2LightCpp*. En effet, le respect de la norme ISO 10303-23 se révèle suffisamment contraignant pour nuire aux performances de la bibliothèque. En limitant la gestion des héritages aux héritages simples et en ignorant les propriétés calculées (attributs DERIVE, clause WHERE, etc. tous les concepts de la catégorie **PC**, cf. section 2.3.3 du deuxième chapitre), le paquetage Express2lightCpp obtient de meilleurs performances et un accès aux données simplifié. Bien qu'il ne respecte cependant pas la norme ISO 10303-23, cela n'entraîne pas de conséquences négatives pour les travaux de cette thèse.

Nous avons choisi d'implémenter le système de comparaison structurelle suivant la même démarche *Early-binding* : la bibliothèque C++ associée au comparateur est générée automatiquement pour un modèle de données, à partir de sa description EXPRESS. Par conséquent, le code pour chaque type d'entité est compilé, les performances sont donc optimisées. La bibliothèque C++ générée devient alors compatible avec celle créée par Express2LightCpp. L'organisation globale des ces applications est représentée sur la figure 4.3.

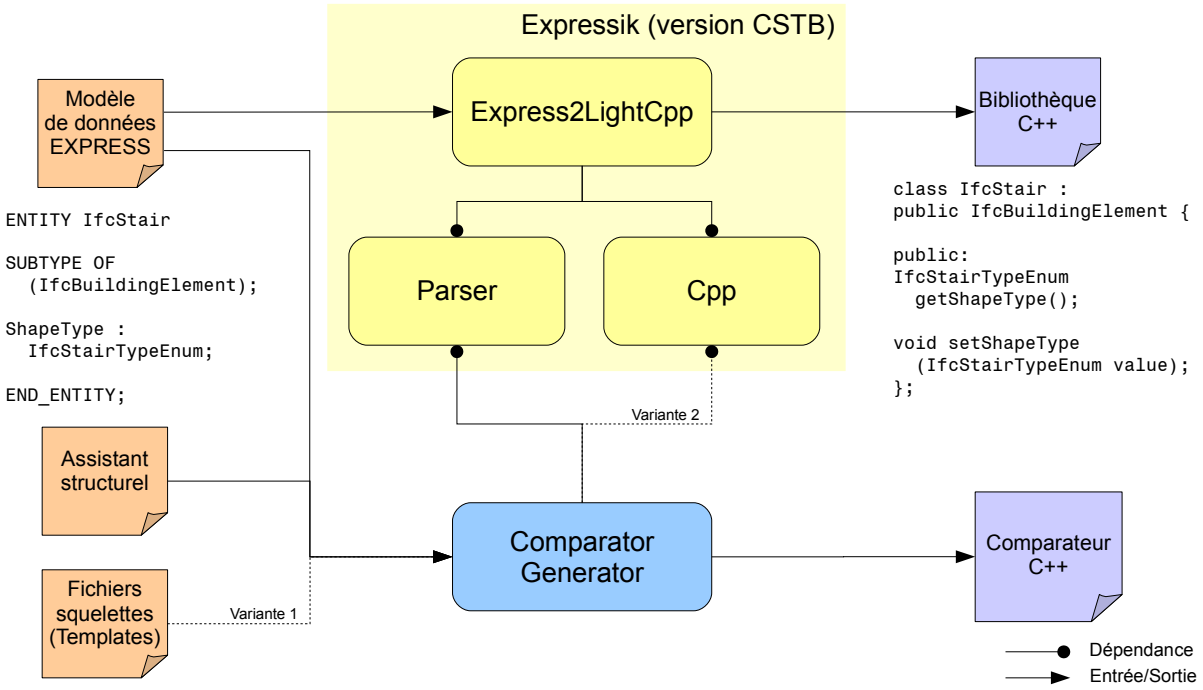


**FIGURE 4.3 :** Architecture générale pour la génération du système de comparaison.

En entrée, le générateur a besoin de la description EXPRESS du modèle de données (évidemment identique à celle utilisée pour générer la bibliothèque C++ qui gère les modèles d'ouvrage), ainsi que d'assistants (qui seront détaillés à la section 4.2.3). Ensuite, cette architecture autorise deux variantes :

1. L'utilisation de fichiers squelettes (*template*) en entrée : ils stockent la structure statique des sources de la bibliothèque C++. Les éléments dynamiques, qui dépendent de la description EXPRESS du modèle de données, sont localisés dans ces fichiers par des balises XML. La mise en oeuvre du générateur est plus rapide au début, mais cette technique se révèle moins flexible pour générer un code avec des dépendances complexes au modèle EXPRESS.
2. L'utilisation du paquetage Expressik **Cpp** : ce paquetage permet de générer n'importe quel algorithme en C++. La flexibilité est donc maximale mais la mise en oeuvre du générateur plus fastidieuse, car une ligne de code C++ générée peut correspondre à une dizaine de lignes de code dans le générateur.

La figure 4.4 confronte ces deux variantes. Si la plupart des résultats sont basés sur une

approche utilisant des fichiers squelettes, certaines limitations nous ont conduit à considérer dans un deuxième temps l'approche du paquetage **Cpp**, en particulier lors de l'implémentation de l'analyse sémantique, étude abordée à la section 4.3.

Variante 1 : Fichiers squelettes XML

```
<template>
[...]
<header>
#ifndef <USN/>_STATE_COMPARATOR_H
#define <USN/>_STATE_COMPARATOR_H

#include "Comparator.h"

class <USN/>_COMPARATOR_DLL_DEF NidComparator;

class <USN/>_COMPARATOR_DLL_DEF StateComparator
                  : public Comparator {
public:
  NidComparator* nidCompare;
  StateComparator(unsigned int groupId);
  ~StateComparator() {}

  <idEntities>
  virtual bool visit<ECN/>(<SN/>::<ECN/>* p_<ECN/>);
  </idEntities>

  <nidEntities>
  virtual bool visit<ECN/>(<SN/>::<ECN/>* p_<ECN/>);
  </nidEntities>
};
#endif
</header>
[...]
</template>
```

Variante 2 : Code Java avec paquetage 'Cpp'

```
package comparator.statecomparator

import expressik.parser.*
import expressik.cpp.*

[...]

ClassDatatype scClass =
    new ClassDatatype("StateComparator");
scClass.setExported(true, dllDef);

Variable groupId = new Variable("groupId",CPP.INT);
scClass.addConstructor(new Constructor(groupId));
scClass.addDestructor(new Destructor());

for(int i=0;i < idEntities.length; i++) {
  Function visit =
    new Function("visit" + idEntities[i].getName());
  visit.setDatatype(CPP.BOOL);
  visit.setModifier(Modifier.VIRTUAL);
  Variable param =
   new Variable("p_" + idEntities[i].getName(),
                idEntities[i].getDatatype());
  visit.addParameter(param);
  scClass.addFunction(visit);
  [...]
}

[...]
```

**FIGURE 4.4 :** Comparaison de deux méthodes pour la structure du comparateur structurel

Après avoir défini la façon d'implémenter le système de comparaison, il convient d'étudier le contenu de cette implémentation, les différentes classes du système, la communisation entre elles, ainsi que leurs méthodes.

### 4.2.2 Implémentation des classes du moteur de comparaison

Au deuxième chapitre, l'algorithme général de la comparaison structurelle a été conçu, illustré par l'organigramme de la figure 2.11. Le but de cette partie consiste à reprendre cette conception *fonctionnelle* pour concevoir le système d'un point de vue POO. Il s'agit d'élaborer le diagramme des classes intervenant dans ce système pour satisfaire le déroulement de l'algorithme. Il convient donc de répondre aux problèmes suivants :

- Parcours simultané des deux instances.
- Comparaison des identifications.
- Comparaison de l'état :
  - Attributs simples.
  - Attributs référençant des entités non identifiables (technique de hachage si nécessaire).
- Comparaison de la position relative dans le graphe.

De plus, en utilisant une bibliothèque externe pour la gestion du modèle de données EXPRESS, il n'est pas possible de modifier les classes existantes pour la gestion de chaque entité du modèle de données. Or, nous avons besoin d'ajouter de nouvelles opérations à ces entités, qui

consistent à les parcourir, les comparer et leur affecter des statuts de comparaison. Pour répondre à cette problématique, il existe une technique de conception informatique (*Design Pattern* en anglais) : le visiteur [Gamma94]. Cette technique permet d'ajouter de nouveaux comportements à un ensemble de classes sans les modifier. Pour cela, cet ensemble de classes requiert seulement de supporter le mécanisme général des visiteurs. Il convient de préciser que ce mécanisme ne fonctionne qu'avec des modèles simples d'héritage (pas d'héritage multiple ni d'héritage AND-OR). La version *Express2LightCpp* supporte cette fonctionnalité, contrairement au paquetage original *Express2Cpp*, car ce dernier gère tous les modèles d'héritage possibles.

Pour créer un visiteur, il suffit d'hériter de la classe `Visiteur`, fournie par la bibliothèque associée au modèle de données. Ce nouveau visiteur possède alors une méthode par classe de cette bibliothèque, correspondant à l'application d'une fonctionnalité à cette classe. La correspondance entre la bibliothèque C++ et le modèle de données EXPRESS associé est telle qu'à une entité du modèle correspond une classe de la bibliothèque. De plus, le principe de séparation des interfaces (Interface Separation Principle) [Martin96] déconseille l'utilisation d'une seule interface ou classe pour plusieurs fonctionnalités. Ce principe s'applique particulièrement aux visiteurs, car pour un visiteur, il n'existe qu'une méthode par entité du modèle de données EXPRESS.

C'est pourquoi nous avons choisi de séparer les fonctionnalités de notre système en plusieurs visiteurs. Une première catégorie de visiteurs représente les *comparateurs*. Dans cette catégorie, le visiteur analyse deux modèles d'ouvrages, représentés par deux ensembles d'instances de classes. Or, un visiteur « classique » opère sur un seul ensemble d'instances. Aussi un comparateur possède-t-il une propriété `original`, qui représente l'instance en cours d'analyse du côté de l'instance originale. Le comparateur s'applique donc en tant que visiteur sur l'instance modifiée. Trois premiers comparateurs sont ainsi conçus :

1. `SemanticComparator`[1] : à partir d'une entité de l'instance originale (propriété `original`) et d'une entité de l'instance modifiée (fournie en paramètre de ce visiteur), les méthodes de cette classe vérifient si ces entités ont déjà été visitées, comparent les identifications persistantes, et appellent le comparateur `StateComparator` si les identifications correspondent.
2. `StateComparator` : l'état des deux entités est comparé. Les méthodes de cette classe confrontent en particulier les attributs simples. Concernant les références aux entités non identifiables, il existe deux variantes que nous verrons plus loin.
3. `ChildComparator` : à partir des deux entités comparées s'effectue le parcours des entités enfants. Cette classe assure le parcours simultané des deux structures. Elle appelle ainsi le comparateur `SemanticComparator` avec de nouvelles entités en paramètres. Par exemple, cette classe s'occupe de gérer le cas où certaines entités enfants sont référencées par des ensembles non ordonnés. Il convient alors de faire des tests d'identifications persistantes

[1] Il s'agit en fait d'un comparateur structurel, tant que les assistants sémantiques n'ont pas été implémentés. Mais au début de nos travaux, cette distinction n'avait pas été faite, d'où le nom de cette classe.

pour envoyer les couples d'entités corrects (instance originale/instance modifiée) au visiteur `SemanticComparator`.

Il manque deux fonctionnalités à cette implémentation : la comparaison de la position relative dans le graphe, et la comparaison d'entités non identifiables. Concernant le premier point, l'organigramme de la figure 2.11 (deuxième chapitre) suggérait d'effectuer cette comparaison juste après la comparaison d'états, pour chaque entité. En réalité, cette approche se révèle peu performante, car en mémoire, les entités du modèle d'ouvrage ne connaissent pas leurs entités parentes[2]. Pour chaque entité visitée, cela nécessiterait d'effectuer un parcours de toutes les entités susceptibles de référencer l'entité en cours. C'est pourquoi nous avons choisi une autre approche pour l'implémentation : au début de la visite d'une entité, au niveau du comparateur `SemanticComparator`, il est possible de connaitre l'entité parente qui référence l'entité en cours (via un paramètre supplémentaire fourni par `ChildComparator` au `SemanticComparator`). Une référence à l'entité parente est sauvegardée pour l'entité en cours. Comme le parcours est exhaustif, tous les liens sont parcourus, c'est-à-dire que chaque entité connait l'ensemble de ses parents, grâce à ces sauvegardes. Ainsi, la comparaison des positions relatives s'effectue à la fin du double parcours. Les couples d'entités reconnus (possédant donc le statut 'OK' ou 'changé') sont comparés grâce à cette nouvelle information.

Concernant le deuxième point sur la comparaison d'entités non identifiables, deux méthodes ont été étudiées. La première consiste à créer un comparateur dédié aux entités non identifiables : `NidComparator` ('Nid' pour 'non identifiable'). Cette classe gère non seulement la comparaison des attributs de ces entités, mais aussi le parcours des liens entre entités non identifiables. Après chaque comparaison de couples d'entités non identifiables, le résultat est sauvé, afin de ne pas effectuer une même comparaison plusieurs fois. Cela peut arriver si une entité non identifiable (un point 3D par exemple) est utilisée plusieurs fois par des entités identifiables.

La deuxième variante utilise la méthode de table de hachage. Cette technique a déjà été introduite au chapitre 2, à la section 2.4.2. Il s'agit de transformer n'importe quelle structure en un nombre entier, comme l'illustre la figure 4.5.

Si des entités non identifiables référencent d'autres entités identifiables via des attributs, le hachage devient récursif. Cependant, le risque d'utiliser une telle technique est le suivant : la garantie que deux structures différentes génèrent deux hachages différents est seulement d'ordre statistique. En effet, il existe une infinité de structures alors que le nombre de hachages possible est fini. Le paradoxe des anniversaires indique que la probabilité d'une *collision* n'est pas négligeable [Weisstein03]. Une formule permet de l'estimer : soit $n$ le nombre d'entités susceptibles d'avoir un hachage. Soit $d$ le nombre de valeurs possibles pour le hachage. La probabilité de collision est alors approximée par la formule suivante :

$$p(n,d) \approx 1 - \exp\left(\frac{-n(n-1)}{2d}\right) \tag{4.1}$$

Dans le cadre de nos travaux, les modèles d'ouvrage les plus volumineux possèdent moins de 400000 entités. En utilisant un codage sur 64 bits, on a alors $n = 400000$ et $d = 2^{64}$, d'où

[2]Sauf si les liens de dépendances utilisent des attributs inverses, mais cela n'est pas le cas pour tous les attributs.

$p(n,d) \approx 4.3.10^{-9}$. Nous avons donc choisi de garder cette technique.

L'implémentation d'une technique de hachage est simple, car de nombreuses bibliothèques sont disponibles pour offrir de telles fonctionnalités, comme Boost en C++ [James06]. Pour créer un hachage adapté à chaque entité non identifiable, un nouveau visiteur `HashingVisitor` est réalisé.

Dans un premier temps, nous avons choisi la première variante de manière intuitive, l'idée des tables de hachage n'est venue que dans un deuxième temps pour améliorer les performances et simplifier le traitement. La figure 4.6 représente le diagramme global des classes intervenant dans cette implémentation, suivant le paradigme UML.

Ce système répond à toutes les fonctionnalités demandées. Cependant, conformément aux conclusions du chapitre 2, le moteur de comparaison a besoin d'informations spécifiques au modèle de données pour fonctionner : quelles sont les entités identifiables et leur attribut servant d'identification persistante ? Quelle est l'entité racine ? Pour résoudre ce problème, nous avons choisi d'intégrer un *assistant structurel* à l'implémentation existante.

### 4.2.3 Intégration d'un assistant structurel

Nous définissons ici un assistant comme étant un ensemble d'informations stockées dans un fichier ou une base de données, à fournir en entrée du générateur du système de comparaison (conformément au schéma de la figure 4.6). L'objectif consiste à aider le système pour améliorer les résultats de comparaison, c'est-à-dire que ces derniers s'accordent avec l'interprétation que pourrait en faire un acteur de projet. L'assistant est dit structurel, car l'analyse sémantique ne joue pas encore son rôle ici, les informations données ne concernent que la structure des modèles d'ouvrage à comparer.

Nous avons choisi de stocker les deux types d'information, cités à la fin de la section précédente, dans un fichier XML :

- La balise `Helper` est la racine de l'assistant.
- La balise `root` contient un attribut `name` correspondant au nom de l'entité racine à parcourir.
- Les balises `idEntity` sont associées aux entités identifiables du modèle de données. Chacune d'elles contient :
  - Un attribut `name` qui est le nom de l'entité concernée.
  - Un contenu entre les balises qui représente une fonction C++ ayant pour nom `haveSameId`. Cette fonction prend en paramètres deux instances des entités concernées et renvoie vrai si les identifications sont les mêmes, faux dans le cas contraire.

Le contenu de l'assistant structurel pour les modèles IFC et IFC-Bridge est représenté par le code XML ci-dessous :

```
<Helper>
<root name="IfcProject"/>
<idEntity name="IfcRoot">
```

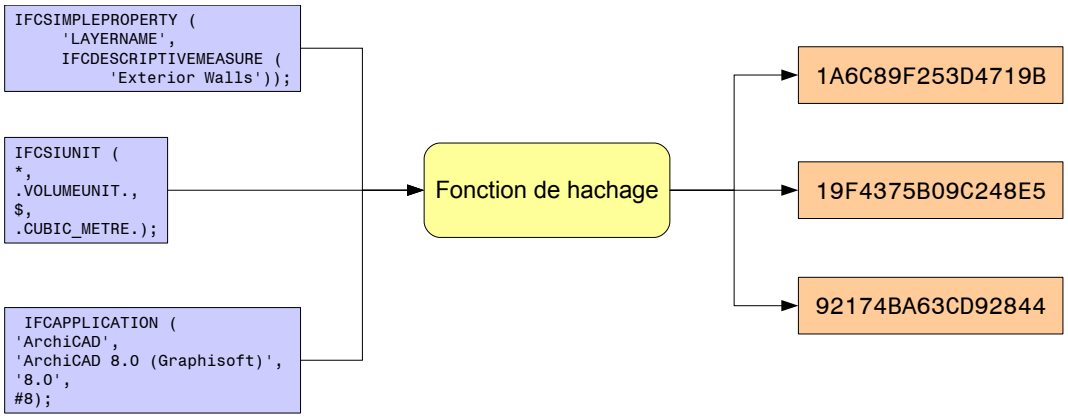


**FIGURE 4.5 :** Exemple d'utilisation d'une fonction de hachage. A droite les nombres entiers résultant, en format hexadécimal, codés sur 64 bits.

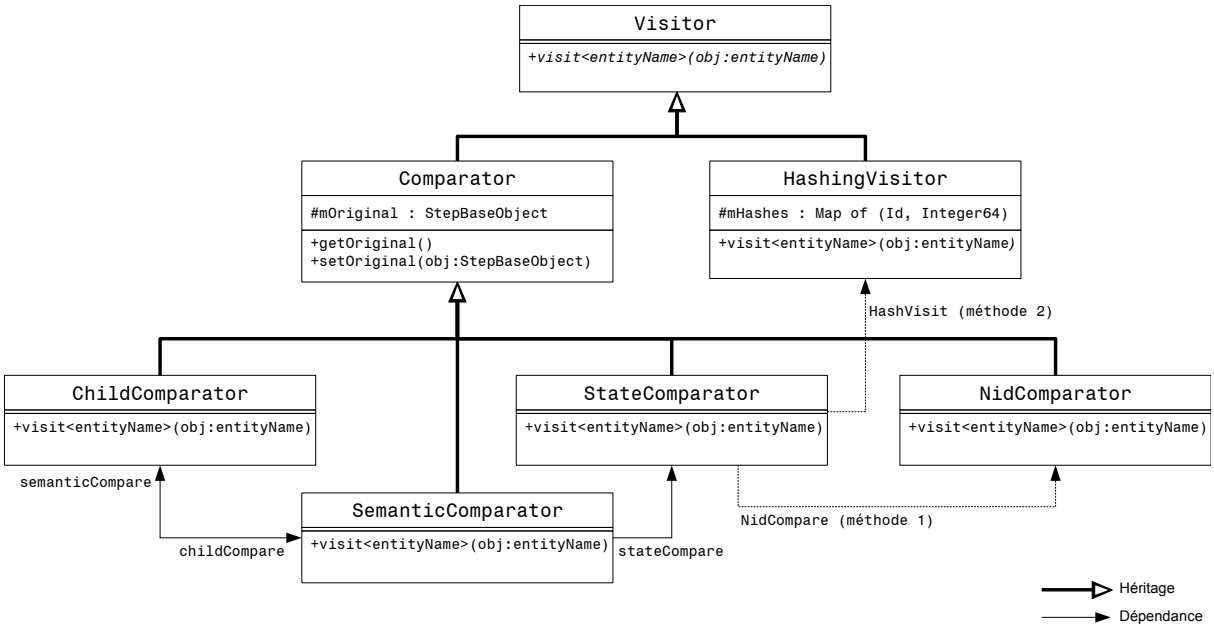


**FIGURE 4.6 :** Diagramme global de classes UML des classes du comparateur structurel.

```
bool haveSameId(IfcRoot* obj1, IfcRoot* obj2)
{
return (obj1->getGlobalId()) == (obj2->getGlobalId());
}
</idEntity>
</Helper>
```

Le système de comparaison est à présent implémenté. Par rapport au diagramme de la figure 4.1, il convient de définir dès lors la structure des résultats de comparaison.

### 4.2.4 Structuration des résultats de comparaison

Quand le système de comparaison a effectué son travail, le résultat se trouve en mémoire. La bibliothèque en C++ permet d'accéder à ces résultats. Cependant, selon l'application visant à utiliser la comparaison structurelle, l'accès direct au résultat en mémoire se révèle difficile, voire impossible. C'est pourquoi l'utilisation d'un fichier de sortie reste une solution potentiellement acceptable pour n'importe quel logiciel susceptible d'utiliser le système.

Comme pour la création de l'assistant structurel, nous avons choisi le standard XML pour hiérarchiser l'information des résultats. Trois types d'informations sont représentés :

- Les **métadonnées** : elles correspondent aux ***métadonnées*** des fichiers à comparer. Nous avons retenu ici le nom de l'instance originale/modifiée, ainsi que son chemin (URL ou chemin local), via la balise `Files` et les sous-balises `original` et `modified`.
- Des références à des entités : ces informations concernent les éléments ayant un statut différent de 'OK'. Chaque entité est référencée grâce à son identification persistante. Par exemple, tous les éléments supprimés de l'instance originale sont référencés, grâce à la sous-balise `Element`, et regroupés grâce à la balise `RemovedElements`.
- Des références à des attributs d'entités : ces informations concernent seulement les éléments ayant un statut 'changé' ou 'changé&déplacé'. Il s'agit d'ajouter aux références d'entités (point précédent) des références aux attributs responsables de la modification d'une entité. Sachant qu'un attribut peut contenir une entité non identifiable, le référencement des attributs est récursif. Ainsi, le nom de l'attribut concerné est ajouté grâce à la balise `Attribute`, au sein de la balise `Element`.

La figure 4.7 représente la structure XML décrite par le paragraphe précédent.

Le système de comparaison structurel devient à présent opérationnel. Nous verrons les résultats obtenus par un tel système à la section 4.4. Cependant, la nécessité d'une analyse sémantique, pour compléter le système, a déjà été démontrée dans les chapitres précédents. Afin d'obtenir un système de comparaison sémantique, nous avons étudié une méthode d'implémentation reprenant la démarche de l'assistant structurel. Il convient alors de définir des assistants *sémantiques*.

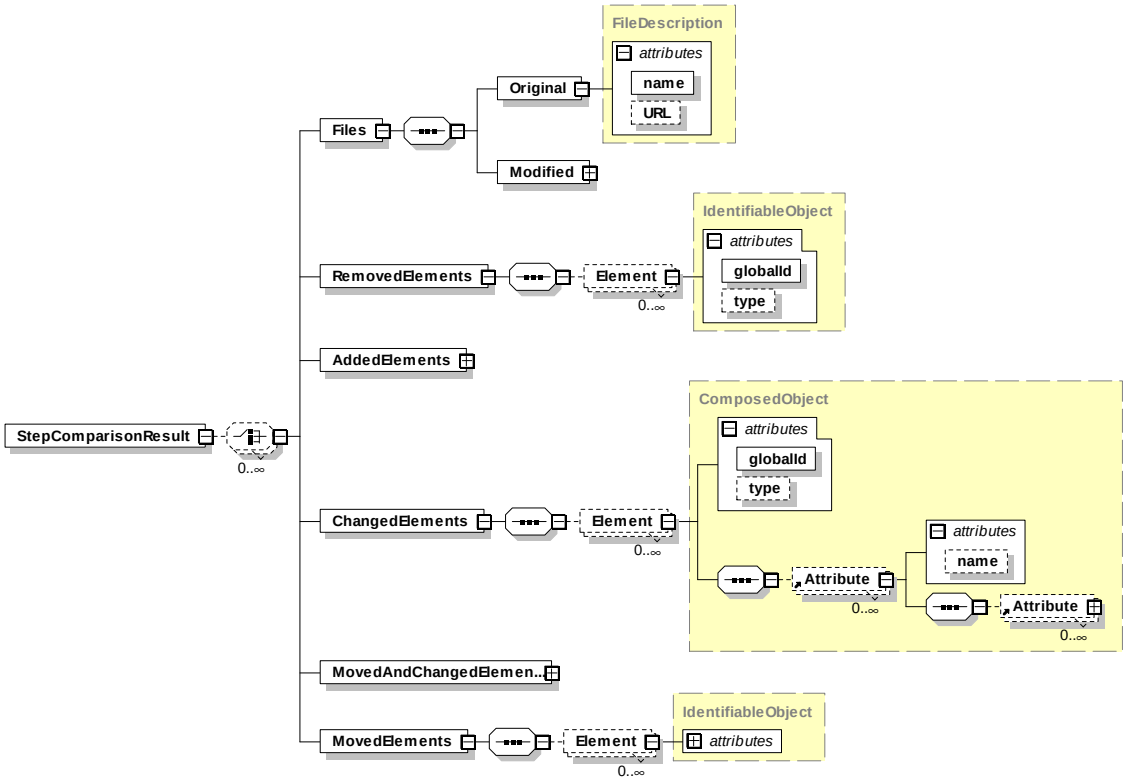


FIGURE 4.7 : Structure des résultats de comparaison. Le diagramme a été obtenu grâce au logiciel Altova XMLSpy©

## 4.3 Méthode d'implémentation de l'analyse sémantique

Au troisième chapitre, nous avons souhaité enrichir le moteur de comparaison structurel, sans modifier pour autant son déroulement global. Il s'agit ici de conserver la même volonté de doter le système de fonctionnalités supplémentaires, sans en modifier l'architecture globale. C'est pourquoi nous avons choisi d'implémenter l'analyse sémantique en ajoutant des informations à l'assistant structurel, afin de créer un *assistant sémantique*.

Compléter l'assistant structurel constitue un choix déterminant pour nos travaux. En effet, conformément à l'architecture générale du comparateur (cf. figure 4.3), cela signifie que l'analyse sémantique est apportée *avant* la génération de bibliothèque C++ de comparaison. Dans cette version, il paraît donc impossible d'ajouter de nouvelles règles sémantiques (tolérance, extraction d'information, etc.) à partir d'une bibliothèque de comparaison déjà créée. L'avantage réside dans la simplicité d'implémentation du système, mais en contre-partie, les projeteurs ne peuvent modifier directement les paramètres de leur système. Cela nécessite l'action d'un créateur d'applications pour regénérer la bibliothèque et intégrer ainsi les nouvelles règles. Nous verrons au prochain chapitre les évolutions possibles du système pour lui fournir plus de flexibilité.

L'implémentation de l'analyse sémantique est développée selon trois axes principaux : le développement de l'assistant d'extraction d'information (section 4.3.1), la création de l'assistant d'équivalents sémantiques (section 4.3.2) et l'adaptation du système à l'adjonction de ces règles, concernant en particulier les techniques de hachage (section 4.3.3).

### 4.3.1 Assistant d'extraction d'information

Dans cette section, l'objectif consiste à implémenter les fonctionnalités d'extraction d'information, à savoir : la sélection d'entités par type, les transformations d'attributs d'entités et la gestion des transformations implicites du modèle d'ouvrage. Pour cela, nous avons choisi de réorganiser et d'étendre les informations données par l'assistant structurel (cf. section 4.2.3), en créant de nouvelles balises :

- `configuration` : cette balise regroupe des propriétés générales inhérentes au comparateur :
  - `root` : elle indique le nom de l'entité racine à parcourir (réorganisation de l'assistant structurel)
  - `defaultEntitiesEnabled` : elle indique par une valeur booléenne si, par défaut, toutes les entités sont sélectionnées. Dans le cas contraire, aucune ne l'est par défaut. Si cette balise est ommise, toutes les entités sont prises en compte.
  - `defaultAttributesEnabled` : elle indique par une valeur booléenne si, par défaut, tous les attributs sont sélectionnés. Dans le cas contraire, aucun ne l'est par défaut. Si cette balise est ommise, tous les attributs sont pris en compte.
- `enableEntity` : elle permet de sélectionner une entité à comparer, dans le cas où elle a été désactivée à cause de la balise `defaultEntitiesEnabled` ou si elle a été implicitement supprimée par une balise `disableEntity`. Si elle hérite d'une ou de plusieurs entités, ces dernières sont implicitement sélectionnées, récursivement en *remontant* le graphe d'héritage.
- `disableEntity` : elle permet de désactiver une entité à comparer. Si plusieurs entités héritent de celle-ci, alors elles sont implicitement désactivées, récursivement en *descendant* le graphe d'héritage.
- `enableAttribute` : elle permet d'activer un attribut lors de la comparaison. Cet attribut peut être simple ou peut référencer une autre entité. Dans tous les cas, l'entité propriétaire de cet attribut doit être sélectionnée, pour assurer la cohérence de l'assistant.
- `disableAttribute` : elle permet de désactiver un attribut lors de la comparaison.
- `virtualAttribute` : elle permet de créer un attribut virtuel. Le contenu de cette balise correspond à la valeur allouée à ce nouvel attribut, en utilisant les attributs du modèle de donnée d'origine.
- `idEntity` : elle est correspond au référencement d'une entité identifiable (réorganisation de l'assistant structurel).
- `nidEntity` : elle permet de déclarer explicitement une entité comme non identifiable, même si elle hérite d'une entité identifiable. Nous verrons plus loin l'intérêt d'une telle définition.
- `preCompare` : elle permet de gérer une transformation implicite par prétraitement d'une

entité. Cette balise contient le code C++ qui sera inséré automatiquement en début de comparaison d'éléments qui instancient cette entité.

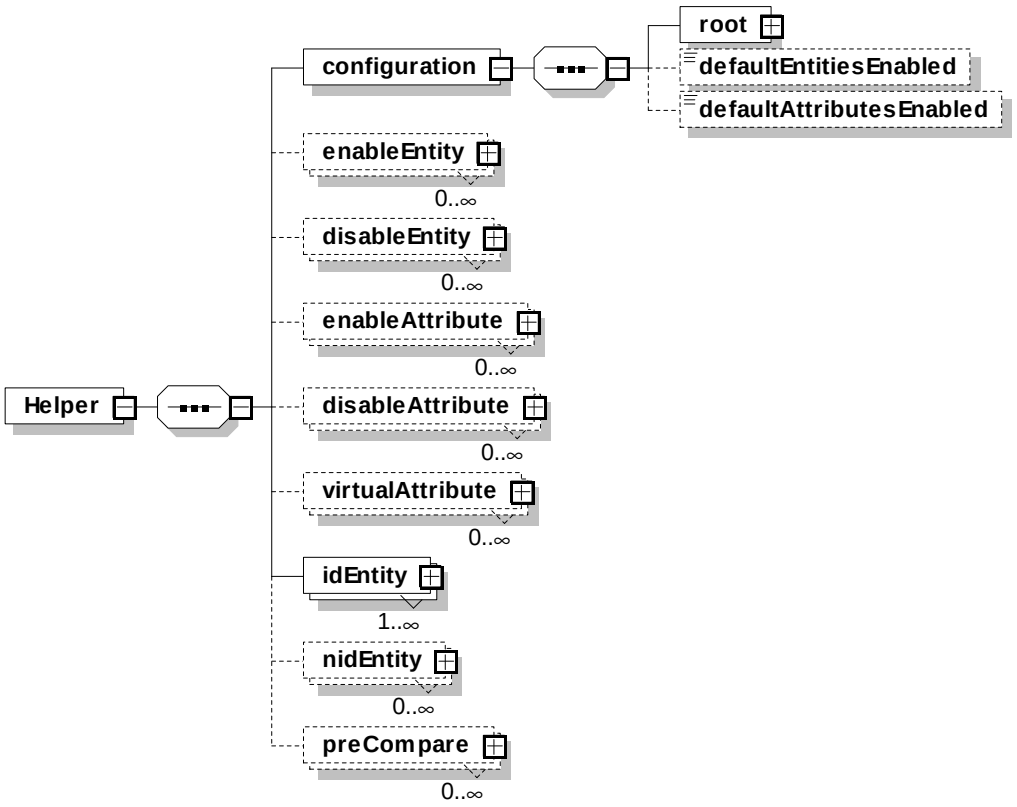


**FIGURE 4.8 :** Structure de l'assistant sémantique d'extraction. Le diagramme a été obtenu grâce au logiciel Altova XMLSpy©

La figure 4.8 montre la structure de l'assistant. L'application au modèle IFC-Bridge devient possible. Ce premier assistant sémantique a besoin d'être complété par des informations concernant les équivalents sémantiques.

### 4.3.2 Assistant d'équivalents sémantiques

Pour implémenter les équivalents sémantiques, il suffit de définir des fonctions personnalisées pour comparer des entités du modèle de données. Notre approche offre deux possibilités :

1. Fournir une fonction de comparaison pour une entité du modèle de données. Tous les attributs ayant pour type cette entité seront comparés via cette fonction. Dans le fichier XML de l'assistant, il s'agit d'une nouvelle balise `compareEntity`.
2. Fournir une fonction de comparaison pour un attribut d'entités du modèle de données. Seul cet attribut sera concerné par l'utilisation de cette fonction pour la comparaison. La nouvelle balise associée se nomme `compareAttribute`.

Dans les deux cas, une information supplémentaire `correction`, pour chacune de ses balises, indique s'il s'agit d'une relation de tolérance, auquel cas la valeur de l'instance modifiée est automatiquement corrigée. La figure 4.9 représente la structure de l'assistant sémantique muni

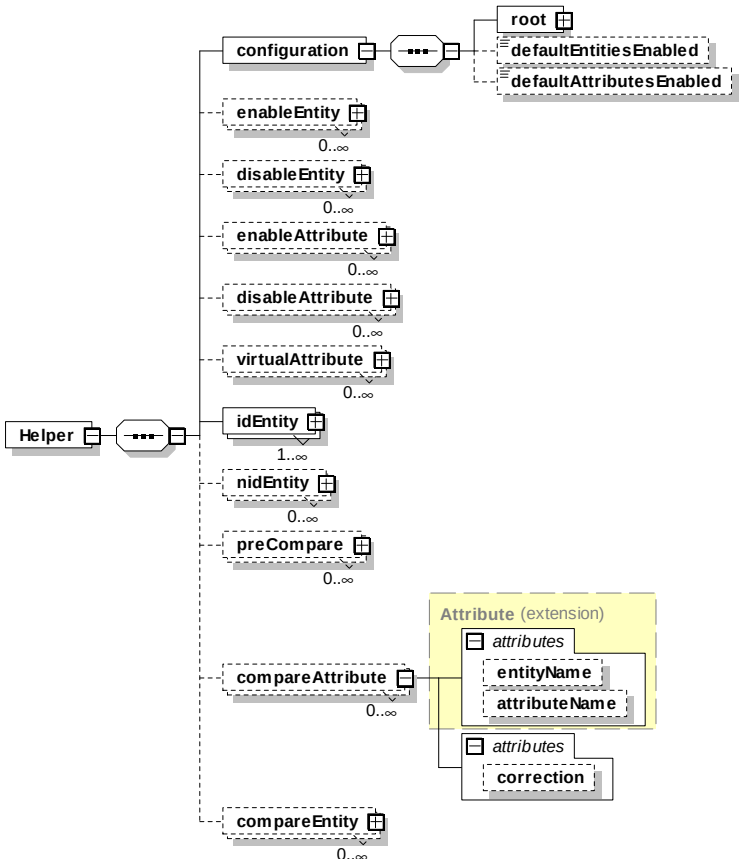


**FIGURE 4.9 :** Structure de l'assistant sémantique complet. Le diagramme a été obtenu grâce au logiciel Altova XMLSpy©

de ses nouvelles balises. Au chapitre précédent, nous avons soulevé le problème résidant dans l'implémentation simultanée des mécanismes d'équivalents sémantiques et des techniques de hachage. Nous proposons donc une méthode pour adapter ces dernières.

### 4.3.3 Adaptation des techniques de hachage

La problématique consiste à fournir un hachage en cohérence avec les équivalents sémantiques. Or, une fonction de hachage est injective (statistiquement) dans le sens où deux données structurellement différentes donnent deux hachages différents. Une solution consisterait alors à transformer les équivalents sémantiques afin de toujours obtenir les mêmes données après transformation. Or, si la relation associée aux équivalents sémantiques est une relation d'équivalence, alors l'espace des valeurs possibles peut être partitionné selon les classes d'équivalence de la relation. Transformer les données revient alors à trouver un représentant unique de chaque classe d'équivalence.

Finalement, nous retrouvons une fonctionnalité vue précédemment dans les transformations de modèle : il s'agit de simplifier sémantiquement les données pour pouvoir comparer deux à deux les valeurs. Par exemple, transformer les coordonnées d'un point, quel que soit son mode

(cartésien, cylindrique, ou sphérique) en coordonnées cartésiennes (cf. figure 1.21 du premier chapitre). Le but de cette partie consiste donc à trouver un représentant unique pour les équivalents sémantiques étudiés dans le cadre du modèle IFC-Bridge, avant de proposer une implémentation pour le calcul du hachage.

Pour la comparaison de courbes, il est difficile de trouver un représentant commun pour toutes les courbes – quel que soit son type – calculable facilement. En effet, nous avons pensé à donner une paramétrisation générale pour tous les types de courbes. Mais cela nécessite l'utilisation de méthodes de calcul formel, ce qui nous entrainerait au-delà du cadre de la thèse. Nous avons aussi pensé à la discrétisation de toutes les courbes, mais nous avons considéré la perte de sémantique trop grande dans ce cas.

Cependant, le tableau 3.1 du troisième chapitre montre que peu de types de courbes sont comparables entre eux. Pour rappel, voici les quatres comparaisons possibles entre types différents :

1. Courbe composite et polyligne.
2. Cercle et ellipse.
3. Courbe décalée et cercle.
4. Courbe décalée et droite.

Nous utilisons deux manières de trouver un représentant commun : soit un type est une spécialisation de l'autre, alors dans ce cas il suffit de transformer systématiquement cette spécialisation en une entité de l'autre type. Soit un type est comparable seulement dans certains cas limites à l'autre type, alors un prétraitement est effectué sur ce premier type, en le transformant en une entité de l'autre type, seulement s'il s'agit d'un cas favorable à la transformation. Pour les courbes composites et les polylignes, nous avons utilisé la première technique : les polylignes représentent un cas particulier des courbes composites. Donc toutes les polylignes sont transformées en courbes composites avant comparaison. De même, pour la comparaison entre cercle et ellipse, tous les cercles sont transformés en ellipses. Concernant les deux dernières comparaisons, avec les courbes décalées, nous avons choisi la deuxième technique : si la courbe décalée est basée sur une droite ou un cercle, alors cette entité est transformée en une droite ou un cercle avec des paramètres calculés. Ensuite, les entités obtenues temporairement n'ont plus besoin d'être comparées avec d'autres types d'entités et grâce au prétraitement (détaillé à la section 3.4.3 du troisième chapitre), deux entités de courbe de même type sont équivalentes sémantiquement si et seulement si elles sont équivalentes structurellement. Ainsi, le hachage est calculable à partir de ces entités temporaires.

Concernant la comparaison de localisations spatiales, la situation devient plus simple car la solution que nous avions abordée au chapitre 3 consistait à transformer tous les modes de représentation possibles (repère local, par rapport à une grille ou à un objet de référence) en une représentation dans un repère global. De plus, ce référentiel commun ne permet pas de définir la même information de plusieurs manières. Les précautions à prendre sont usuelles : normaliser les vecteurs de direction, produire des vecteurs orthogonaux (sinon le modèle en déduit

implicitement un système de vecteurs orthogonaux, il convient d'éviter des calculs implicites, sources de doublons d'information), tronquer les valeurs réelles à une décimale donnée. Ainsi, deux entités sont bien équivalentes sémantiquement si et seulement si elles sont structurellement équivalentes.

L'adaptation de la fonction de hachage s'implémente alors simplement en ajoutant les informations suivantes à l'assistant sémantique : ajouter une sous-balise `generateHash` au sein de l'environnement `compareAttribute` et `compareEntity` de l'assistant d'équivalent sémantique (cf. section précédente). Cette balise contient le code C++ pour générer une valeur de hachage. Ce code effectue donc le prétraitement nécessaire, une seule fois, car le système stocke ensuite cette valeur en cas de comparaisons ultérieures.

Cette partie clôt l'implémentation de l'assistant sémantique, pour enrichir le système de comparaison structurel et créer un véritable comparateur sémantique. Le système est à présent conçu et des méthodes d'implémentation ont été données. Cependant, il ne peut fonctionner de manière optimal seul. Nous avons donc étudié plusieurs applications possibles utilisant ce système de comparaison.

## 4.4 Intégrations au sein de systèmes de gestion de la MNC

### 4.4.1 Visualisation des résultats de comparaison sémantique

Nous avons choisi comme première application, pour l'intégration des résultats de comparaison sémantique, la visualisation des résultats. En effet, les concepteurs utilisent principalement des outils visuels (CAO, viewer 3D, etc.) pour valider leur démarche. Dans cette partie sont abordées deux applications de visualisation qui exploitent le comparateur sémantique.

La première application représente le fruit des travaux de la division MOD-EVE du CSTB : la plateforme EVE. Il s'agit d'un environnement très modulaire pour des applications de réalité virtuelle. Ces applications ont principalament pour but la visualisation de résultats de simulations, dans le cadre du secteur de la construction. La plateforme supporte divers moteurs de visualisation 3D, et surtout divers jeux de données grâce aux modules d'import. La figure 4.10 illustre l'architecture globale de EVE[3].

L'intégration visuelle de la comparaison structurelle devient alors possible. Pour cela, il est nécessaire de créer deux modules : un module utilisant la bibliothèque C++ générée par *Expressik*, afin de charger le modèle en mémoire, d'afficher sa structure hiérarchique globale, de visualiser des propriétés, etc. Les fonctionnalités de la plateforme permettent de sélectionner et de mettre en surbrillance certains objets du modèle. Le deuxième module est dédié à la comparaison sémantique, en utilisant aussi des bibliothèques C++ générées. Il dépend du premier pour le chargement des deux instances. Ensuite, une interface affiche les résultats de comparai-

[3]Dans le cadre de cette thèse, il s'agit de la version EVE2. Depuis 2006, le projet EVE3 propose une plateforme différenciable selon le type d'utilisateur : maîtrise d'ouvrage ou maîtrise d'oeuvre. Il utilise de plus des technologies orientées *Serious Games* [Zyda05] basées notamment sur le moteur Delta3D© [Darken05].

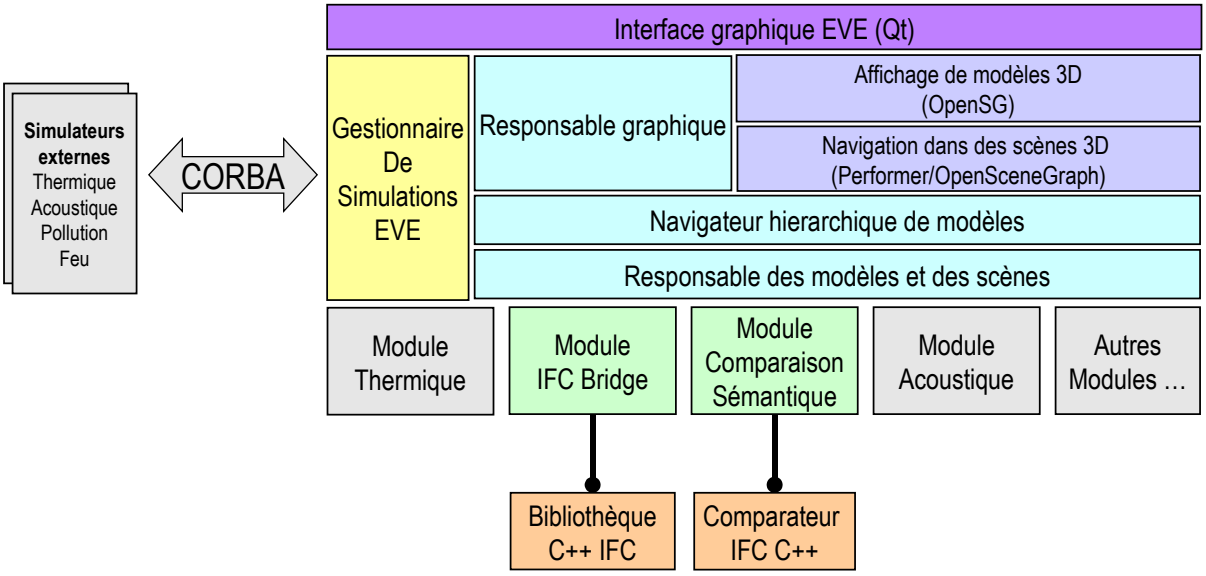


**FIGURE 4.10 :** Intégration de la comparaison sémantique au sein de la plateforme EVE 2.

son. La géométrie des modèles d'ouvrage est affichable en mettant en surbrillance les éléments supprimés, ajoutés, changés, etc. La figure 4.11 illustre le procédé sur un exemple simple d'espace à trois pièces, dont une cloison aurait été déplacée. L'utilisateur peut aussi superposer les deux modèles d'ouvrages, en jouant sur la transparence, afin de visualiser les changements géométriquement.

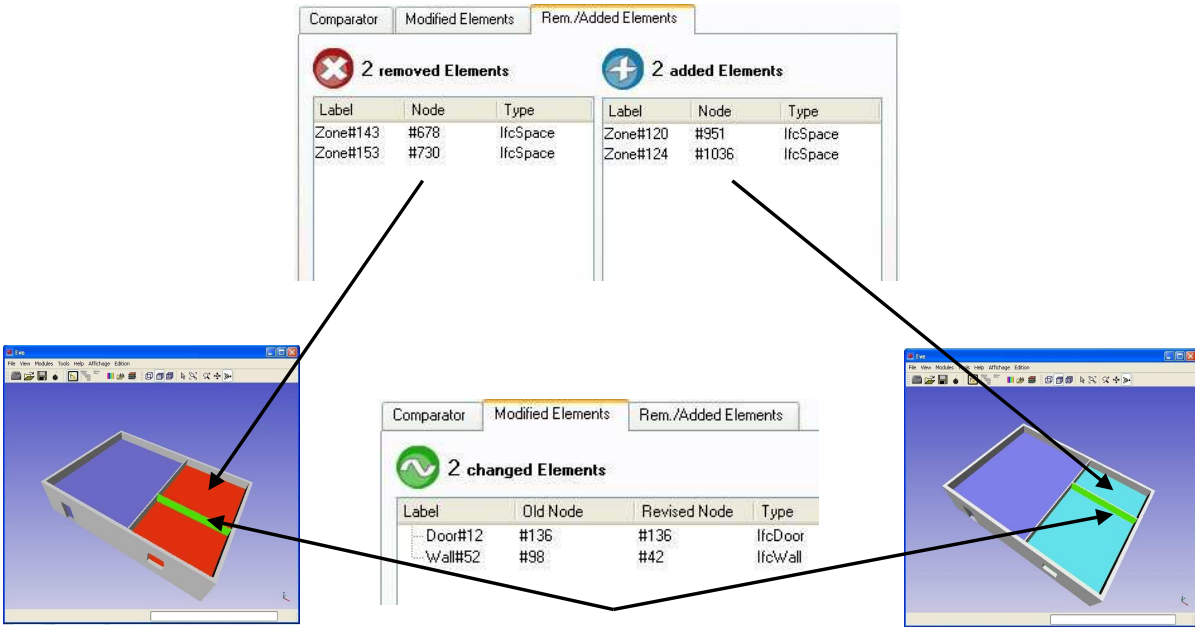


**FIGURE 4.11 :** Exemple d'utilisation de la comparaison sémantique dans la plateforme EVE 2.

La deuxième application résulte du partenariat franco-japonais SAKURA (cf. section 1.2.4

du premier chapitre). Ce projet vise à réaliser un démonstrateur pour la comparaison sémantique du côté français et un cas d'étude pour le plug-in IFC-Bridge de visualisation/édition sur AutoCAD©, du côté japonais. Dans ce partenariat, les japonais possèdent des jeux de données sur une poutre caisson et une poutre en T. Ils possèdent aussi un plug-in de visualisation sur AutoCAD© *PM2CAD* pour convertir un modèle d'ouvrage (Product Model) en données CAO pures. Le formalisme utilisé pour les modèles d'ouvrages n'est pas STEP-21 mais ifcXML [Nisbet05] (dérivé de la norme STEP-28 [ISO03]) étendu à IFC-Bridge. L'objectif consiste donc à proposer une chaîne de traitement pour comparer sémantiquement les deux modèles d'ouvrages et de visualiser les différences sur AutoCAD©. La figure 4.12 montre le mécanisme global du procédé.

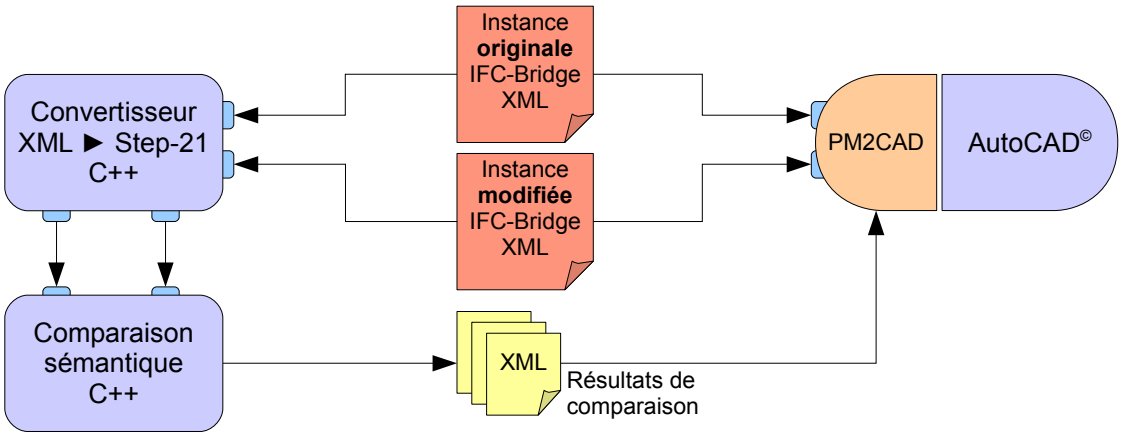


**FIGURE 4.12 :** Mécanisme global du démonstrateur dans le cadre du partenariat SAKURA.

A partir de deux modèles d'ouvrages, spécifié en ifcXML, la première étape consiste à les convertir au format STEP-21. Ensuite, les deux modèles sont comparés afin d'obtenir un rapport de comparaison. Le plug-in *PM2CAD* charge les deux modèles d'ouvrages et utilise le rapport de comparaison pour mettre surbrillance certaines pièces du modèle et ajouter des annotations sur le plan.

Dans le cadre de ce partenariat, nous avons été amenés à développer le convertisseur ifcXML vers STEP-21. La figure 4.13 détaille le procédé de conversion. Contrairement à la comparaison sémantique, nous n'avons pas utilisé ici la bibliothèque Expressik, car l'application de conversion est plus rapidement implémentable par une approche *Late-Binding* (cf. section 4.2.1) : les chaînes de caractères des balises XML permettent de créer les entités du modèle d'ouvrage en mémoire avant de les sauver dans un fichier STEP-21[4]. C'est pourquoi nous avons utilisé ST-Developer [STEP-Tools06b] et la bibliothèque C++ ROSE [STEP-Tools06a].

Les partenaires japonais ont réussi de leur côté à visualiser les différences sur AutoCAD©

[4]Expressik (version CSTB) n'autorise pas ce mécanisme qui créé des entités dont le type est mis en paramètre de la fonction, sous forme de chaîne de caractères.

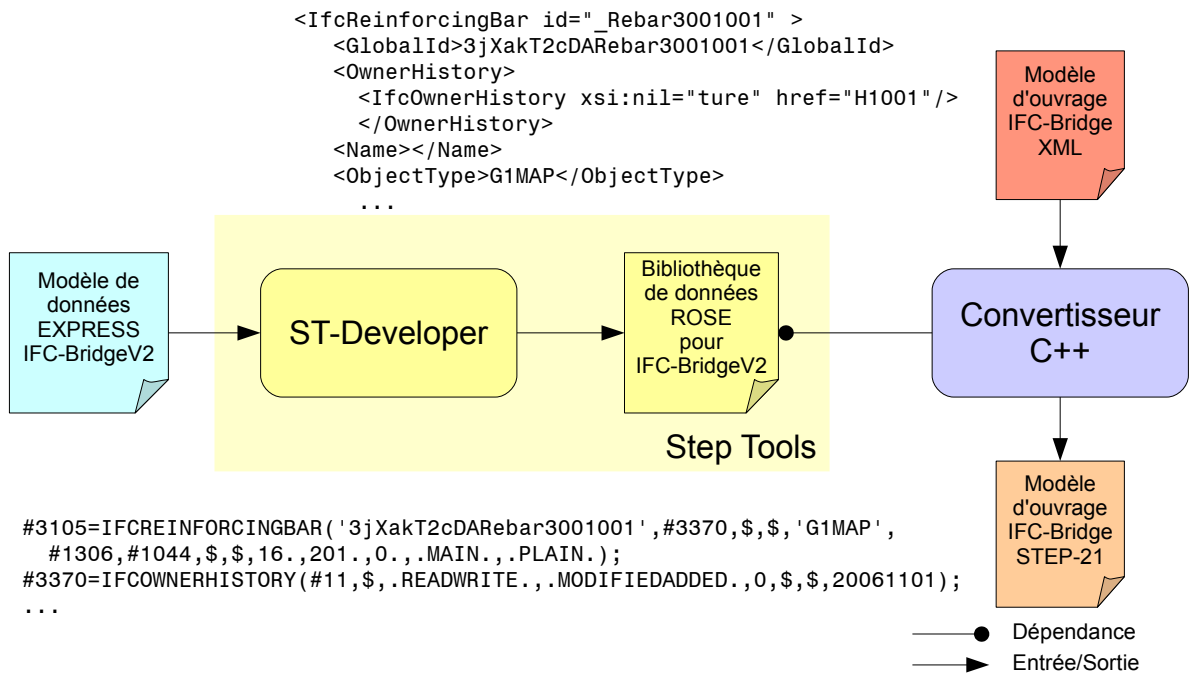


**FIGURE 4.13 :** Conversion de modèles d'ouvrages ifcXML au format STEP-21.

grâce au plug-in qui met en surbrillance les objets ajoutés/supprimés/modifiés et ajoute des annotations pour les objets n'ayant pas de représentation géométrique. La figure 4.14 regroupe plusieurs photos d'écran correspondant au procédé effectué sur une poutre en T.

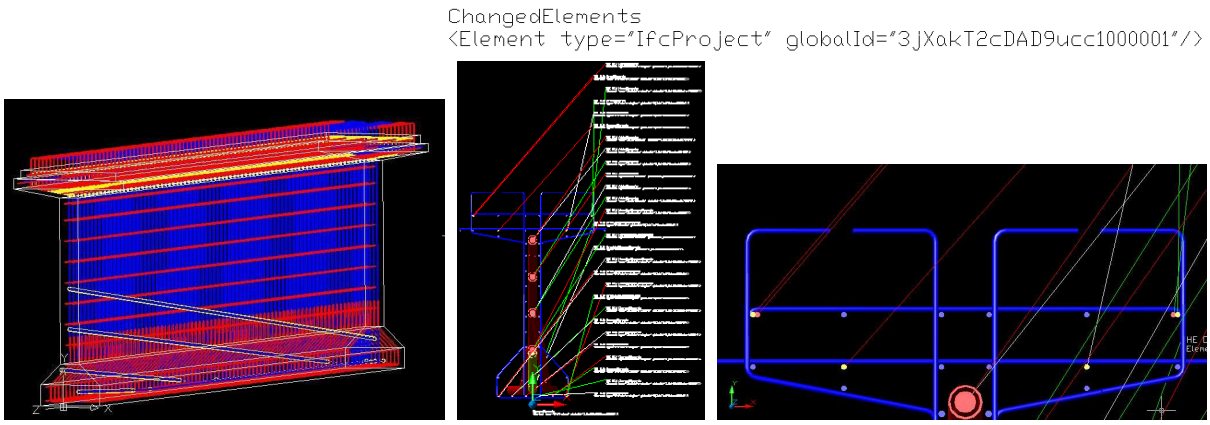


**FIGURE 4.14 :** Visualisation des résultats de comparaison sur AutoCAD©.

### 4.4.2 Interopérabilité avec des logiciels de CAO

La première intégration à des applications externes consiste à lier le comparateur sémantique avec des logiciels de CAO. Dans le cas du bâtiment, plusieurs logiciels supportent déjà les IFC :

Graphisoft ArchiCad©, Autodesk Revit©, Nemetschek Allplan©. La liaison avec le système de comparaison est alors aisée :

- **Intégration manuelle** : à chaque chargement/sauvegarde d'un modèle IFC, l'utilisateur lance une petite application basée sur le comparateur sémantique. Au chargement, l'instance originale est stockée. A la sauvegarde, l'instance modifiée est comparée avec l'instance originale, et un rapport est généré (document lisible au format Word, ou fichier XML). Ce processus est implémentable immédiatement mais il est manuel et donc plus contraignant pour l'utilisateur de l'application, cf. figure 4.15 (a).
- **Intégration par plug-ins** : les logiciels cités précédemment sont tous munis de fonctionnalités modulaires (*plug-ins*). En connaissant la façon de développer un module pour chaque logiciel, il est tout à fait possible de créer un plug-in qui appelle les routines de la bibliothèque de comparaison sémantique. Une fois le module installé, l'utilisateur exploite le logiciel CAO de manière transparente, le module s'occupant automatiquement d'intercepter les actions de chargement et de sauvegarde, pour effectuer une comparaison sémantique et générer le rapport. Cette méthode nécessite l'implémentation d'un module supplémentaire, mais simplifie le travail de l'utilisateur au quotidien, cf. figure 4.15 (b).

Cependant, il n'existe aucune application CAO ou de calcul de structure supportant le modèle IFC, ou son extension IFC-Bridge[5]. C'est pourquoi le CSTB et le SETRA travaillent sur l'interopérabilité autour du logiciel de calcul de structures PCP© (Pont Construit par Phase).

Le format natif de PCP n'est pas basé sur un modèle de données EXPRESS. Il s'agit d'un format propriétaire au format ***ASCII***. Afin d'assurer l'interopérabilité de ce logiciel, il convient d'implémenter un convertisseur entre le format PCP et le modèle IFC-Bridge. Tout d'abord un convertisseur du format PCP vers un modèle d'ouvrage IFC-Bridge permet d'utiliser le comparateur sémantique. En effet, à la création d'un modèle de pont sous PCP, une première conversion permet d'obtenir l'instance originale. Ensuite, à chaque sauvegarde, le convertisseur est lancé ainsi que le comparateur, générant ainsi le rapport de comparaison. Cela permet à PCP de se doter de fonctionnalités de suivi. Ce processus est en cours de tests sur les modèles d'ouvrages géométriques fournis par le SETRA (cf. figure 4.16).

Cependant, ce scénario n'est pas adapté à un environnement multi-applications de conception collaborative. Il manque la conversion dans l'autre sens : modèle IFC-Bridge vers le format PCP. En effet, si d'autres personnes modifient la MNC grâce à d'autres logiciels, les calculs de structure nécessitent parfois d'être relancés. De plus, la création de la MNC pourrait être effectuée par un logiciel de CAO et non pas un logiciel de calcul. Il s'agit alors d'un scénario favorable à la transformation de modèles [Katranuschkov06]. Nous rappelons que cette dernière se décompose en une étape de mapping direct (IFC-Bridge vers PCP) et une étape de mapping inverse (PCP vers IFC-Bridge). Cependant, le mapping inverse n'est pas une simple conversion PCP vers IFC-Bridge, mais une fusion de modèles prenant en compte les modifications effectuées par le logiciel PCP. L'étude de cette fusion dépasse le cadre de cette thèse, mais nous pensons qu'elle peut être réalisée plus facilement, grâce aux résultats du rapport de comparaison sémantique.

[5]IFC-Bridge n'a pas fini son processus d'intégration, c'est pour cela qu'il n'est pas encore implémenté par des logiciels commerciaux.

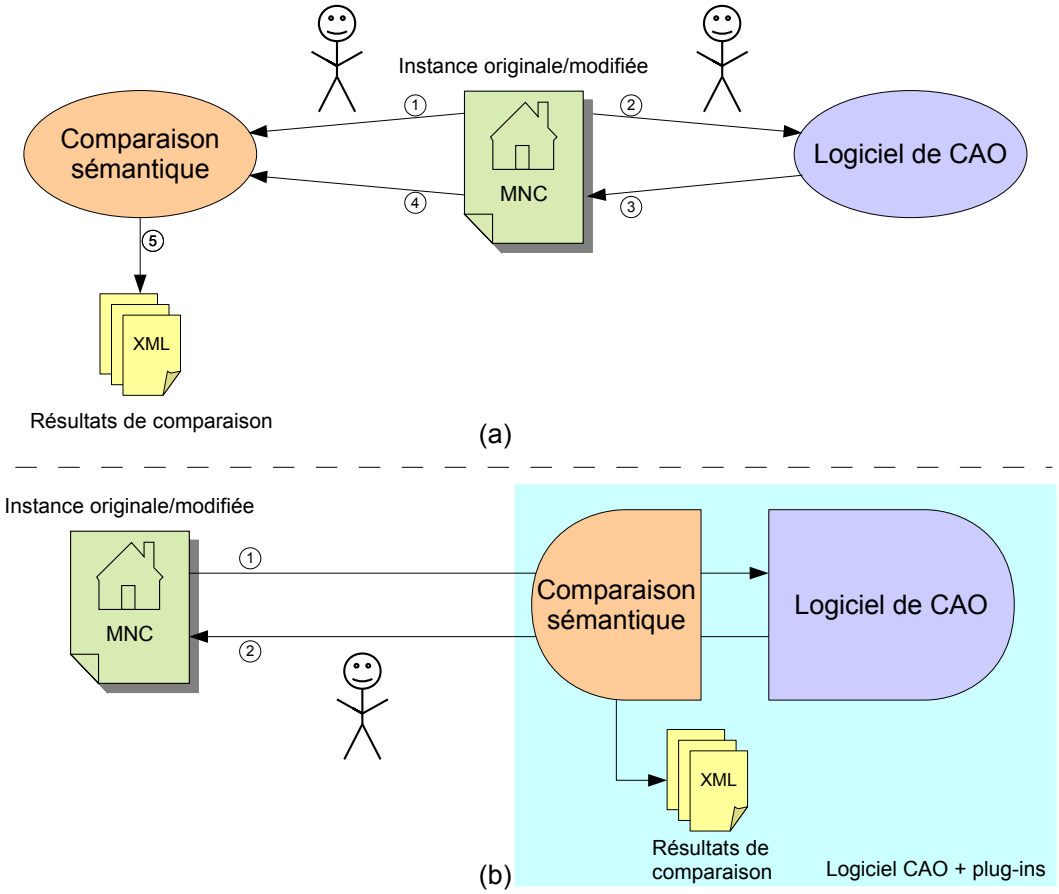


**FIGURE 4.15 :** Interopérabilité entre la comparaison sémantique et le logiciel de CAO. En haut, le processus est manuel, dans le sens où l'acteur de projet doit entrer le modèle original dans le système de comparaison (1), avant de l'éditer sur le logiciel de CAO (2,3), pour enfin le réentrer dans le comparateur (4,5). En bas, le processus est automatisé grâce à une intégration du comparateur par plug-in.

A travers ces deux parties, nous avons souhaité montrer plusieurs exemples d'intégration du système de comparaison sémantique afin d'assurer sa réutilisabilité. Cependant, d'après le processus-type de gestion de la MNC que nous avons choisi comme base au premier chapitre (cf. figure 1.26), si le suivi des modifications est assuré par le comparateur sémantique, la synchronisation documentaire n'a pas encore été abordée, alors qu'elle intervient durant la même phase de conception distribuée.

### 4.4.3 Liens avec la synchronisation documentaire

Dans le cadre de nos travaux, la documentation se limite à l'ensemble des documents *techniques*. Il s'agit d'une documentation permettant de justifier des choix de conception. Elle accompagne le modèle d'ouvrage et évolue ainsi en même temps que lui. Nous définissons alors un document comme le résultat d'une transformation de modèles, enrichie d'une mise en pages et d'annotations. De plus, le résultat de la transformation se limite ici aux structures d'arbres. Nous excluons ainsi les ***hypertextes*** qui peuvent être associés, via les ***hyperliens***, à des structures

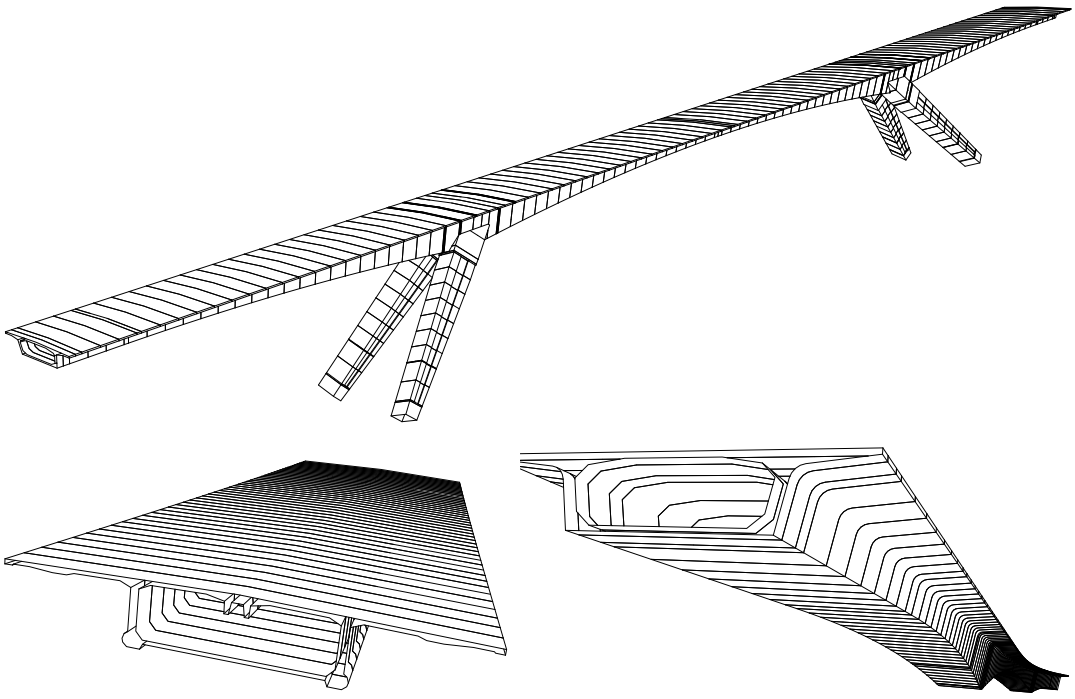

**FIGURE 4.16 :** Extraits des modèles d'ouvrages fournis par le SETRA. En haut, il s'agit du pont sur la Truyère à Garabit (Cantal (15), sur l'A75). En bas, à gauche, la section du tablier du pont de l'Iroise (Finistère (29)), à droite, une partie du tablier du pont de Genevilliers sur la Seine (Hauts-de-Seine (92)).

de graphes. En outre, un document technique contient les informations relatives à un seul corps de métier, il s'agit seulement d'une vue du modèle, résultat d'une extraction d'informations.

La synchronisation documentaire consiste alors à mettre à jour le document en y insérant une structure d'arbres modifiée par une nouvelle étape de conception, sans altérer si possible la mise en pages éventuellement effectuée par le concepteur ainsi que les annotations qu'il a ajoutées. La productivité est ainsi augmentée, car le concepteur ne doit pas recommencer sa mise en page et l'ajout de ses annotations.

Par exemple, un nouveau matériau fait son apparition dans le modèle d'ouvrages, et le document technique liste dans un tableau l'ensemble des matériaux utilisés. De plus, le concepteur a commenté un choix technique se situant à un autre endroit du document. Sans synchronisation, il doit ajouter des nouvelles lignes au tableau de matériaux, au risque d'en oublier, ou de faire une erreur. Il peut aussi utiliser un outil qui génère sa documentation technique à partir du modèle d'ouvrage (conformément à la définition faite précédemment), mais il doit remettre dans ce cas ses commentaires dans le nouveau document généré. En effet, nous constatons que les commentaires et justifications sont rarement intégrés au modèle d'ouvrage, mais font l'objet d'une entrée manuelle dans un document. Si le document est regénéré, ces informations sont perdues.

La synchronisation documentaire s'adapte ainsi aux pratiques professionnelles (entrées manuelles d'informations dans le document et non dans le modèle d'ouvrage), tout en proposant une amélioration de la productivité.

Le travail consiste alors à définir une chaîne de traitement permettant de :

1. Transformer un modèle d'ouvrage en une structure d'arbre contenant l'information à placer dans le document.
2. Enrichir la structure d'arbre d'une mise en page par défaut pour obtenir un document.
3. Synchroniser au besoin ce document avec la version antérieure déjà annotée, commentée, et mise en page.

La comparaison sémantique joue alors un rôle lors de la synchronisation des documents. Comme pour le démonstrateur du projet SAKURA (cf. section 4.4.1), les résultats de la comparaison sont réutilisés au moment de la synchronisation. Cette dernière va cependant plus loin que le démonstrateur SAKURA : il ne s'agit pas ici de superposer deux modèles en annotant les changements. Il s'agit d'extraire du nouveau document tous les changements du modèle de données, et de les insérer dans l'ancien, sans altérer l'information autour de cette insertion. Dans cette optique nous avons choisi pour ce travail des structures simples d'arbres. Synchroniser des structures plus complexes dépasse le cadre de cette thèse.

L'hypothèse principale rendant possible la synchronisation documentaire est la suivante : l'information générée par le modèle d'ouvrage est séparée de la mise en forme du document. Il s'agit d'un principe fondamental en gestion de contenu (cf. préface de [Goossens99]). C'est pourquoi nous avons souhaité décomposer la génération du document en deux étapes : extraction de l'information sous forme d'arbre structuré, et création d'un document mis en pages en utilisant un formalisme permettant de séparer le fond de la forme, même au sein du document final.

Pour implémenter ce système, nous avons choisi des technologies avec un souci de réutilisabilité. Ainsi, la structure d'arbres est encapsulée dans un fichier XML, et la transformation de la structure en un document mis en pages utilise la technologie XSLT [Clark99], permettant de transformer une structure XML en une autre. Enfin, nous avons choisi de développer la synchronisation documentaire sur le logiciel Microsoft Word©, et son format XML WordML [Aitken03]. Ce dernier supporte en effet le principe de séparation, grâce à une distinction entre les balises de l'information pure et les balises de mise en page. De plus, la solution Microsoft Office© reste globalement très utilisée dans tous les secteurs industriels. La figure 4.17 illustre la chaîne globale de traitement. Elle suppose l'emploi d'une application de visualisation/édition de modèle d'ouvrage, qui exploite la comparaison sémantique et la génération d'un fichier XML de la structure en arbre. Dans nos travaux, nous avons choisi la plateforme EVE 2 munie des modules IFC-Bridge et de comparaison sémantique (cf. section 4.4.1).

La synchronisation documentaire se décompose selon deux phases. La première correspond à la génération du document à partir du modèle d'ouvrage. Elle suit trois étapes :

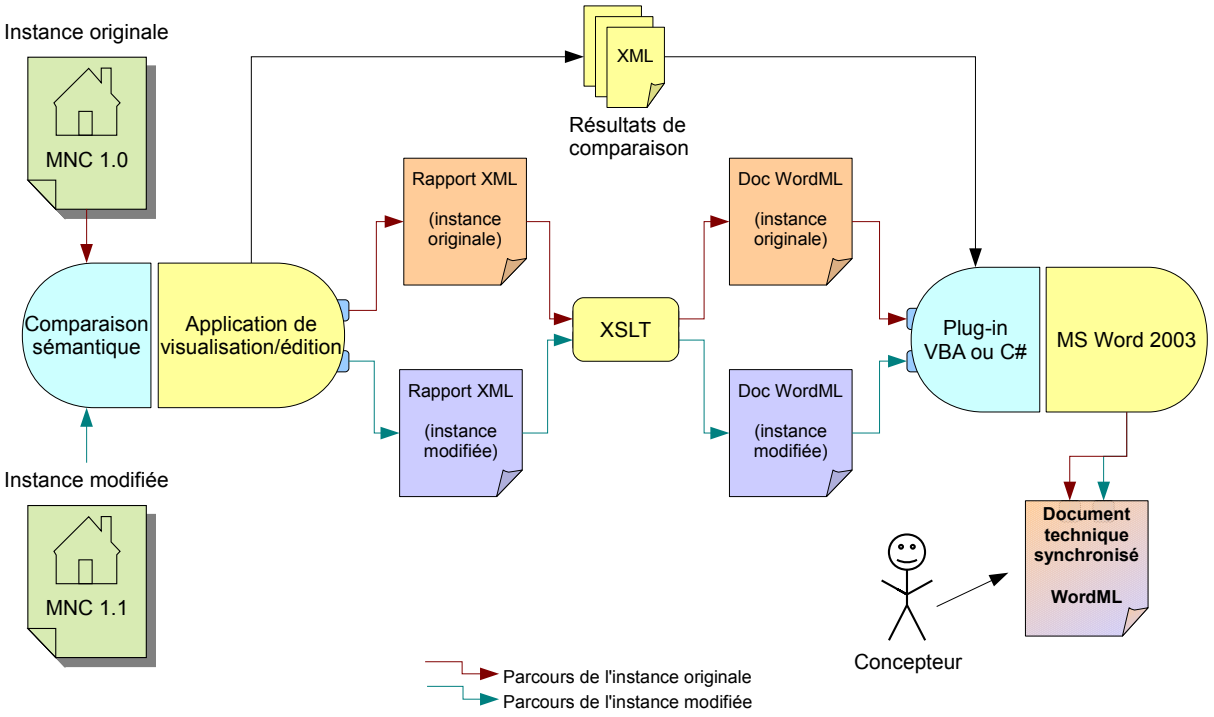


**FIGURE 4.17 :** Diagramme global de la synchronisation documentaire.

1. Création de la structure XML à partir du modèle d'ouvrage. Dans le cadre de nos travaux, nous n'avons pas développé de mécanisme générique permettant d'extraire l'information d'un modèle STEP pour créer une structure XML. Nous avons néanmoins implémenté des fonctions d'extraction et d'export XML, grâce à la bibliothèque Xerces [Dubchak03], pour une vue spécifique d'un modèle de données particulier.
2. Transformation du fichier XML en un document WordML. Grâce à une feuille XSLT, qui suit aussi le formalisme XML, un fichier XML est transformable en un autre fichier XML. Le but est ici de fournir une mise en pages par défaut, correspondant à la charte graphique d'une société, ou d'un projet.
3. Modification du document par le concepteur. Ce dernier annote, commente, et ajuste la mise en page. L'annotation et les commentaires sont effectués dans des zones prévues à cet effet. L'ajustement de la mise en pages consiste à n'effectuer que des changements locaux (modification du style d'une phrase, centrer un paragraphe, etc.).

La deuxième phase intervient lorsque le modèle d'ouvrage a été modifié. Les étapes suivantes s'enchaînent :

1. Comparaison sémantique des deux modèles d'ouvrage. Un résultat de comparaison est généré au format XML.
2. Génération d'un nouveau document WordML, suivant les étapes 1. et 2. de la première phase.
3. Synchronisation du document original avec le nouveau. Grâce à une macro VBA ou un plug-in C#, le document original, enrichi de ses annotations, est synchronisé, grâce au

nouveau document WordML et au rapport XML de comparaison. Les éléments éventuellement ajoutés dans le nouveau document possèdent la mise en page par défaut.

Microsoft Word© permet d'utiliser le mécanisme des révisions : les modifications ne sont pas appliquées directement, mais suggérées en marge du document, et sont donc soumises à l'accord du concepteur. L'utilisation de ce logiciel (et du format WordML) n'est cependant pas obligatoire. En effet, l'architecture que nous avons choisie autorise l'adaptation à d'autres traitements de texte. Il suffit pour cela de recréer la feuille de transformation XSLT — une connaissance de ce langage est cependant nécessaire — tâche à affecter au constructeur d'application.

## 4.5 Conclusion

Ce chapitre a permis d'abord de démontrer la possibilité d'implémenter les méthodes de comparaison structurelle et sémantique de la MNC. La solution proposée semble un bon compromis entre performance (approche *Early-Binding*) et facilité d'implémentation (génération automatique de la biliothèque C++). Le recours à un constructeur d'application demeure néanmoins nécessaire afin de créer le système adapté à un projet, à une équipe de projet, ou encore à un corps de métier particulier, selon l'utilisation du comparateur.

Ensuite, plusieurs exemples ont détaillé des cas d'utilisation du comparateur sémantique. Ainsi, la productivité est-elle potentiellement améliorable grâce à ces outils, tout en respectant les pratiques profesionnelles de la construction.

Par rapport au processus-type défini au premier chapitre (figure 1.26), nous avons donc proposé une méthodologie concernant le suivi des modifications (à travers la comparaison sémantique) et la synchronisation documentaire. La fusion des modèles n'a pas été abordée. Cette question mériterait en effet une analyse qui dépasse la cadre de nos travaux. Néanmoins, il convient au moins d'étudier les perspectives de cette méthodologie de comparaison sémantique, par rapport à la fusion des modèles d'ouvrage, point essentiel pour réaliser la synchronisation de la MNC.

## Résumé

Ce chapitre englobe l'ensemble des implémentations informatiques de nos travaux, à travers trois axes : le système de comparaison structurelle, l'analyse sémantiques, et l'intégration au sein d'applications existantes. Notre choix s'est porté sur le C++ car son adoption est très large dans tous les domaines industriels et reste très bien placé en termes de performance. Cependant, nos travaux pourraient très bien s'implémenter dans n'importe avec langage orienté objet. Dans tous les cas, l'implémentation de la comparaison structurelle nécessite une bibliothèque pour gérer les modèles de données EXPRESS, en particulier les IFC et IFC-BRIDGE. Il existe deux types d'implémentations : *early-binding* et *late-binding*. La première correspond à des bibliothèques compilées pour un modèle de données particulier pour des performances optimisées, alors que le deuxième est adaptable à n'importe quel modèle de données, au détriment des performances. Nous avons choisi la première approche, via l'utilisation du projet Open-Source *Expressik*, écrit en Java. A partir de la spécification EXPRESS d'un modèle de données (comme IFC ou IFC-BRIDGE), le programme génère une bibliothèque C++ capable de charger, modifier et sauver n'importe quel modèle d'ouvrage, instance du modèle de données précédent. Nous avons alors choisi une approche similaire pour la comparaison structurelle : à partir du modèle de données EXPRESS, il s'agit de générer le code source d'une application C++ capable de comparer deux modèles d'ouvrages. Un tel programme a besoin de paramètres supplémentaires : la structure statique (squelette) du code C++ en sortie et l'ensemble des informations structurelles spécifiques au modèle de données (racine de parcours, entités identifiables). Pour le premier, deux variantes ont été envisagées : les fichiers squelettes (template) munis de tags ou l'utilisation d'un paquetage Java capable de générer n'importe quel code C++. Concernant le deuxième paramètre, ces informations sont encapsulées dans un fichier XML nommé *assistant structurel*. Pour le modèle IFC-BRIDGE, il indique ainsi que l'entité racine est l'instance de `IfcProject` et que les entités sont identifiables si et seulement si elles instancient `IfcRoot`. De plus, le code C++ généré utilise la bibliothèque créée par Expressik et son architecture utilise massivement le modèle de conception *Visiteur* [Gamma94]. Les résultats de comparaison sont quant à eux encapsulés dans un autre fichier XML. Concernant l'analyse sémantique, nous avons choisi d'étendre les capacités de l'assistant structurel par de nouveaux tags, créant ainsi un assistant d'extraction d'information et un assistant d'équivalents sémantiques. Nous remarquons que l'utilisation de techniques de hachage, permettant d'accélérer les performances du comparateur, nécessite une adaptation lorsque des équivalents sémantiques sont définis. La dernière partie de ce chapitre est consacrée à l'intégration de nos travaux au sein de systèmes de gestion de la MNC. Ces intégrations sont d'ordre visuels : utilisation de la plateforme EVE 2, visualisation sur AutoCAD dans le cadre du partenariat SAKURA. Elles peuvent aussi concerner l'interopérabilité avec des logiciels de CAO ou de calculs de ponts. Enfin, pour la synchronisation documentaire, essentielle au niveau du processus-type défini au premier chapitre, un framework est proposé pour réutiliser les résultats de comparaison au sein d'un traitement de texte supportant le mécanisme de révisions, comme MS Word 2003.

## Abstract

This chapter encapsulates all implementations of our work through three main axes : structural comparison system, semantic analysis, integration into existing software. We choose C++ language because of its large adoption by every industrial field and its good performance. However our work could be implemented using any object-oriented language. Anyway implementation of structural comparison needs a library to handle EXPRESS data models, especially IFC and IFC-BRIDGE. Two kinds of implementations coexist : *early-binding* and *late-binding*. The first one relates to compiled libraries for a specific data model with optimized performance, whereas the second one is a generic solution for every data model to the expense of less performance. We choose the first approach by using the Open Source Java project : *Expressik*. From EXPRESS specifications of a data model (for example IFC and IFC-BRIDGE), this tool generates a C++ library able to load, edit, and save any product model, which instantiates this data model. Therefore we choose a similar approach for structural comparison : from an EXPRESS data model, the goal is to generate the source code of a C++ tool able to compare two product models. Such a program needs complementary parameters : static structure (template) of output C++ code and all specific information from the data model (root entity, identifiable entities). Concerning the first item two options can be implemented : template files with tags or the use of a Java package able to generate any C++ code. Concerning the second parameter this information is encapsulated into a XML file called *structural helper*. Applied on IFC-BRIDGE model, it indicates the root entity is the instance of `IfcProject` and identifiable entities inherit from `IfcRoot`. Furthermore generated C++ code uses a library created by Expressik and its architecture is based on *Visitor* design pattern [Gamma94]. Comparison results are then encapsulated into another XML file. Concerning semantic analysis we choose to extend the structural helper with new tags, in order to create an *extraction helper* and an *equivalence helper*. We notice that using hashing mechanisms increases system's performance but needs to be adapted when the system exploits semantic equivalences. Last part from this chapter is dedicated to integration of our work within systems of DMUC handling. These integrations could be visual : use of EVE 2 platform, AutoCAD plug-in for the SAKURA partnership. They could also refer to interoperability with CAD software and engineer software. Last documents' synchronizing is essential for the DMUC framework (defined at the first chapter). Another framework is proposed to reuse comparison results within word processing software, if it supports revision functionalities, like MS Word 2003.

# Chapitre 5

# Évaluation et perspectives de la comparaison sémantique

## 5.1 Introduction

Après avoir mené un premier examen de nos travaux par rapport à sa faisabilité et sa capacité à s'adapter au sein d'environnements logiciels existants, ce chapitre a pour objectif de montrer les limites actuelles de nos travaux, dans certaines phases du cycle de vie d'un ouvrage et quelles seraient les perspectives et les pistes d'analyse, permettant de trouver des solutions pour contourner ces limites.

Pour cela, il convient d'aborder une problématique concernant la synchronisation de MNC, à savoir la fusion des modèles d'ouvrages en préparation à une réunion de synthèse/co-conception (section 5.2). Afin de de répondre aux problèmes posés, une analyse des relations entre la MNC, les méthodes de comparaison sémantiques et les activités de conception est mené (section 5.3). Enfin l'objectif consiste à étudier la portée de nos travaux au-delà de la conception d'un projet, c'est-à-dire lors de l'éxécution des travaux et la maintenance de l'ouvrage (section 5.4).

## 5.2 Problématiques relatives à la fusion des modèles d'ouvrages

### 5.2.1 Difficultés d'automatisation lors de la synthèse

Reprenons le schéma global du processus-type utilisé dans nos travaux. Dans un premier temps, la figure 5.1 détaille un exemple du suivi des modifications, par comparaisons sémantiques, avant d'aboutir à un point de synthèse.

Au niveau du point de synthèse, la fusion consiste à obtenir une nouvelle MNC commune à partir des modèles d'ouvrages locaux. La nouvelle MNC commune intègre les étapes de conception des deux versions locales. De plus, cette transformation semble en lien avec la comparaison sémantique car les résultats de comparaison pourraient aider l'élaboration de la fusion. Cependant, sa mise en oeuvre pose plusieurs questions :

- Existe-t-il toujours une fusion à partir de deux versions locales ? Est-elle unique ?

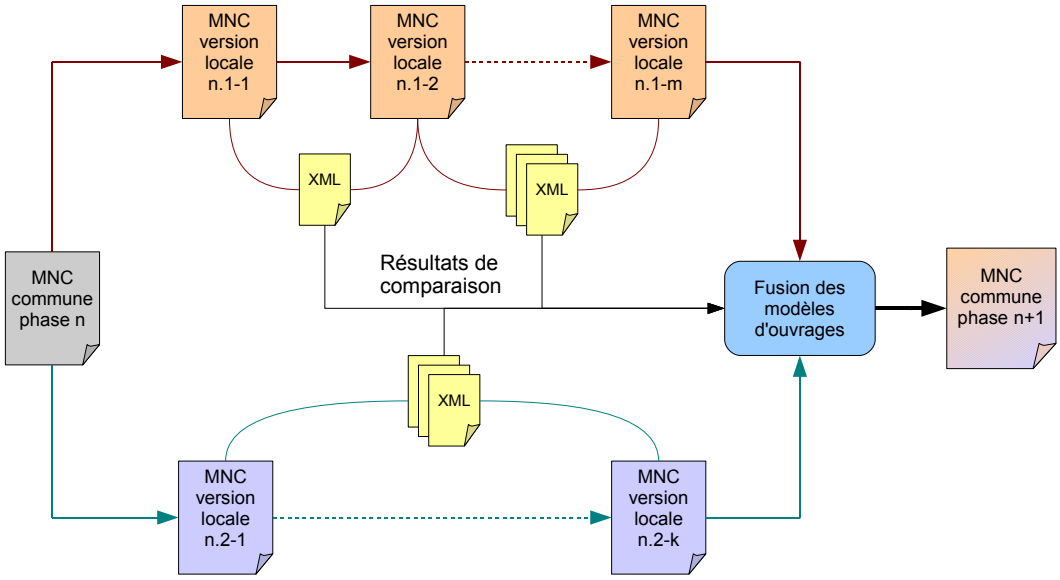


**FIGURE 5.1 :** Application du processus-type avec deux versions locales.

- Si plusieurs fusions sont possibles, existe-t-il un algorithme permettant de trouver une solution ?
- Comment effectuer la fusion ? Car s'offrent alors plusieurs alternatives : mise à jour d'une version locale à partir de l'autre version locale (sous condition de savoir quelle version sert de référence), intégration des modifications des deux versions locales à la version commune, issue du point de synthèse/co-conception précédent, etc.

La dernière question soulève la problème de la séquentialité des étapes de conception, au sein d'un travail collaboratif. Nous reviendrons sur cette problématique à la section 5.2.3. Les deux premières questions remettent en cause l'existence et la calculabilité d'une solution pour la fusion. De manière générale et formelle, il n'est donné aucune garantie d'existence de solution. Et même si cette dernière existe, est-elle sémantiquement acceptable ? En effet, supposons le scénario suivant, sur un modèle d'ouvrage IFC :

Lors du premier point de synthèse, un hall est pourvue d'une colonne. Dans le paradigme IFC, cela signifie qu'il existe une instance de l'entité `IfcSpace`, ayant pour nom « hallA », liée à une instance de l'entité `IfcColumn`, nommée « colonne1 », grâce à une instance de l'entité de relation `IfcRelContainedInSpatialStructure`. De son côté, l'architecte modifie cet espace sur une version locale. Il supprime notamment la colonne qui coupe l'espace du hall et ajoute un éclairage au plafond. D'un autre côté, un ingénieur effectue des calculs de structure et dimensionne la colonne en ajoutant des propriétés de volume et de section. Il ajoute aussi une poutre qu'il estime nécessaire. La figure 5.2 applique cet exemple dans le paradigme IFC et utilise les propriétés d'extraction d'informations du comparateur sémantique (cf. chapitre 4., section 3.4.1).

Au moment de la synthèse, la fusion des deux modèles d'ouvrages posent plusieurs problèmes :

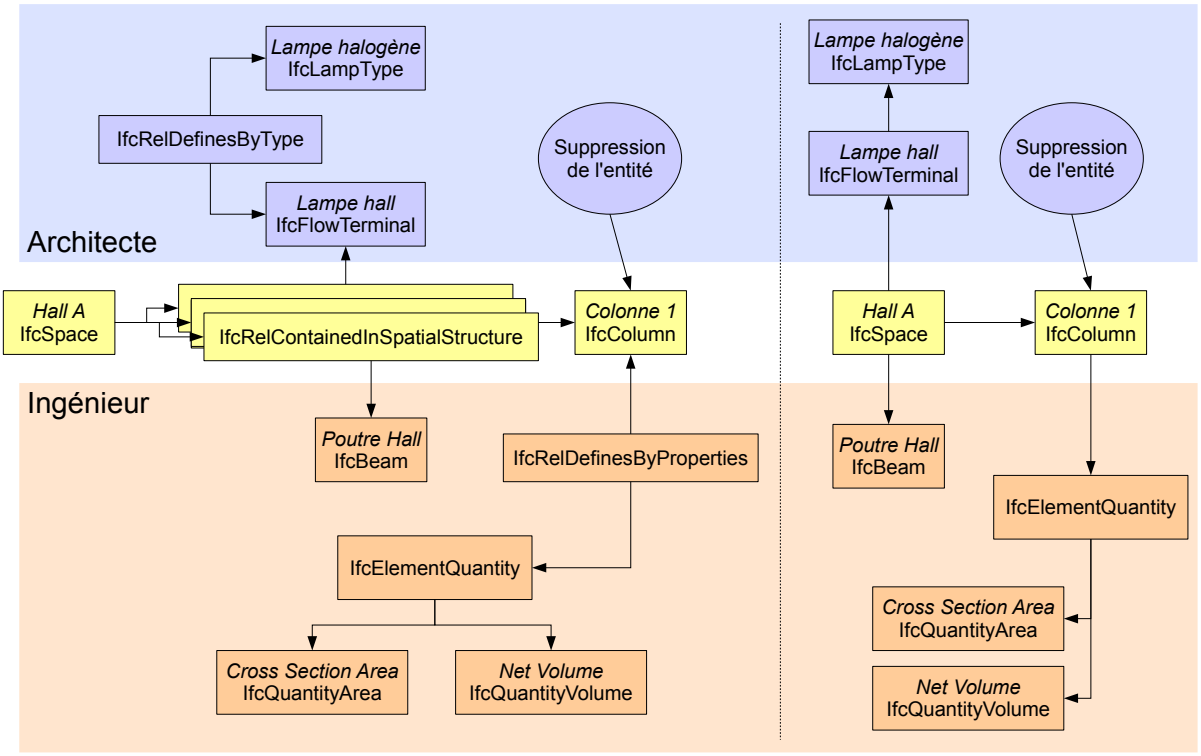


**FIGURE 5.2 :** Exemple de modifications simultanées sur la MNC. A gauche, il s'agit de la modélisation IFC originale. A droite, le modèle a été simplifié selon les règles d'extraction d'information de l'assistant sémantique : suppression des entités de relation.

- L'instance « colonne1 » a été supprimée d'une part et modifiée d'autre part. Il s'agit d'une incohérence, aucune solution ne peut prendre en compte ces deux changements[1].
- L'instance « hallA » a été modifiée de part et d'autre. Cependant, ces modifications peuvent être fusionnées : l'éclairage et la poutre sont alors ajoutés à la nouvelle MNC commune. Structurellement, la solution existe, et peut même être facilement calculée automatiquement (en réponse à la deuxième question du paragraphe précédent). Cependant, si la poutre entre en collision avec l'éclairage, la MNC n'est plus valide géométriquement.
- L'utilisation d'un comparateur sémantique avec extraction d'informations pose un problème à la fusion : dans la nouvelle MNC commune, les entités de relation sont nécessaires, quelles instances convient-il d'implémenter ? Réutilisation des entités d'une des versions ou bien création dynamiquement de nouvelles relations ?

Le premier point montre que le problème de fusion n'a pas toujours de solution exacte dans la mesure où elle ne peut suivre les modifications des deux versions. Il existe des solutions approchées qui sélectionnent certaines modifications sans conséquence sur la cohérence du modèle. Mais il persiste un conflit entre deux versions qui ne peut être résolu sans intervention

[1]Comme il a été montré au chapitre 3, section 3.4.2, une entité identifiable peut être supprimée sans disparaître structurellement du modèle d'ouvrage décrit en IFC. Cela signifie qu'une fusion serait quand même possible. La colonne « fusionnée » aurait ses propriétés de volume et section mais serait considérée comme supprimée. Cependant, tous les logiciels de CAO n'exploitent pas cette fonctionnalité ; de plus, si l'architecte avait effectué des modifications de la colonne au niveau des mêmes attributs que ceux issus des modifications de l'ingénieur, l'incohérence persisterait.

de l'homme. Concernant le deuxième point, le système de fusion pourrait se doter de modules de vérification sémantique. Dans cet exemple, il s'agit de vérifier la cohérence géométrique. Les travaux de [Chen04] suggèrent un environnement collaboratif pour répondre à ce problème et [Yabuki04] propose un ***système multi-agents*** en tant qu'aide à la conception, notamment pour la vérification de collisions géométriques. Cependant, cela nous ramène au premier point concernant la détection d'incohérences et n'apporte pas de solutions de résolution. Enfin, le troisième point pose un problème de mise en oeuvre de la fusion, lorsque la comparaison utilise un assistant sémantique. En liaison avec la troisième question posée au début de cette section, ce point sera aussi abordé à la section 5.2.3.

Cette réflexion nous conforte dans l'hypothèse que la fusion n'est pas un processus automatisable. Néanmoins, l'utilisation des résultats de comparaison sémantique permet de détecter des incohérences au niveau de la MNC. Un outil informatique pour la fusion des modèles d'ouvrages, supervisé par le coordinateur de projet, rendrait ce dernier plus réactif et l'aiderait à préparer une réunion de synthèse. Cependant, le scénario précédent ne semble pas retracer toute la complexité du processus de conception distribuée, et les risques de divergence de conception peuvent rendre la tâche de fusion des modèles très complexe, même avec un outil informatique.

### 5.2.2 Risques de divergence en conception distribuée

Dans le domaine du partage des connaissances, [Díaz05] définit la divergence comme étant « un conflit cognitif et se traduit par l'apparition d'alternatives, d'arguments et de points de vue différents autour d'un concept ou d'un sujet d'intérêt. ». Cette définition semble assez adaptée dans la conception architecturale. Cette dernière réunit en effet des acteurs aux sensibilités et aux objectifs différents, citons seulement l'opposition traditionnelle entre architecte et ingénieur (rappelée par [Lemoine04] et [CNISF05]), et favorise en conception distribuée l'apparition d'alternatives.

Le scénario précédent incluait deux versions locales d'un modèle d'ouvrage et pour chacune d'elle, deux modifications élémentaires. Si le nombre de versions et de modifications élémentaires augmente, le risque de divergence semble croître très rapidement : soit $n$ le nombre de versions locales entre deux points de synchronisation. Sur chaque version locale, $m_i$ représente le nombre de résultats de comparaison, et pour chaque résultat de comparaison, $p_{i,j}$ modifications ont été effectuées. La figure 5.3 illustre ce cas général.

A la synthèse, le nombre de combinaisons possibles et potentiellement sources de divergence vaut :

$$\prod_{i=1}^{n} \sum_{j=1}^{m_i} p_{i,j}$$

Il s'agit là d'une limite du processus-type choisi pour nos travaux. La liberté d'élaborer des versions locales pour chaque groupe d'acteur est contrebalancée par une difficulté de synchronisation au niveau du point de synthèse, et cela y compris avec un suivi des modifications. Ce

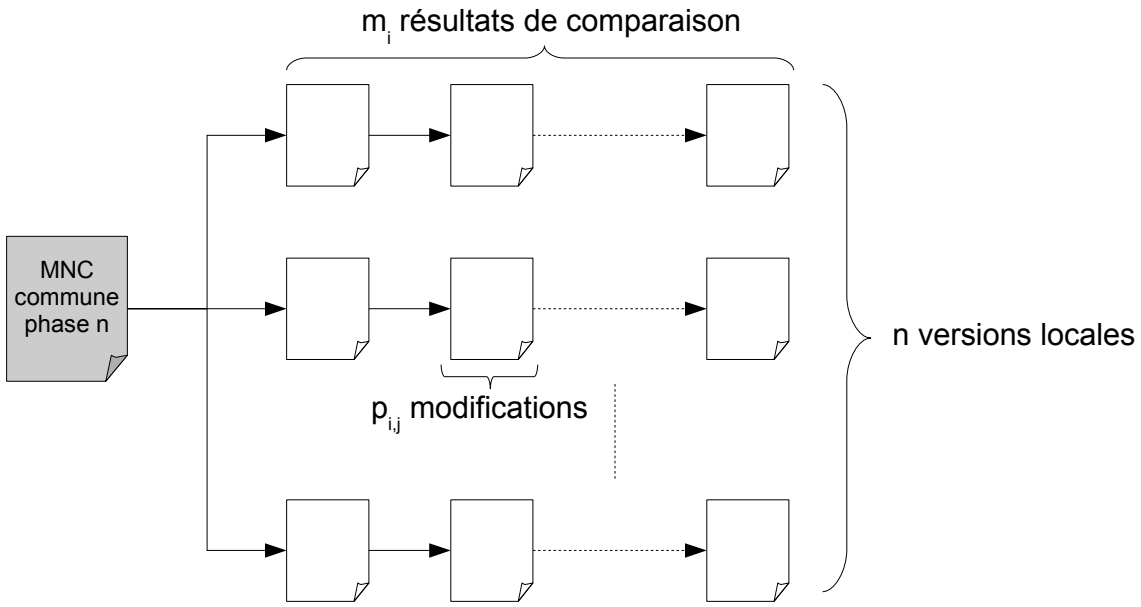


**FIGURE 5.3 :** Divergence des modèles d'ouvrage en conception distribuée.

dernier permet seulement de détecter les incohérences. Même muni d'un outil informatique, le coordinateur de projet peut perdre du temps à comprendre les incohérences détectées par le système, afin de préparer une réunion de synthèse.

Avant de proposer des perspectives, à la section 5.3, dans l'objectif de résoudre les problématiques précédentes, un dernier point, concernant la mise en oeuvre de la fusion de modèles d'ouvrages, mérite d'être abordé. Il s'agit d'analyser comment les modifications de la MNC sont fusionnées, sur quel modèle d'ouvrage. Pour cela, il convient d'étudier le processus de versionnement de la MNC, dans le cas où sa gestion suit le processus-type choisi dans nos travaux.

### 5.2.3 Liens entre la fusion de modèles d'ouvrages et le versionnement de la MNC

Dans cette partie ne seront pas abordées les notions de versionnement d'objets, concept introduit au chapitre 1, section 1.3.2. En effet, les bases de données supportant de telles fonctionnalités contiennent déjà des mécanismes de comparaison et de synchronisation. De plus, les règles de propagation (cf. [Urtado98]) semblent permettre de configurer le système afin d'obtenir un outil aussi flexible que notre comparateur sémantique, en particulier pour la comparaison sélective et l'extraction d'information.

Il s'agit donc d'adapter les techniques de versionnement usuelles, utilisées pour des projets informatiques, comme CVS [Cederqvist06] et Subversion (aussi appelé SVN [Pilato04]). Même si nous avons plutôt étudié SVN (plus récent) dans le cadre de cette thèse, les mécanismes généraux s'appliquent aussi à CVS. Il convient d'abord d'analyser le fonctionnement de Subversion pour l'utilisateur.

Vis-à-vis d'un projet versionné, la première étape pour l'utilisateur consiste à télécharger

(*checkout*) du serveur les documents (pour un projet informatique, il s'agit des sources et éventuellement des fichiers de configuration) correspondant à la dernière version centrale du projet. L'utilisateur possède alors une version locale sur son système. Il peut à présent travailler sur ces fichiers. Tant qu'il effectue des modifications sur sa version locale, personne n'a accès à ces évolutions.

Dans une première variante, deux acteurs travaillent sur le même projet, et sur le même fichier. L'acteur A a modifié le fichier localement. Il met d'abord à jour (*update*) son projet par rapport aux travail des autres. Aucune modification sur son fichier n'a été effectuée. Il enregistre sur le serveur son fichier modifié (*commit*). Pendant ce temps, l'acteur B a aussi modifié le même fichier. A son tour, il met à jour son projet, notamment les modifications de l'acteur A. Le système Subversion utilise un algorithme de comparaison de fichier ASCII LCS [Myers86] (déjà abordé au chapitre 1, cf. figure 1.13) et tente de fusionner les deux documents. Cependant, si certaines lignes se contredisent, l'utilisateur est notifié et doit résoudre lui-même le conflit avant d'enregistrer la nouvelle version du fichier auprès du serveur. La figure 5.4 illustre le procédé.

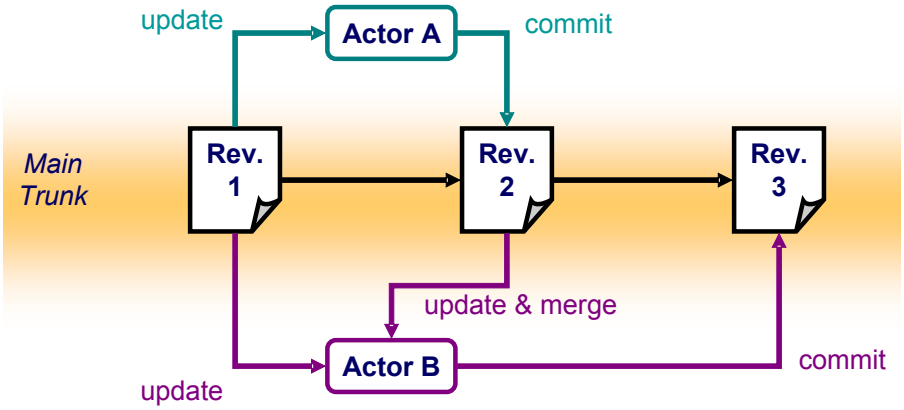


**FIGURE 5.4 :** Utilisation d'un système de versionnement sans branches.

Une première remarque sur cet exemple concerne la séquentialité du processus. Même s'il s'agit d'un environnement collaboratif à concurrence additive voire totale (d'après la classification de [Hanser03], inspiré de [Ellis94]), le document évolue de manière séquentielle. De plus, l'acteur responsable de la synthèse des modifications n'est pas clairement défini. Si, dans l'exemple, l'acteur B avait enregistré ses modifications avant l'acteur A, ce dernier aurait été responsable de cette fusion[2]. Subversion permet de limiter l'accès en écriture selon les acteurs, mais cela affaiblit l'aspect collaboratif du projet.

Une deuxième variante utilise le mécanisme des *branches*. Cette fois, l'acteur B créé une branche sur le serveur. Cela signifie que ce dernier copie le fichier et qu'un nouveau versionnement y est effectué en parallèle. Aussi l'acteur B n'a-t-il pas à se soucier des modifications de l'acteur A tant qu'il sont sur des branches différentes. Ces deux acteurs enregistrent ainsi

[2]Avec Subversion et CVS, il existe un mécanisme de blocage d'accès qui empêche deux acteurs d'enregitrer ses modifications en même temps.

leurs modifications. De plus tout le monde peut accéder aux diverses branches. Cela permet de développer une alternative, sans entraver le développement normal du document sur le tronc commun. Cependant, si cette alternative est acceptable, les modifications de la branche nécessitent d'être fusionnées sur le tronc commun. N'importe quel acteur, assez courageux, peut effectuer cette synchronisation. La figure 5.5 illustre cette deuxième variante.

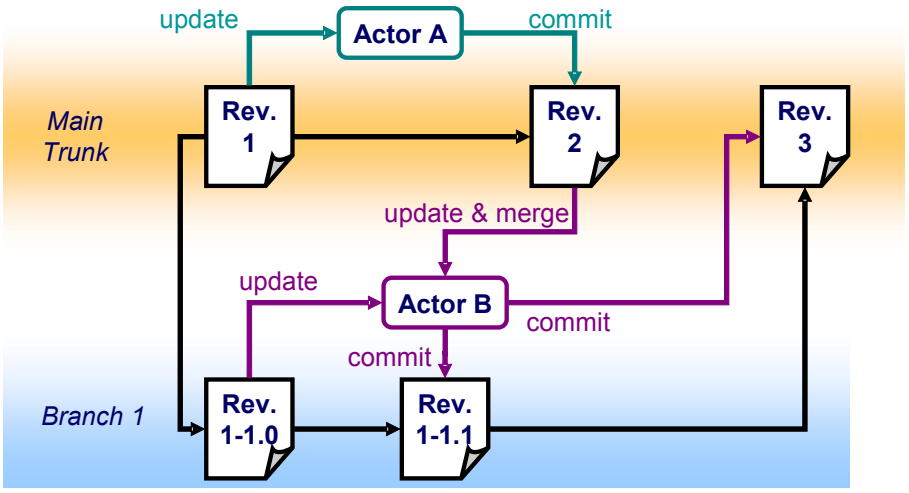


FIGURE 5.5 : Utilisation d'une branche pour le versionnement d'un fichier.

Dans ce scénario, l'évolution du document n'est plus séquentielle, grâce aux mécanismes des branches. Néanmoins, il soulève le même problème de la fusion des documents, sa mise oeuvre, par qui, etc. Ce procédé fonctionne correctement dans les projets informatiques car les acteurs pratiquent le même métier, chacun est légitime d'effectuer cette fusion. Pour les projets architecturaux et de génie civil, la situation diffère à cause de l'hétérogénéité des acteurs en jeu. La prise en compte des acteurs dépasse le cadre de notre analyse. Cependant, les précédents exemples montrent les limites d'une analyse portée uniquement sur l'artefact de projet. Par conséquent, l'objectif consiste à étudier la portée de la comparaison sémantique — et plus globalement celle de nos travaux — sur les activités de projet, en conception.

## 5.3 Relations entre la comparaison sémantique et les activités de conception

### 5.3.1 Synchronisation de la MNC suivant une activité prédictive

La conception suit des processus structurés, illustrés par différents modèles. Dans le cadre de nos travaux, nous avons eu besoin d'identifier un processus-type, même s'il reste très général et très macroscopique. Il s'agit donc d'une activité prédictive, en ce sens où elle paraît trop complexe pour être conduite de façon totalement intuitive [Laaroussi07b]. Le but de cette structuration consiste à prévoir un résultat [Raichvarg97]. Cela peut sembler paradoxal par rapport à la nature même de la conception : elle vise en effet à *inventer* un ouvrage unique, ce dernier n'est donc pas connu d'avance.

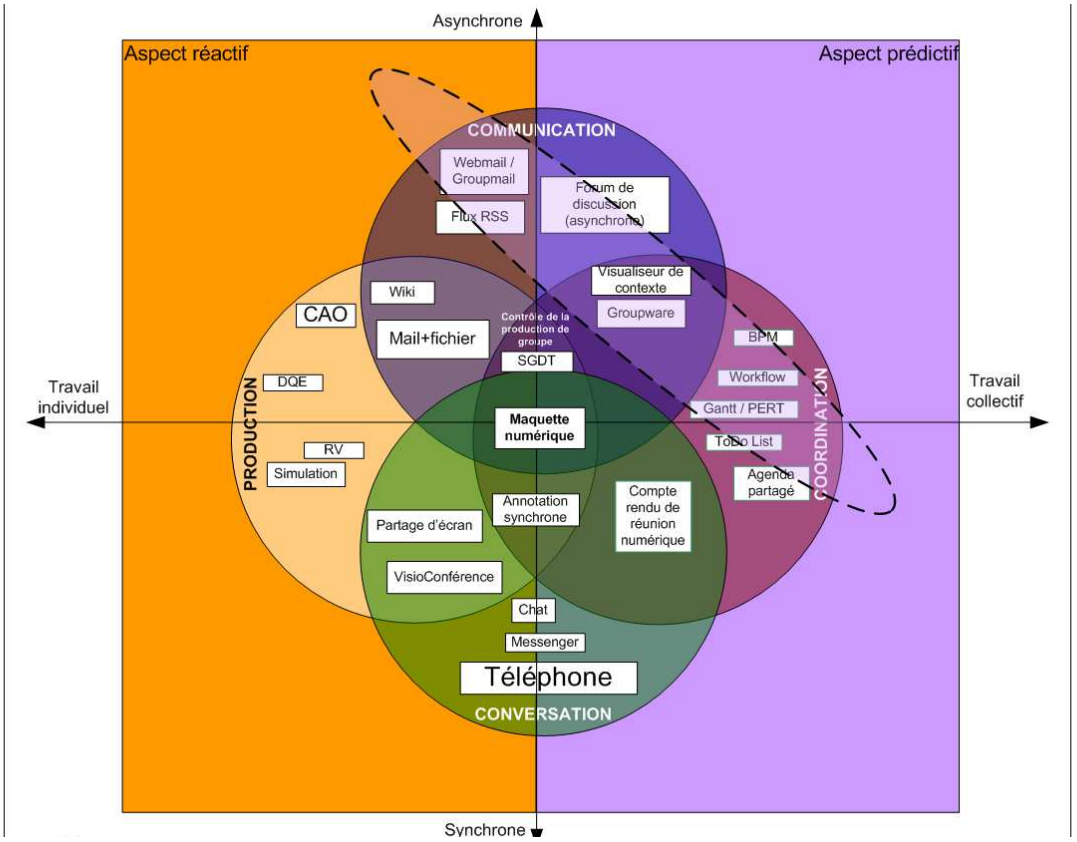


**FIGURE 5.6 :** Dualité entre activité prédictive et réactive, parmi les outils de conception en architecture. D'après [Laaroussi07a] et [Ellis94].

Par conséquent, la conception revêt un aspect prédictif non pas sur toute la nature de l'ouvrage, mais sur les activités permettant d'inventer ce nouvel ouvrage et ses fonctionnalités par rapport à des normes ou un cahier des charges. La loi MOP, la démarche qualité et la satisfaction de contraintes environnementales représentent des exemples entraînant des activités prédictives. De plus l'expérience personnelle capitalisée par chaque acteur de projet favorise cette logique, en particulier pour la gestion des risques et la prévision de conflits.

Nous avons montré précédemment que la problématique de la cohérence de la MNC par rapport aux différentes versions locales, issues d'une phase de conception distribuée, est un point fondamental pour un projet BTP. L'objectif consiste donc ici à proposer des solutions dans une logique prédictive.

Le diagramme de la figure 5.6 montre tout d'abord que l'activité prédictive demeure collective. Par rapport au processus-type que nous avons défini, une activité prédictive a lieu durant les réunions de synthèse ou de co-conception. Nous proposons alors plusieurs actions, lors de ces étapes de conception, où la comparaison sémantique peut jouer un rôle lors de l'étape suivante

de conception distribuée :

- Mise en place de milestones (points de synchronisation) supplémentaires avant la réunion de synthèse suivante. Il s'agit pour une partie ou la totalité des acteurs de synchroniser leur travail, notamment après l'aboutissement d'une tâche jugée complexe.
- Encourager le travail collectif, même lors de la phase de conception distribuée. Il s'agit de favoriser l'auto-coordination des acteurs.

Par rapport à ces deux actions, la comparaison sémantique peut jouer un rôle d'indicateur, grâce aux résultats de comparaison. En effet, les milestones supplémentaires permettent de fusionner une partie des versions locales de MNC, évitant dès lors une divergence trop grande (cf. section précédente). Or, les résultats de comparaison pourraient être réutilisés comme indicateur de la progression du modèle, et de son éloignement par rapport à la progression prédite. La notion d'indicateur est très utilisée dans la démarche qualité. Cependant, cela demande de transformer les résultats de comparaison en quantité mesurable par rapport à une référence validée. Cette considération dépasse largement le cadre de nos travaux. Concernant le deuxième point, [Hanser03] propose dans sa thèse un modèle d'auto-coordination des acteurs d'un projet architectural :

> *"La solution envisageable dans le cas qui nous intéresse est de laisser l'initiative aux acteurs tout en favorisant leur auto-coordination. Dans ce cas, c'est l'acteur et non plus le système qui prend l'initiative de mener une action de coordination ou de régulation de l'activité du groupe. Il est par conséquent impératif de permettre aux acteurs d'obtenir une information fiable concernant l'état du projet afin de déterminer quelles sont les actions à mener."*
>
> Damien Hanser, *Proposition d'un modèle d'auto coordination en situation de conception, application au domaine du bâtiment*, page 83.

L'obtention de cette information fiable peut aussi résulter, au moins partiellement, d'une transformation des résultats de comparaison. Dans tous les cas, les résultats de comparaison jouent ici un rôle passif dans une logique prédictive, car ils ne modifient pas le déroulement du processus. La figure 5.7 illustre le rôle de la comparaison sémantique au sein d'une activité prédictive.

La conception dans le secteur BTP n'a cependant pas pour vocation — ni pour habitude — d'être réduite à un ensemble d'activités prédictives. La tradition oral de ce secteur et la notion d'ouvrage unique font que l'activité réactive prend une très grande place au sein de ce secteur, contrairement à la conception de produits manufacturés. Par conséquent, notre analyse a besoin d'être complétée en étudiant la portée de la comparaison sémantique en activité réactive.

### 5.3.2 Portée de la comparaison sémantique en activité réactive

L'activité réactive englobe l'ensemble des réponses à des situations imprévues. Lors de la conception d'un projet de BTP, les imprévus sont multiples, suite à des calculs, des simulations,

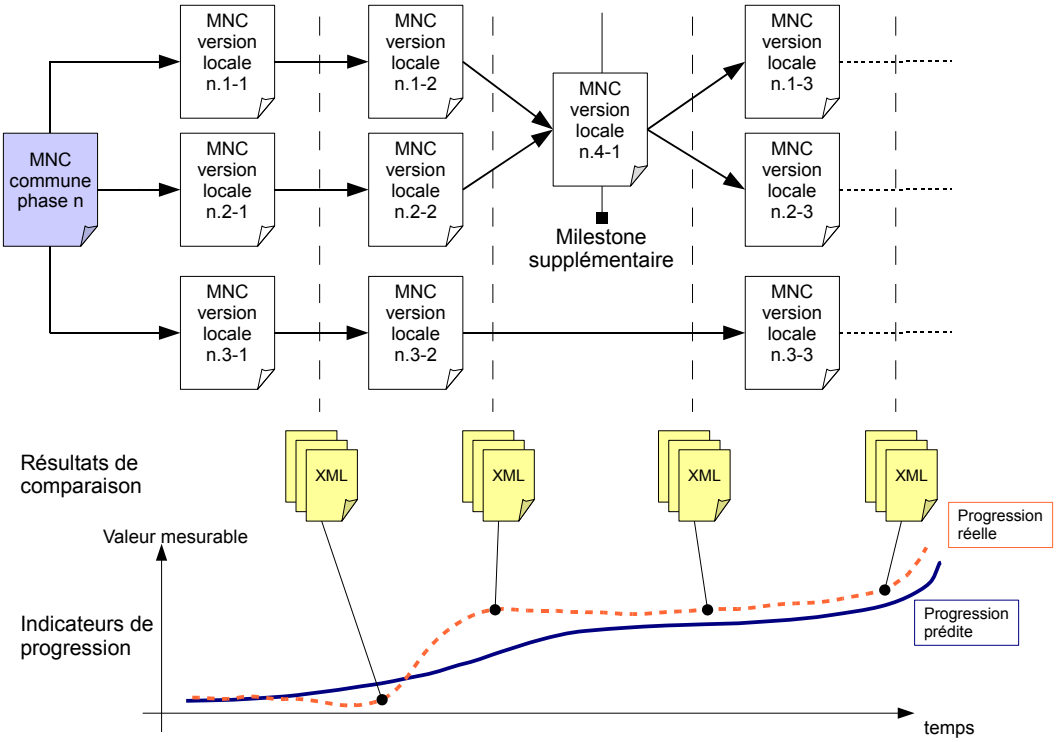


**FIGURE 5.7 :** Utilisation de la comparaison sémantique comme indicateur pour une conception prédictive. Dans ce scénario, le milestone supplémentaire a été prévu par la réunion de synthèse qui précède.

ou des nouvelles données de terrain, etc. Contrairement à l'activité prédictive, ces imprévus sont découverts de manière individuelle, comme l'illustre la figure 5.6. Par rapport au processus-type, il s'agit d'activités intervenant en particulier durant la phase de conception distribuée.

Nous proposons alors des perspectives, pour le comparateur sémantique, qui favorisent une activité réactive productive : l'envoi de notifications à partir des résultats de comparaison. Il existe dans le commerce des environnements logiciels architecturaux permettant l'envoi de notifications automatiques. Autodesk Revit©, en association avec Autodesk Buzzsaw©, possède une modélisation propriétaire de la MNC. Un même objet peut posséder une vue différente selon l'acteur de projet, notamment pour la géométrie (les IFC sont munis de fonctionnalités équivalentes). Lorsque l'architecte modifie une pièce de l'ouvrage, si cette dernière est munie d'une vue « ingénieur », une notification est alors envoyée au bureau d'étude concerné, qui peut accepter ou non la modification sur sa version locale. Par rapport à la classification de [Hanser03] inspiré de [Ellis94], il s'agit d'un système à concurrence totale, avec mise à jour de la MNC en temps réel.

Une application qui réutilise les résultats de comparaison sémantique pourrait jouer alors ce même rôle actif : notifier certains acteurs d'un changement de conception. Afin d'apporter une information fiable, il convient pour le système de sélectionner les notifications selon les

réultats de comparaison : la détection d'un changement, ou d'une incohérence, au niveau de l'étude des prix n'implique pas forcément l'envoi d'une notification au bureau d'étude. L'assistant sémantique tel que nous l'avons conçu pourrait être étendu pour supporter l'envoi sélectif de notifications, la sélection étant basée sur les techniques d'extraction d'information et d'équivalents sémantiques, vues au chapitre 3.

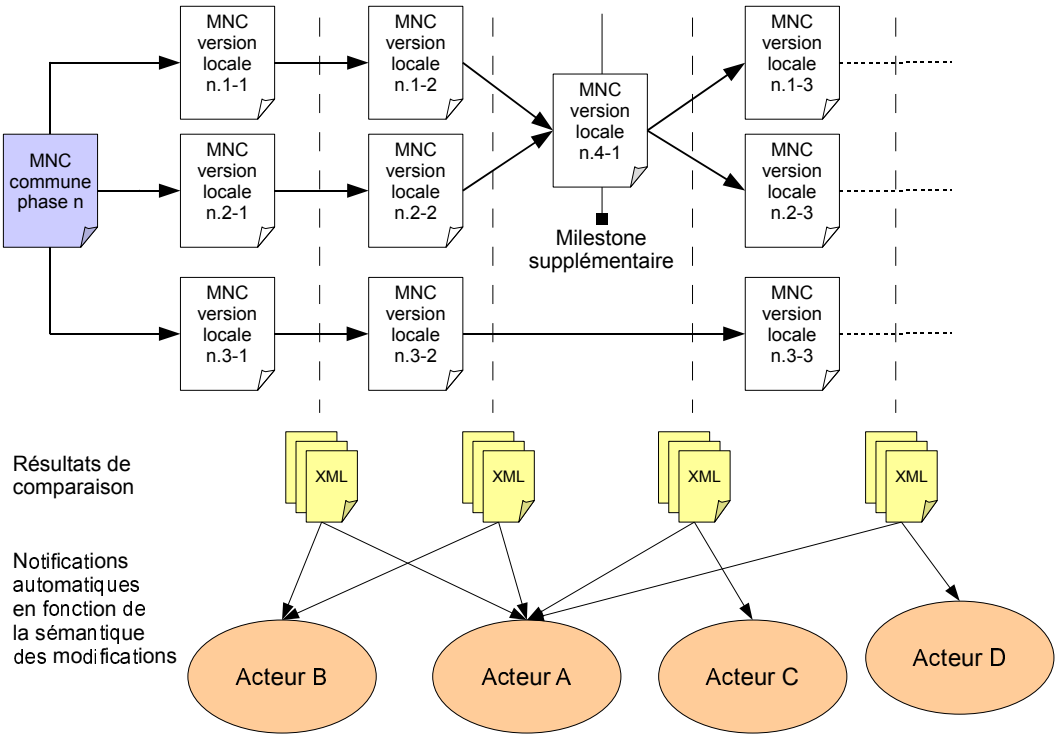


**FIGURE 5.8 :** Utilisation de la comparaison sémantique pour la création de notifications. Le milestone supplémentaire n'est ici pas forcément prévu depuis la réunion de synthèse précédente, mais peut découler des notifications envoyés aux acteurs concernés.

La figure 5.8 illustre le comportement actif de la synchronisation de MNC, via les comparaisons sémantiques. Durant la conception, nous pensons ainsi que les perspectives les plus productives pour le comparateur sémantique se situent au niveau de l'activité réactive. De manière plus générale, l'apport de nos travaux se situe en grande partie lors de la conception d'un projet. Cependant, la viabilité de la MNC dépend de son utilisabilité durant tout le cycle de vie de l'ouvrage, incluant ainsi la construction et la maintenance. Par conséquent, l'objectif consiste à imaginer quelles sont les perspectives de la comparaison sémantique lors de ces phases.

## 5.4 Portée de la comparaison sémantique au-delà de la conception

### 5.4.1 Mise à jour de la MNC en phase de construction

La séparation entre conception et construction est caractéristique d'un projet du secteur BTP [Anfosso05]. La loi MOP demeure en partie responsable via son principe d'une maîtrise d'oeuvre distincte de l'entreprise. Or, les marchés de type « conception-réalisation » fleurissent, surtout suite à certaines lois et ordonnances qui permettent d'utiliser ce type de contrat sans se soumettre aux conditions de la loi MOP. Ces dispositions sont valables pour certains ministères, en particulier la Santé, et ont permis de construire des hôpitaux sous cette forme de marché [Cabanieu06].

Même si la barrière réglementaire est contournable, la perte d'informations et la faible collaboration entre acteurs de conception et les entreprises rendent la MNC encore difficilement exploitable. [Anfosso05] met en avant les problèmes d'ordre logistique : fragilité du matériel informatique, sites de construction loin des bureaux d'études, etc. Vis-à-vis de la MNC, il s'agit de problèmes d'accès à l'information. D'abord, en lecture, l'utilisation de plans limitent la portée de la MNC. Cette dernière reste évidemment capable de générer des plans papier, mais il ne s'agit que d'une projection d'un vaste espace de données : l'information représentée est correcte mais certaines données sont perdues car non représentées sur des plans.

Ensuite, le problème semble s'aggraver pour la mise à jour de la MNC. En effet, durant la construction, il existe des imprévus sur le chantier. Les notes de chantier et les études d'exécution viennent donc modifier si nécessaire certains détails de l'ouvrage. Malheureusement ce type d'informations est perdu à cause de l'incapacité à accéder en écriture à la MNC. Cela se répercute lors de la maintenance où l'information disponible date de la conception et se révèle obsolète.

La problématique ne se situe pas au niveau des données et de la MNC elle-même. En effet, l'activité de construction reste très prédictive (respect des délais, cahier des charges précis, normes de construction, etc.) et la MNC ne subit que peu de modifications, en comparaison avec la phase de conception. Nous pensons donc que le processus-type et la comparaison sémantique, tels qui sont développés dans nos travaux, apportent une réponse suffisante pour le contenu à ajouter dans la MNC, lors de la phase de construction.

Les solutions à apporter correspondent plus à des technologies *in-situ*, à savoir les technologies ambiantes [Anfosso05] et le concept de *chantier numérique*. Il s'agit dans tous les cas de trouver des interfaces adaptés entre l'acteur et le document (ici la MNC) durant la phase de construction. Cette problématique d'IHM (interface homme-machine) dépasse le cadre de nos travaux.

Les considérations précédentes s'étendent facilement à la phase d'exploitation/maintenance. En effet, l'entretien d'un ouvrage nécessite une mise à jour de la MNC, facilitée de la même

façon par les technologies ambiantes et les IHM adaptées. Cependant, un autre problème se pose lorsque l'ouvrage à maintenir n'est pas basée sur une MNC mais sur une armoire à plans « classique ».

### 5.4.2 Maintenance des ouvrages existants

Comme cela a été vue au premier chapitre, l'adoption de la MNC, notamment autour des IFC ne fait pas l'unanimité parmi les projeteurs. Pour les ouvrages d'arts et en particulier les ponts, le manque d'implémentation autour de standards de MNC n'améliore pas la situation. Par conséquent, l'informatisation des données est basée généralement sur des standards de fait comme DXF/DWG de AutoCAD©. Il s'agit donc de données graphiques et géométriques dont la sémantique n'est pas beaucoup plus riche que celle des plans papier.

La portée de la comparaison sémantique reste très pauvre dans ce cas, à cause de l'inexistence de MNC. Néanmoins, si ces données géométriques sont converties dans un format issu d'un modèle de données EXPRESS, la comparaison sémantique pourrait tout de même s'appliquer sur la géométrie. A ce jour, le standard des données géométrique STEP se nomme AP 203 [ISO94b]. Malheureusement ce formalisme ne respecte pas les hypothèses pour l'élaboration de la comparaison structurelle : le modèle AP 203 utilise des modèles d'héritage ANDOR.

Afin de pallier à ce problème, il existe des méthodes pour ajouter des informations technologiques et plus généralement sémantiques à un modèle d'ouvrage purement géométrique. Ces processus restent manuels ou tout au moins supervisé par un expert, il s'agit donc de solutions onéreuses. Elles peuvent néanmoins être envisagées lors de projets de rénovation. Dans tous les cas, il s'agit de problématiques complémentaires mais différentes de celles de la comparaison et la synchronisation de MNC.

## 5.5 Conclusion

Une des hypothèses principales de nos travaux, à savoir le processus-type pour la gestion de MNC, montre ses limites pour la fusion des modèles d'ouvrages. Notre volonté de proposer un environnement flexible et qui autorise une grande liberté aux acteurs, lors de la conception distribuée, peut induire un report systématique des situations de conflits. Il s'agit d'une dérive qui entrave la productivité et favorise un comportement de procrastination[3]. Si un conflit nécessite le passage à une activité collective de synthèse ou de co-conception, une logique réactive impliquerait l'apparition d'activités collectives non prévus au départ et n'attend pas le point de synthèse prédit lors de la réunion précédente. Nous estimons donc que le processus-type tel que nous l'avons proposé au départ semble trop lié à une approche prédictive. Par conséquent, nous avons proposé des méthodes pour utiliser au mieux les mécanismes de synchronisation de la MNC, même en activité réactive. Les perspectives pour le comparateur sémantique à ce

[3]La procrastination est un terme relatif à la psychologie qui désigne la tendance pathologique à remettre systématiquement au lendemain quelques actions, qu'elles soient limitées à un domaine précis de la vie quotidienne ou non

sujet sont donc les suivantes : confirmer le bon déroulement d'un processus de conception en transformant les résultats de comparaison en indicateurs de progression (approche prédictive), détecter les conflits potentiels et encourager dès lors à une activité réactive par un mécanisme de notifications automatiques, sélectionnées en fonction des acteurs et des tâches en cours.

| | **Utile** | **Néfaste** |
|---|---|---|
| **Interne** | *Forces :* Indicateurs de progression | *Faiblesses :* Création d'un indicateur mesurable |
| **Externe** | *Opportunités :* Démarche qualité et développement durable | *Menaces :* Résistance au changement, historiquement très réactive |

**TABLEAU 5.1:** Matrice SWOT pour l'apport de la comparaison sémantique en conception prédictive.

| | **Utile** | **Néfaste** |
|---|---|---|
| **Interne** | *Forces :* Information sélectionnée et fiable | *Faiblesses :* Dérives d'une approche purement réactive et intuitive |
| **Externe** | *Opportunités :* Adaptation aux pratiques professionnelles | *Menaces :* Délais incertains |

**TABLEAU 5.2:** Matrice SWOT pour l'apport de la comparaison sémantique en conception réactive.

Au delà de la conception, nous pensons que cette dernière est déterminante pour la réussite des phases de construction et de maintenance. Les difficultés recontrées lors de ces phases ne sont pas relatives à la gestion interne de la MNC, mais plus aux interfaces permettant d'y accéder. Nous estimons alors que le processus-type et les mécanismes de comparaison et de synchronisation telles que nous les avons conçus seraient adaptables aux nouvelles technologies pour les IHM sur le chantier et au-delà. Des cas d'étude seraient néanmoins nécessaires pour confirmer ces considérations.

## Résumé

Ce chapitre vise à évaluer les limites de nos travaux et d'en proposer des perspectives, notamment pour la fusion des modèles d'ouvrage lors d'un processus de synchronisation. La première difficulté correspond à l'automatisation impossible de l'étape de synthèse des modèles, en cas de conflit. Le problème consiste non seulement à détecter le conflit entre deux versions (conflit structurel explicite, conflit de la géométrie sous-jacente, etc.) mais aussi à le résoudre. Une automatisation ne peut dépasser le cadre de la détection. La deuxième difficulté correspond au risque de divergence des modèles durant la phase de conception distribuée. Le nombre de combinaisons possibles pour fusionner les modèles devient explosif s'il n'y a pas suffisamment de points de coordination. Enfin, le problème de fusion de modèles reste fortement lié aux mécanismes de versionnement de la MNC. Dans le cadre de l'utilisation d'un versionnement à branches de type CVS ou SVN, il appartient à n'importe quel acteur de synchroniser les modèles d'ouvrage. Cette situation ne semble donc pas adaptée au domaine de la construction où l'ensemble des acteurs reste très hétérogène. Afin de pallier à toutes ces difficultés, une analyse est proposée afin de relier l'utilisation de la comparaison sémantique avec l'activité de conception. Cette dernière est alors vue de façon duale entre activité prédictive et activité réactive. Dans les deux cas, le système de comparaison sémantique apporte plusieurs perspectives : en activité prédictive, il serait utilisé comme indicateur de progression, de manière passive. En activité réactive, le système pourrait envoyer des notifications selon le type d'acteurs et le type de modification de la MNC. Ces notifications peuvent être à l'origine de points de coordination supplémentaires en cas d'imprévus. Enfin, les apports et limites de nos travaux sont vus après la conception. Lors de la construction, la problématique ne situe pas au niveau de la gestion des modèles d'ouvrage, mais plutôt sur les IHM et la mise à jour de la MNC à partir de notes de chantier. Durant la maintenance d'ouvrages existants, le principal problème reste l'absence de données sémantiques sur l'ouvrage. Avant de pouvoir appliquer nos travaux, il conviendrait de sémantiser les données informatiques disponibles, souvent purement géométriques.

## Abstract

This chapter aims to evaluate limitations of our work and propose enhancement perspectives, especially for product models merging during a synchronization process. The first issue relates to an impossible automation of the models synthesis step, during a conflict. This problem refers not only to detect a conflict between two versions (explicit structural conflict, geometry conflict, etc.) but also to solve it. Automation cannot go over detection phase. The second issue refers to divergence risk of product models during distributed design. Possible combinations for models merging intensively increase if there are not enough coordination milestones. Last merging problem is still strongly linked to DMUC versioning. By using branches based versioning like CVS or SVN, any actor can do synchronization between product models. This seems not be adapted to AEC sector, where actors have very different roles. In order to work around these issues, an analysis is proposed to link the use of semantic comparison with design activities. Design activities could be seen as a duality between a predictive activity and a reactive activity. Semantic comparison could bring solutions for both situations : this system could be used as a progress indicator in a passive way for a predictive activity. Otherwise this system could also actively send notifications to involved actors according to modification types. These notifications could create new milestones in case of unpredicted events. Last benefits and limitations from our work are seen over the design process. During the construction phase product models handling is not an issue here, but user interfaces and update of DMUC from notes of the construction site could be knotty. During maintenance phase the main problem is still about the lack of semantic models. Before applying our work, adding semantics to pure geometric models would be necessary.

# Conclusion générale

A travers cette thèse, nous avons étudié plusieurs aspects de la modélisation d'information, allant de la façon dont les entités peuvent être structurées aux méthodes permettant d'extraire sémantiquement des données. Cela nous a permis de développer une méthodologie générale pour le suivi d'un modèle d'ouvrage, durant la conception distribuée. L'étude et les tests des applications de CAO existantes nous ont notamment montré à quel point les données, même standardisées comme IFC, sont sujettes à des altérations, dues à une mauvaise interprétation par le logiciel, à des imprécisions numériques ou à des changements structurels profonds. Une mauvaise implémentation d'un standard de MNC remet en cause l'interopérabilité, pourtant nécessaire au travail collaboratif dans un environnement hétérogène d'acteurs et d'outils informatiques. Nous avons ainsi appris à porter un regard critique sur les détails du contenu des données et de leur changement. Nous estimons que la prise en compte de la sémantique permet de corriger, au moins partiellement, ces imperfections et constituent une solution de transition avant l'adoption généralisée des technologies de MNC, telles que IFC ou IFC-Bridge.

Nous avons aussi été témoins de l'évolution rapide des offres logiciels durant nos travaux de thèse. Peu à peu, ces dernières s'adaptent réellement aux méthodes de conception du secteur du BTP et aux besoins des acteurs, afin de favoriser une utilisation généralisée de la MNC, et faire ainsi avancer positivement les IFC sur la courbe d'adoption de Geoffrey A. Moore (cf. figure 1.22 à la page 45). Cependant, pour faire face à la résistante au changement, des travaux supplémentaires sont nécessaires, au-delà de problématiques purement informatiques.

## Apports de nos travaux

Les apports de nos travaux sont dans le champ des systèmes de comparaison structurelle et sémantique. L'analyse a été portée au niveau du méta-modèle (langage EXPRESS) plutôt que le modèle de données lui-même (IFC, IFC-Bridge, etc.). Cette généricité assure une robustesse face aux évolutions futures des MNC. A partir d'une configuration sur la base d'un assistant structurel (seulement quelques lignes de code XML), le constructeur d'application génère automatiquement une bibliothèque de comparaison structurelle, adaptée au modèle de données métiers, en langage C++. Cependant, les altérations de l'information, dues à des conversions entre logiciels, ou simplement par des cycles de chargement/sauvegarde, nous ont conduit à affiner le système par un assistant sémantique. L'analyse redescend alors au niveau du modèle de données mais rend le système très flexible : possibilité d'extraire partiellement l'information,

définition d'équivalents sémantiques, et de tolérances de valeurs numériques.

Par ailleurs, nous pouvons retenir deux atouts majeurs de cette approche :

- L'indépendance du système vis-à-vis de la persistance des données. L'implémentation ne nécessite aucune base de données orientée objets. Elle est compatible avec un système d'informations basé sur des fichiers (utilisation de la norme STEP-21). Cette indépendance laisse la liberté aux acteurs de projet d'utiliser leur propre solution de stockage, notamment pour les versions locales en conception distribuée.
- L'extraction d'information sans transformation de modèles. Grâce aux assistants sémantiques, il est possible de mener des comparaisons partielles, selon des vues métiers par exemple. Accompagné de l'assistant d'équivalents sémantique, le constructeur d'applications peut comparer des sous-modèles de données, extraits du modèle principal. Le mapping est alors créé automatiquement, de manière implicite.

Enfin, les techniques de synchronisation documentaire, en complément du suivi des modifications de la MNC, assurent une gestion complète de la MNC en conception distribuée. Elles allègent de plus certaines tâches répétitives, et donc peu épanouissantes pour le concepteur.

## Limitations de notre approche

La première limitation concerne les restrictions sur le modèle de données. Les algorithmes testés ne sont validés que sur des modèles de données EXPRESS n'utilisant pas d'héritage complexe (multiple, AND ou ANDOR). Si cela convient pour IFC, IFC-Bridge, ou encore le modèle de thermique spatiale TAS-ARM, il est par exemple impossible, en l'état, de développer ce comparateur sémantique pour la norme AP203 (norme EXPRESS supportée par certains logiciels de CAO).

Une deuxième limitation correspond à une dépendance sur un langage d'implémentation, ici C++. Cette dépendance a lieu dès la conception des assistants structurels et sémantiques. En effet, ces derniers contiennent directement des lignes de code dans le langage d'implémentation. Pour le comparateur structurel, la variante « fichiers squelettes » est aussi liée au langage. En l'état, il est donc impossible de concevoir un système de comparaison sémantique, implémentable directement à la fois en C++, C#, Java ou encore Python. Nous nous sommes limités à un seul langage d'implémentation afin d'obtenir rapidement des résultats. Néanmoins, aucune spécificité de ce langage n'a été utilisée, ces implémentations sont donc transposables d'autres langages de POO.

Enfin, une dernière limitation concerne l'indépendance vis-à-vis de la persistance. L'avantage de la flexibilité est contre-balancée par l'absence d'implémentation-test. Même si plusieurs scénarii avec un versionnement Subversion ont été détaillés, aucun lien entre le processus-type proposé et une application de persistance n'a été développé, principalement par manque de temps.

Si certaines limitations sont difficilement contournables sans une refonte complète des algorithmes (notamment les restrictions à certains types de modèles EXPRESS), d'autres au contraire permettent d'imaginer des perspectives possibles à partir de nos travaux.

## Perspectives

La première perspective constitue la création d'une application complète autour de la synchronisation de MNC. Cette perspective représente une application directe de nos travaux, sans travail de recherche supplémentaire. Afin de suivre l'émergence des clients légers et des applications web partagées, une variante pourrait s'orienter autour d'une architecture trois tiers :

- La couche présentation : il s'agit de l'interface web du client logiciel. Elle permettrait par exemple de mettre à jour la version locale de la MNC, de créer une nouvelle branche de versions locales, de consulter l'arborescence du projet, de synchroniser un document technique, de mener des tâches de coordination : synchroniser deux branches, créer des notifications automatiques, etc.
- La couche métiers : il s'agit d'une part du lien entre les requêtes de la couche présentation et les algorithmes de comparaison sémantique, de synchronisation documentaire et de versionnement. D'autre part, il s'agit de lier ces algorithmes à la persistance de la MNC.
- La couche persistante : elle contient les données elles-mêmes. Il est à noter que l'utilisation d'outils de versionnement Subversion ou CVS propose par défaut un support de persistance, côté serveur.

Une deuxième perspective consisterait à approfondir et hiérarchiser l'assistant sémantique. L'approfondissement se traduit par la recherche d'autres enrichissements sémantiques pour le comparateur, en complément de l'extraction d'information et les équivalents sémantiques. La hiérarchisation pourrait avoir lieu lorsque plusieurs règles sont en concurrence. Cela pourrait s'étendre jusqu'à l'obtention d'un ***moteur d'inférence***. De plus, en l'état actuel, l'assistant est un paramètre d'entrée du générateur de la bibliothèque de comparaison. Une fois généré, l'assistant sémantique ne peut plus être modifié. Une amélioration consisterait donc à introduire des assistants sémantiques dynamiquement. Le recours à un constructeur d'applications n'est plus nécessaire, car grâce à une interface simple, un acteur de projet pourrait changer des paramètres comme la sélection de telle ou telle information, le changement de la valeur de tolérance, etc.

Concernant les assistants, une autre perspective consisterait à enlever la dépendance au langage d'implémentation. L'assistant sémantique pourrait être défini en langage EXPRESS, (ce dernier n'étant pas un langage d'implémentation, mais un langage de spécification). Cela donnerait une plus grande cohérence car le modèle de données à la base d'une MNC est spécifié dans le même langage.

Enfin, une dernière amélioration envisageable concerne la réutilisation des résultats de comparaison. En construisant des relations avec un modèle orienté processus et activités de conception, le but consiste à déduire, des résultats de comparaison, des indicateurs de progression de la conception, l'éloignement par rapport au modèle prédit, etc. Ces indicateurs, couplés à

un mécanisme de notifications sélectives, selon la tâche et selon l'acteur, constituerait un outil novateur d'aide à la conception prédictive-réactive.

# Bibliographie


[AFN93] AFNOR. *Automatisation industrielle - Spécifications du standard d'échange et de transfert (SET) - NF Z68-300*, Octobre 1993.

[AFNOR94] AFNOR, editeur. STEP, Concepts fondamentaux. AFNOR, Saint-Denis-la-Plaine, France, 1994.

[AFPC94] AFPC, editeur. Les échanges de données informatisés dans le domaine des structures du génie civil. Bulletins scientifiques et techniques. Association Française Pour la Construction et SETRA, Paris, France, février 1994.

[Aitken03] Peter G. Aitken. Powering Office 2003 with XML (Power Pack Series). John Wiley & Sons, 2003.

[Ameziane95] Farid Ameziane. *Échange de données et ingénierie concourante dans le secteur bâtiment et synthèse des projets en cours dans le contexte normatif STEP*. In Rencontre des doctorants des ecoles d'architecture du sud de la France, Marseille, France, juin 1995.

[Amor06] R.W. Amor & H. Ma. *Preservation of meaning in mapped IFCs*. In Proceedings of the 6th European Conference on Product and Process Modelling, September 13-15, 2006, Valencia, Spain, pages 233–236. Taylor & Francis, 2006.

[Anfosso05] Alan Anfosso, Pierre T. Kirisci, Patrice Labussière, Marc Bourdeau & Alain Zarli. *Improving the construction process thanks to ambient technology solutions*. In AMI Forum. AMI@Work, 2005.

[Aranda-Mena06] G. Aranda-Mena & R. Wakefield. *Interoperability of building information - Myth of reality ?* In Proceedings of the 6th European Conference on Product and Process Modelling, September 13-15, 2006, Valencia, Spain, pages 127–133. Taylor & Francis, 2006.

[Bangert97] Claudia Bangert & Hartmut Prautzsch. *Circle and sphere as rational splines*. Neural, Parallel Sci. Comput., vol. 5, no. 1-2, pages 153–162, 1997.

[Bille03] Philip Bille. *Tree Edit Distance, Alignment Distance and Inclusion*, 2003.

[Björk95] Bo-Christer Björk. *Requirements and information structures for building product data models*. PhD thesis, VTT Building Technology, 1995.

[BLIS-Project05] BLIS-Project. *SABLE Project*, 2005. http://www.blis-project.org/~sable/.

[Borhani92] M. Borhani, J.P. A. Barthès, P. Anota & F. Gaillard. *A synthesis of the versioning problems in object-oriented engineering systems*. In Proceedings of 3rd international conference on Data and Knowledge Systems for Manufacturing and Engineering, pages 165–182, Lyon, France, March 1992.

[Bray04] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau & John Cowan. *Extensible Markup Language (XML) 1.1*, 2004.

[Brickley04] Dan Brickley. *RDF Vocabulary Description Language 1.0 : RDF Schema*, février 2004. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

[Broad03] Andrew Philip Broad. *Comparative Code Understanding of Information Models*. PhD thesis, University of Manchester, Manchester, UK, jun 2003.

[Brown01] A. Brown, M. Fuchs, J. Robie & P. Wadler. *XML Schema : Formal Description*. W3C Working Draft, September 2001.

[Cabanieu06] Jacques Cabanieu. Conception Réalisation, recommandations pour un bon usage du processus. Mission Interministérielle pour la Qualité des Constructions Publiques, avril 2006.

[Cederqvist06] Per Cederqvist. *Version Management with CVS*, 2006. http://ximbiot.com/cvs/manual/.

[Chawathe96] Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-Molina & Jennifer Widom. *Change detection in hierarchically structured information*. In Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 493–504, 1996.

[Chen04] Po-Han Chen, Lu Cui & Caiyun Wan. *Implementation of IFC-based web server for collaborative building design between architects and structural engineers.* Automation in Construction, 2004.

[Clair94] Dave St. Clair. *Versant : a powerful object database*. Adv. Syst., vol. 7, no. 9, pages 45–46, 1994.

[Clamen92] Stewart M. Clamen. *Type Evolution and Instance Adaptation*. Rapport technique CMU-CS-92–133, Carnegie Mellon University, Pittsburgh, PA, USA, 1992.

[Clark99] James Clark. *XSL Transformations (XSLT) Version 1.0*. W3c recommendation, W3C, November 1999.

[CNISF05] CNISF. *La maîtrise d'oeuvre au service du cadre de vie. Architectes et ingénieurs.* Le Moniteur, page cahier détaché nř2, septembre 2005. Rapport du groupe de travail présidé par Jean-Claude Parriaud, Comité du génie civil.

[CNRS06] Direction des Systèmes d'Information CNRS. *Glossaire du guide de conduite de projet systèmes d'information*, 2006. `http://www.dsi.cnrs.fr/conduite-projet/glossaire.htm#C`.

[Cobena02] Gregory Cobena, Serge Abiteboul & Amelie Marian. *Detecting Changes in XML Documents*. In ICDE, 2002.

[Cordella99] L. P. Cordella, P. Foggia, C. Sansone & M. Vento. *Performance Evaluation of the VF Graph Matching Algorithm*. In ICIAP '99 : Proceedings of the 10th International Conference on Image Analysis and Processing, page 1172, Washington, DC, USA, 1999. IEEE Computer Society.

[Corneil70] D. G. Corneil & C. C. Gotlieb. *An Efficient Algorithm for Graph Isomorphism*. J. ACM, vol. 17, no. 1, pages 51–64, 1970.

[Cruz02] Christophe Cruz, Christophe Nicolle & Marc Neveu. *The Active3D-Build : A Web-Based Civil Engineering Platform.* IEEE MultiMedia, vol. 9, no. 4, pages 87–90, 2002.

[Cruz04] Christophe Cruz, Renaud Vanlande & Christophe Nicolle. *ACTIVe3D : Semantic and 3D Databases for Civil Engineering Projects.* In Hamid R. Arabnia, editeur, Proceedings of the International Conference on Information and Knowledge Engineering. IKE'04, June 21-24, 2004, Las Vegas, Nevada, USA, pages 56–61. CSREA Press, 2004.

[Darken05] Rudy Darken, Perry McDowell & Erik Johnson. *The Delta3D Open Source Game Engine*. IEEE Computer Graphics and Applications, vol. 25, no. 3, pages 10–12, 2005.

[Debras98] Philippe Debras, Jean luc Monceyron, Fabrice Bauer, Philippe Ballesta & François-Xavier Rocca. *From Product Data Technologies to Applications : illustrative cases in the AEC domain*. In Proceedings of the CIB Working Commission W78, pages 163–170, 1998.

[Díaz05] Alicia Díaz. *Supporting Divergences in Knowledge Sharing Communities*. PhD thesis, Université Henri Poincaré, Nancy 1 — Universidad Nacional de La Plata, France — Argentine, 2005.

[Drogemuller06] R. Drogemuller. *Testing collaboration using IFC based software*. In Proceedings of the 6th European Conference on Product and Process Modelling, September 13-15, 2006, Valencia, Spain, pages 173–178. Taylor & Francis, 2006.

[Dubchak03] John Dubchak. *A template-based approach to XML parsing in C++*. Linux J., vol. 2003, no. 110, page 3, 2003.

[Ellis94] Clarence Ellis & Jacques Wainer. *A conceptual model of groupware*. In CSCW '94 : Proceedings of the 1994 ACM conference on Computer supported cooperative work, pages 79–88, New York, NY, USA, 1994. ACM Press.

[EPM06] EPM. *EXPRESS Data Manager, EPM Technology*, 2006. `http://www.epmtech.jotne.com/products/index.html`.

[ESA06] ESA. *TASVerter homepage*, 2006. `http://mechanical-engineering.esa.int/thermal/tools/?p=tasverter`.

[Foggia01] P. Foggia, C. Sansone & M. Vento. *A performance comparison of five algorithms for graph isomorphism*. Proc. of the 3rd IAPR TC-15 Workshop on Graph-based Representations in Pattern Recognition, pages 188–199, 2001.

[FSF02] Free Software Foundation FSF. *UNIX man pages : diff()*, 2002. `http://unixhelp.ed.ac.uk/CGI/man-cgi?diff`.

[Gamma94] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides. Design Patterns : Elements of Reusable Object-Oriented Software. Addison-Wesley, Août 1994.

[Garey90] Michael R. Garey & David S. Johnson. Computers and Intractability ; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA, 1990.

[Goossens99] Michel Goossens. The Latex Companion. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[Graustein30] W. C. Graustein. Introduction to Higher Geometry, chapitre Ch. 3, Homogeneous Cartesian Coordinates. Linear Dependence of Points and Lines, pages 29–49. Macmillan, New York, USA, 1930.

[Guerriero02] Annie Guerriero. *Etude de la coordination dans la coopération entre acteurs au cours de la conception d'un bâtiment*. Master report, Centre de recherche en architecture et ingénierie de Nancy, October 2002.

[Haas97] Wolfgang Haas. *AP225/ISO 10303-225*, juillet 1997. `http://www.haspar.de/Ap225/index_eng.htm`.

[Halfawy02] Mahmoud R. Halfawy & Thomas Froese. *Modeling and Implementation of Smart AEC Objects : An IFC Perspective*. In CIB w78 conference, 2002.

[Hannus00] Matti Hannus. *Islands of Automation in Construction*, 2000. `http://cic.vtt.fi/hannus/islands.html`.

[Hanser03] Damien Hanser. *Proposition d'un modèle d'auto coordination en situation de conception, application au domaine du bâtiment*. PhD thesis, Institut National Polytechnique de Lorraine, Nancy, France, October 2003.

[Hassanain01] M.A. Hassanain, T.M. Froese & D.J. Vanier. *Development of a maintenance management model based on IAI standards.* Artificial Intelligence in Engineering, vol. 15, pages 177–193, 2001.

[Hyenne93] J. Hyenne & R. Junge. *Le contexte normatif STEP*. In Les Échanges de données informatisés dans la construction, numéro 36 in Collection Recherches. Plan Construction et Architecture - Ministère de l'Equipement, du Logement et des Transports, 1993.

[IAI06] IAI. *IFC 2x3 final documentation*, 2006. `http://www.iai-international.org/Model/R2x3_final/index.htm`.

[ISO94a] ISO. Industrial automation systems and integration — Product data representation and exchange — Part 11 : Description methods : The EXPRESS language reference manual. International Organization for Standardization, Geneva, Switzerland, 1994. Norme ISO 10303-11.

[ISO94b] ISO. Industrial automation systems and integration — Product Data Representation and Exchange — Part 203, Application Protocol : Configuration Controlled Design. International Organization for Standardization, Geneva, Switzerland, 1994. Norme ISO 10303-11.

[ISO94c] ISO. Industrial automation systems and integration — Product data representation and exchange — Part 21 : Implementation methods : Clear text encoding of the exchange structure. International Organization for Standardization, Geneva, Switzerland, 1994. Norme ISO 10303-21.

[ISO95] ISO. Industrial automation systems and integration — Product data representation and exchange — Part 22 : Implementation methods : Standard Data Access Interface. International Organization for Standardization, Geneva, Switzerland, 1995. Norme ISO 10303-22.

[ISO96] ISO. Industrial automation systems and integration — Product data representation and exchange — Part 23 : Implementation methods : C++ Programming Language Binding to the Standard Data Access Interface Specification. International Organization for Standardization, Geneva, Switzerland, 1996. Norme ISO 10303-23.

[ISO00] ISO. Industrial automation systems and integration — Product data representation and exchange — Part 42 : Integrated generic resources : Geometric and topological representation. International Organization for Standardization, Geneva, Switzerland, 2000. Norme ISO 10303-42, second edition.

[ISO01] ISO. Industrial automation systems and integration — Product data representation and exchange — Part 27 : Implementation methods : Java TM programming language binding to the standard data access interface with

Internet/Intranet extensions. International Organization for Standardization, Geneva, Switzerland, 2001. Norme ISO 10303-27.

[ISO03] ISO. Industrial automation systems and integration — Product data representation and exchange — Part 28 : Implementation methods : STEP-XML XML representation for EXPRESS-driven data. International Organization for Standardization, Geneva, Switzerland, 2003. Norme ISO 10303-28.

[James06] Daniel James. *Boost C++ libraries – Functional/Hash*, 2006. `http://www.boost.org/doc/html/hash.html`.

[Katranuschkov06] P. Katranuschkov, U. Wagner, M. Weise & R.J. Scherer. *Supporting model-based cooperation in distributed web spaces*. In Proceedings of the 6th European Conference on Product and Process Modelling, September 13-15, 2006, Valencia, Spain, pages 237–244. Taylor & Francis, 2006.

[Katz86] Randy H. Katz, M. Anwaruddin & Ellis E. Chang. *A Version Server for Computer-Aided Design Data*. Rapport technique UCB/CSD-86-266, EECS Department, University of California, Berkeley, 1986.

[Knuth98] Donald E. Knuth. The Art of Computer Programming Volume 2. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[Ku06] K. Ku & S.N. Pollalis. *3D model-based collaboration and geometry control ; research needs for contractual standards : A case study of the main street bridge, Columbus, Ohio*. In Proceedings of the 6th European Conference on Product and Process Modelling, September 13-15, 2006, Valencia, Spain, pages 641–649. Taylor & Francis, 2006.

[Laaroussi07a] Ahmed Laaroussi. *Assister la conduite de la conception en architecture : Vers un système d'information orienté pilotage des processus.* PhD thesis, Institut National Polytechnique de Lorraine, Nancy, France, 2007.

[Laaroussi07b] Ahmed Laaroussi, Alain Zarli, Jean-Claude Bignon & Gilles Halin. *Towards a flexible IT-based system for process steering in architecture design*. In 24th W78 Conference — Bringing ITC knowledge to work, 2007.

[Lamb91] Charles Lamb, Gordon Landis, Jack A. Orenstein & Daniel Weinreb. *The ObjectStore Database System*. Commun. ACM, vol. 34, no. 10, pages 50–63, 1991.

[Lamour05] Gérard Lamour. Maîtrise d'Ouvrage et Maîtrise d'Oeuvre en Ouvrages d'Art. Mission Interministérielle pour la Qualité des Constructions Publiques, janvier 2005.

[Lemesle00] R. Lemesle. *Techniques de Modélisation et de Méta-modélisation*. PhD thesis, Ecole Centrale de Nantes, France, 2000.

[Lemoine04] Bertrand Lemoine. *L'architecte et l'ingénieur*. Discours à l'académie d'architecture, 2004.

[Liebich04] Thomas Liebich. *IFC 2x Edition 2*. Rapport technique, IAI Modeling Support Group, mars 2004.

[Maïssa03] Sandrine Maïssa. *Accès intuitif à l'information technico-règlementaire via une interface immersive. Application au domaine du bâtiment*. PhD thesis, ENSAM, Cluny, France, February 2003.

[Marache00] Mathieu Marache. *Application à l'ingénierie de la construction d'architectures client/serveur pour des applications de Réalité Virtuelle fondées sur des données STEP*. PhD thesis, Université de Nice, Sophia-Antipolis, France, avril 2000.

[Martin96] Robert C. Martin. *The Interface Segregation Principle*. objectmentor.com, 1996.

[Maruyama00] H. Maruyama, K. Tamura & N. Uramoto. Digest Values for DOM (DOMHASH). RFC Editor, United States, 2000.

[Mcguinness04] Deborah L. Mcguinness & Frank van Harmelen. *OWL Web Ontology Language Overview*, February 2004. `http://www.w3.org/TR/2004/REC-owl-features-20040210/`.

[Medi@construct04] Medi@construct. *Building Smart France*, 2004. `http://www.iai-france.org/index.html`.

[Moore01] Geoffrey A. Moore. Crossing the chasm. HarperCollins Publishers, 2001.

[Myers86] Eugene W. Myers. *An O(ND) Difference Algorithm and Its Variations*. Algorithmica, vol. 1, no. 2, pages 251–266, 1986.

[Nagel80] Roger N. Nagel, Walt W. Braithwaite & Philip R. Kennicott. *Initial Graphics Exchange Specification IGES, Version 1.0*. Rapport technique NBSIR 80-1978, U.S. National Bureau of Standards, Washington, DC, USA, 1980.

[Nell03] Jim Nell. *STEP on a Page http ://www.mel.nist.gov/sc5/soap/*, 2003.

[Nguyen89] Gia Toan Nguyen & Dominique Rieu. *Schema Evolution in Object-Oriented Database Systems.* Data Knowl. Eng., vol. 4, no. 1, pages 43–67, 1989.

[Nisbet05] Nick Nisbet & Thomas Liebich. *ifcXML Implementation guide*. Rapport technique, IAI Modeling Support Group, février 2005.

[OMG06] Object Management Group OMG. *Meta Object Facility (MOF) Specification*, 2006. `http://www.omg.org/technology/cwm/`.

[Pazlar06] T. Pazlar & Z. Turk. *Analysis of the geometric data exchange using the IFC*. In Proceedings of the 6th European Conference on Product and Process Modelling, September 13-15, 2006, Valencia, Spain, pages 165–171. Taylor & Francis, 2006.

[Pierra00] G. Pierra. *Spécification de données, le langage EXPRESS*. ENSMA, December 2000.

[Pilato04] Michael Pilato. Version Control With Subversion. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2004.

[Poyet97] Patrice Poyet & JL. Monceyron. *Les classes d'objets IFCs. Finalité et mode d'emploi*. Rapport technique, Centre Scientifique et Technique du Bâtiment, October 1997.

[Pras03] A. Pras & J. Schoenwaelder. *On the Difference between Information Models and Data Models*. RFC 3444 (Informational), January 2003.

[Raichvarg97] Daniel Raichvarg. *Les mots pour le dire.* TDC 741 — L'expérimentation scientifique, 1997.

[Rogers03] Everett M. Rogers. Diffusion of Innovations. Collier Macmillan, fifth edition, 2003.

[Rumbaugh88] James Rumbaugh. *Controlling propagation of operations using attributes on relations*. In OOPSLA '88 : Conference proceedings on Object-oriented programming systems, languages and applications, pages 285–296, New York, NY, USA, 1988. ACM Press.

[Schmidt76] Douglas C. Schmidt & Larry E. Druffel. *A Fast Backtracking Algorithm to Test Directed Graphs for Isomorphism Using Distance Matrices*. J. ACM, vol. 23, no. 3, pages 433–445, 1976.

[Simon96] Herbert A. Simon. The sciences of the artificial (3rd ed.). MIT Press, Cambridge, MA, USA, 1996.

[Skarra87] Andrea H. Skarra & Stanley B. Zdonik. *Type evolution in an object-oriented database*. Research directions in object-oriented programming, pages 393–416, 1987.

[STEP-Tools06a] STEP-Tools. *ROSE library reference*, 2006. `http://www.steptools.com/support/stdev_docs/roselib/roselib.html`.

[STEP-Tools06b] STEP-Tools. *ST-Developper*, 2006. `http://www.steptools.com/products/stdev/`.

[Sutter02] Herb Sutter. More exceptional C++ : 40 new engineering puzzles, programming problems, and solutions. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[Tanyer04] Ali Murat Tanyer & Ghassan Aouad. *Moving beyond the fourth dimension with an IFC-based single project database*. Automation in Construction, 2004.

[Turk92] Z. Turk. *Object Oriented Modelling Techniques and Integrated CAD*. In CIB W78 and CBS Meeting, Montreal, Quebec, Canada, Mai 1992.

[Turk93] Z. Turk & D. J. Vanier. *Classification systems in object oriented modelling of buildings*. In International Conference Design To Manufacture in Modern Industry, pages 571–578, 1993.

[Turk97] Z. Turk, R. Wasserfuhr, P. Katranuschkhov, R. Amor, M. Hannus & R. J. Scherer. *Conceptual Modelling of a Concurrent Engineering Environment*. Concurrent engineering in construction, pages 195–205, 1997.

[Ullmann76] J. R. Ullmann. *An Algorithm for Subgraph Isomorphism*. J. ACM, vol. 23, no. 1, pages 31–42, January 1976.

[Urtado98] Christelle Urtado. *Versions d'entités complexes. Approches microscopiques et macroscopiques*. PhD thesis, Université Montpellier II, Montpellier, France, October 1998.

[Vanlande03] Renaud Vanlande, Christophe Cruz & Christophe Nicolle. *Managing IFC for civil engineering projects*. In Proceedings of the 2003 ACM CIKM International Conference on Information and Knowledge Management, New Orleans, Louisiana, USA, November 2-8, 2003, pages 179–181. ACM, 2003.

[Wang03] Yuan Wang, David J. DeWitt & Jin-Yi Cai. *X-Diff : An Effective Change Detection Algorithm for XML Documents*. icde, vol. 00, page 519, 2003.

[Weisstein99] Eric W Weisstein. *Homogeneous Coordinates. From MathWorld–A Wolfram Web Resource.*, 1999. `http://mathworld.wolfram.com/HomogeneousCoordinates.html`.

[Weisstein03] Eric W Weisstein. *Birthday Problem. From MathWorld–A Wolfram Web Resource.*, 2003. `http://mathworld.wolfram.com/HomogeneousCoordinates.html`.

[Yabuki04] Nobuyoshi Yabuki, Jun Kotani & Tomoaki Shitani. *A Cooperative Design Environment Using Multi-Agents and Virtual Reality*. In Cooperative Design, Visualization, and Engineering, First International Conference, volume 3190 of *Lecture Notes in Computer Science*, pages 96–103. Springer, 2004.

[Yabuki06] Nobuyoshi Yabuki, Eric Lebegue, Jean Gual, Tomoaki Shitani & Li Zhantao. *International collaboration for developing the bridge product model*

"*IFC-BRIDGE*"'. In Joint International Conference on Computing and Decision Making in Civil and Building Engineering, pages 1927–1936, Juin 2006.

[Zhang92] Kaizhong Zhang, Rick Statman & Dennis Shasha. *On the editing distance between unordered labeled trees*. Inf. Process. Lett., vol. 42, no. 3, pages 133–139, 1992.

[Zyda05] Michael Zyda. *From Visual Simulation to Virtual Reality to Games*. Computer, vol. 38, no. 9, pages 25–32, 2005.

# Annexe A

# Structure du modèle IFC-Bridge

## A.1 Introduction

Les différents schémas sont extraits de la documentation officielle (non publiée) du modèle IFC-Bridge V2 de la révision datant du 01/01/2007. La figure A.1 est une introduction aux éléments de structure spatiale et aux éléments de construction.

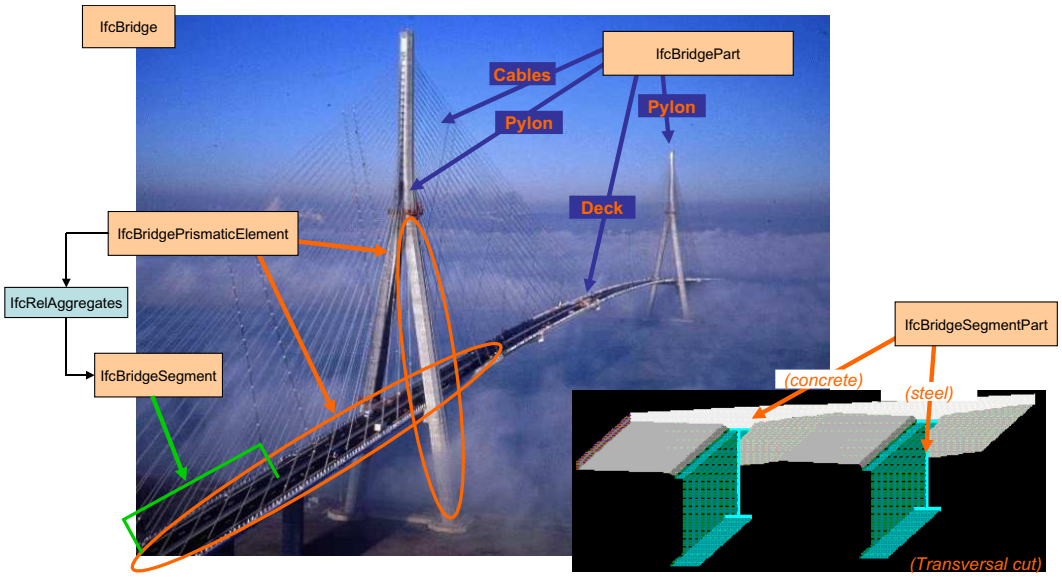


**FIGURE A.1 :** Introduction générale aux concepts du modèle IFC-Bridge.

## A.2 Éléments de structure spatiale

Le modèle IFC-Bridge introduit quatre nouvelles entités de structure spatiale, qui s'appuyent sur quatre nouveaux types. La notion de structure spatiale pour le génie civil préfigure aux prochaines extensions du modèle IFC aux routes. Le code EXPRESS associé est le suivant :

```
ENTITY IfcCivilStructureElement
ABSTRACT SUPERTYPE OF (ONEOF(IfcBridgeStructureElement))
```

```
SUBTYPE OF (IfcSpatialStructureElement);
END_ENTITY;

ENTITY IfcBridgeStructureElement
ABSTRACT SUPERTYPE OF (ONEOF(IfcBridge,IfcBridgePart))
SUBTYPE OF (IfcCivilStructureElement);
 StructureIndicator : IfcBridgeStructureIndicator;
END_ENTITY;

TYPE IfcBridgeStructureIndicator = ENUMERATION OF (
COMPOSITE,
COATED,
HOMOGENEOUS,
OTHER);
END_TYPE;

ENTITY IfcBridge
SUBTYPEãOFă(IfcBridgeStructureElement);
 StructureType : IfcBridgeStructureType;
END_ENTITY;

TYPE IfcBridgeStructureType = ENUMERATION OF (
BOX GIRDER BRIDGE,
ARCHED BRIDGE,
SUSPENSION BRIDGE,
CABLE-STAYED BRIDGE,š
GIRDER BRIDGE,
SLAB BRIDGE,
SLAB BRIDGE WITH BROAD CANTILEVER,
BOW STRING BRIDGE,
LADDER BRIDGE,
FRAMEWORK BRIDGE,
GISCLARD BRIDGE,
PORTAL BRIDGE);
END_TYPE;

ENTITY IfcBridgePart
SUBTYPE OF (IfcBridgeStructureElement);
 StrutureElementType : IfcBridgeStructureElementType;
 TechnoElementType : IfcBridgeTechnologicalElementType;
END_ENTITY;

TYPE IfcBridgeStructureElementType = ENUMERATION OF (
DECK,
PIER,
SMALL PIER,
PYLON,
ARCH,
LAUNCHING NOSE,
TEMPORARY BENT,
TRANSVERSE GIRDER,
STRUT,
COUNTER STRUT,
```

```
CABLE,
SUSPENDED TENDON,
SUSPENDER,
MOBILE FALSEWORK,
STAYING MAST,
LAUNCHING BEAM);
END_TYPE;

TYPE IfcBridgeTechnologicalElementType = ENUMERATION OF (
UNICELLULAR MONO BOX-GIRDER,
MULTICELLULAR MONO BOX-GIRDER,
UNICELLULAR MULTI BOX-GIRDER,
MULTICELLULAR MULTI BOX-GIRDER,
SOLID SLAB,
HOLLOW SLAB,
SLAB WITH BROAD CANTILEVER,
DOUBLE BEAM RIBBED SLAB,
MULTI BEAM RIBBED SLAB,
MASSIVE SECTION ELEMENT,
HOLLOW SECTION ELEMENT,
MARKETED SECTION GIRDER,
RE-ASSEMBLED SECTION GIRDER,
TRUSS,
LADDER OR VIERENDEEL,
BOW STRING);
END_TYPE;
```

La figure A.2 représente le diagramme d'héritage associé. La figure A.3 illustre, quant à elle, un exemple de hiérarchisation des éléments de structure spatiale et des éléments de construction.

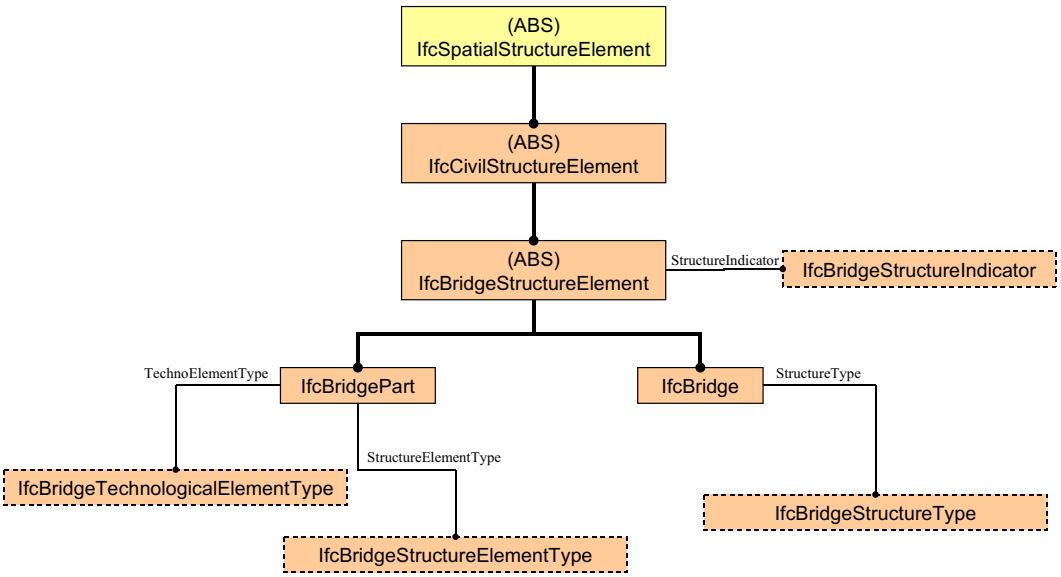


**FIGURE A.2 :** Diagramme d'héritage des éléments de structure spatiale.

## A.3 Éléments de construction

Quatre nouvelles entités s'appuyant sur deux nouveaux types enrichissent les éléments de construction pour un pont. En particulier, une pièce prismatique de pont est un élément de construction pouvant contenir plusieurs segments, eux-mêmes sont des éléments de construction à une échelle plus fine. Le code EXPRESS associé est le suivant :

```
ENTITY IfcCivilElement
ABSTRACT SUPERTYPE OF (ONEOF(IfcBridgeElement))
SUBTYPE OF (IfcElement);
END_ENTITY;

ENTITY IfcBridgeElement
ABSTRACT SUPERTYPE OF (ONEOF(IfcBridgePrismaticElement, IfcBridgeSegment))
SUBTYPE OF (IfcCivilElement);
END_ENTITY;

ENTITY IfcBridgePrismaticElement
SUBTYPE OF (IfcBridgeElement);
 PredefinedType : OPTIONAL IfcBridgePrismaticElementType;
END_ENTITY;

TYPE IfcBridgePrismaticElementType = ENUMERATION OF (
UNICELLULAR MONO BOX-GIRDER,
MULTICELLULAR MONO BOX-GIRDER,
UNICELLULAR MULTI BOX-GIRDER,
MULTICELLULAR MULTI BOX-GIRDER,
DOUBLE BEAM RIBBED SLAB,
MULTI BEAM RIBBED SLAB,
MASSIVE SECTION ELEMENT,
HOLLOW SECTION ELEMENT,
SOLID SLAB,
HOLLOW SLAB,
SLAB WITH BROAD CANTILEVER,
MASTER BEAM,
LONGITUDINAL GIRDER,
RIGIDITY BEAM,
BRACING,
UPPER FLANGE,
LOWER FLANGE,
UPPER FOOTING,
LOWER FOOTING,
WEB,
FLOORING SHEET,
BOTTOM SHEET,
KERB SHEET,
CANTILEVER SHEET,
PAVEMENT SHEET,
AUGET,
LONGITUDINAL WEB STIFFENER,
RAKER,
TRANSVERSE GIRDER,
```

```
DEFLECTER,
TRANSVERSE MEMBER,
TRANSVERSE,
DIAGONALE,
JAMB,
TENSION MEMBER,
BONDING BAR,
TRANSVERSAL STIFFENER,
STIFFENER FOOTING,
TENDON);
END_TYPE;

ENTITY IfcBridgeSegment
SUBTYPE OF (IfcBridgeElement);
 SegmentType: IfcBridgeSegmentType;
END_ENTITY;

TYPE IfcBridgeSegmentType = ENUMERATION OF (
TYPICAL SEGMENT,
PIER SEGMENT,
PIECE,
LIFT,
ELEMENT,
JAMB,
PYLON HEAD,
SPAN,
CANTILEVER,
FINITE ELEMENT REFERENCE);
END_TYPE;
```

Ces éléments de construction sont présents dans le diagramme d'héritage de la figure A.4.

## A.4 Parties d'éléments de construction.

L'extension IFC-Bridge propose une réorganisation des entités existantes dans IFC, permettant de connecter des éléments de construction « secondaires », et ajoute des éléments propres au génie civil. 6 nouvelles entités sont donc proposées, s'appuyant sur trois nouveaux types. La figure A.5 illustre un exemple d'utilisation de plusieurs parties d'éléments de construction.

Le code EXPRESS de ces entités est le suivant :

```
ENTITY IfcElementPart
ABSTRACT SUPERTYPE OF(ONEOF(IfcBuildingElementPart, IfcReinforcingElement, IfcCivilElementPart))
SUBTYPE OF (IfcElement);
END_ENTITY;

ENTITY IfcCivilElementPart
ABSTRACT SUPERTYPE OF(ONEOF
(IfcBridgeSegmentPart
,IfcBridgeContactElement
,IfcCivilSheath
,IfcCivilVoid))
```

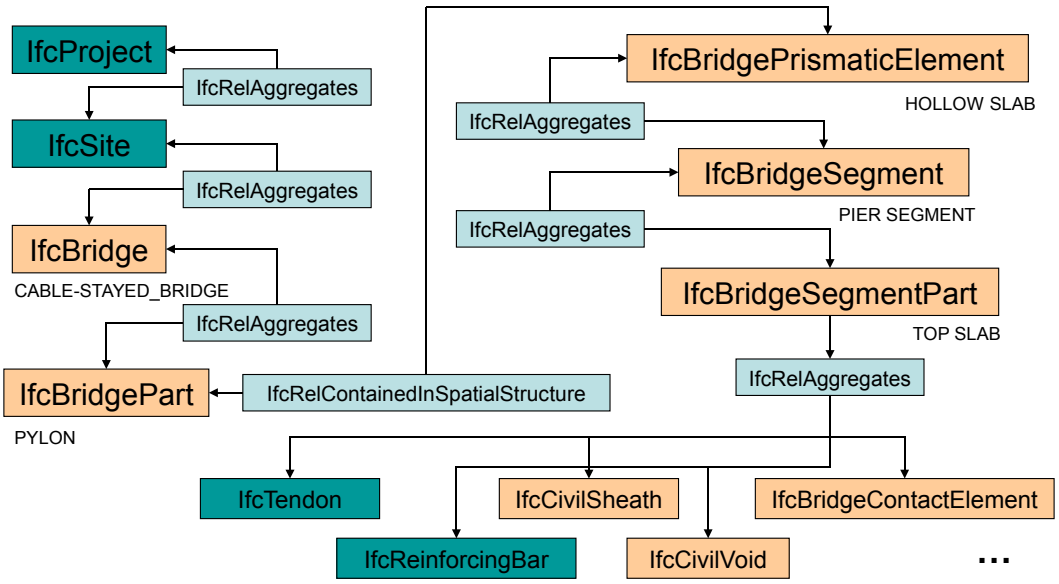


**FIGURE A.3 :** Exemple de hiérarchisation des éléments IFC-Bridge.

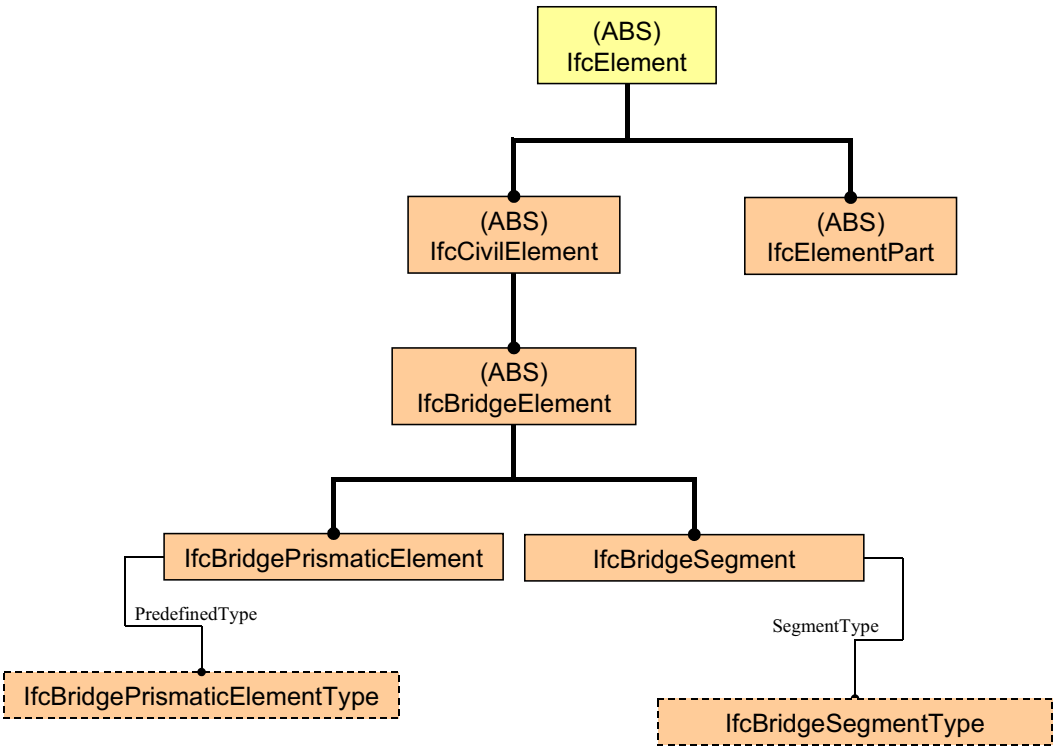


**FIGURE A.4 :** Diagramme d'héritage des éléments de construction.

```
SUBTYPE OF(IfcElementPart);
END_ENTITY;

ENTITY IfcBridgeContactElement
SUBTYPE OF (IfcCivilElementPart);
ContactType : IfcBridgeContactType;
END_ENTITY;

TYPE IfcBridgeContactType = ENUMERATION OF (
GLUE,
RIVET,
CONNECTOR,
WELD,
RESUMPTION_OF_CONCRETE,
SLIDING,
USERDEFINED,
NOTDEFINED);
END_TYPE;

ENTITY IfcBridgeSegmentPart
SUBTYPE OF (IfcCivilElementPart);
SubPartType : IfcBridgeSubPartType;
MechanicalRole : IfcBridgeMechanicalRoleType;
END_ENTITY;

TYPE IfcBridgeSubPartType = ENUMERATION OF (
LEFT WEB,
RIGHT WEB,
CENTRAL WEB,
TOP SLAB,
LOWER SLAB,
RIGHT OVERHANG,
LEFT OVERHANG,
UPPER FLANGE ,
LOWER FLANGE,
LOWER FLOORING,
UPPER FLOORING,
MORPHOLOGY NODE,
REFERENCE FIBRE,
BRANCH WALL);
END_TYPE;

TYPE IfcBridgeMechanicalRoleType = ENUMERATION OF (
LONGITUDINAL,
  TRANSVERSAL,
  COMPLETE,
  NONE,
  USERDEFINED,
NOTDEFINED);
END_TYPE;

ENTITY IfcCivilVoid
SUBTYPE OF(IfcFeatureElementSubtraction);
```

```
END_ENTITY;

ENTITY IfcCivilSheath
SUBTYPE OF(IfcCivilElementPart);
END_ENTITY;
```

La figure A.6 représente le diagramme d'héritage associé

En outre certaines propriétés on été ajoutés aux entités de câbles de précontrainte et aux éléments de connexions, via le mécanisme de PropertySets, non détaillé ici.

## A.5 Éléments d'alignements

Lors de la conception de ponts, les éléments ont besoin d'être localisé par rapport à un axe de référence. Plus généralement, le modèle IFC-Bridge propose de réorganiser certaines entités existantes et ajoute un nouveau concept : le référencement par rapport à une courbe. La figure A.7 propose un exemple de courbe de référence pour un pont et les systèmes d'axes locaux sont illustrés par la figure A.8.

D'un point de vue implémentation, quatre nouvelles entités font leur apparition. Voici le code EXPRESS associé :

```
ENTITY IfcAlignmentElement
ABSTRACT SUPERTYPE OF (ONEOF(IfcGrid, IfcPort, IfcReferenceCurve))
SUBTYPE OF (IfcProduct);
END_ENTITY;

ENTITY IfcReferenceCurve
 ABSTRACT SUPERTYPE OF (ONEOF
(IfcReferenceCurve3D,IfcReferenceCurveAlignment2D))
 SUBTYPE OF (IfcAlignmentElement);
 INVERSE
HasPlacements : SET [1:?] OF
IfcReferenceCurvePlacementSystem FOR BasedOn;
END_ENTITY;

ENTITY IfcReferenceCurve3D
SUBTYPE OF (IfcReferenceCurve);
Curve3D : IfcCurve;
END_ENTITY;

ENTITY IfcReferenceCurveAlignment2D
 SUBTYPE OF (IfcReferenceCurve);
HorizontalAlignment : IfcCurve;
VerticalAlignment : IfcCurve;
WHERE
WR1 : HorizontalAlignment.Dim = 2;
WR2 : VerticalAlignment.Dim = 2;
END_ENTITY;
```

La figure A.9 représente le diagramme d'héritage associé.

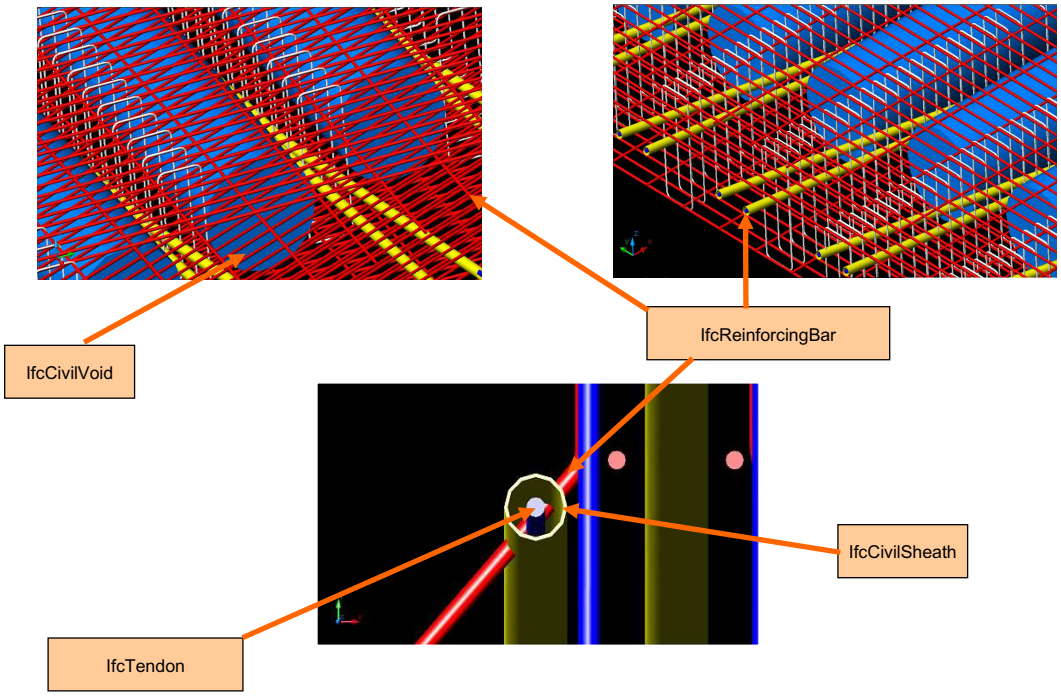


**FIGURE A.5 :** Exemples de parties d'éléments de construction.

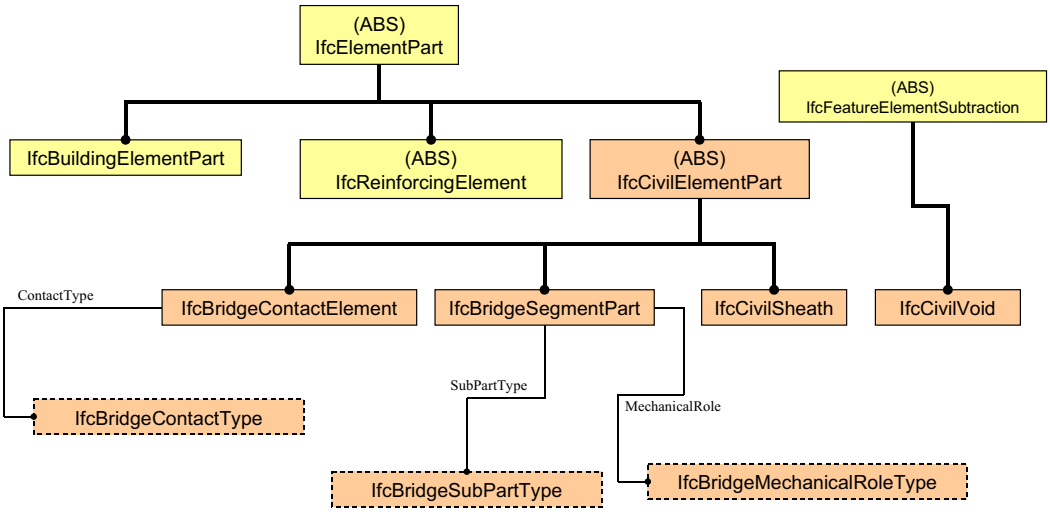


**FIGURE A.6 :** Diagramme d'héritage des parties d'élements de construction.

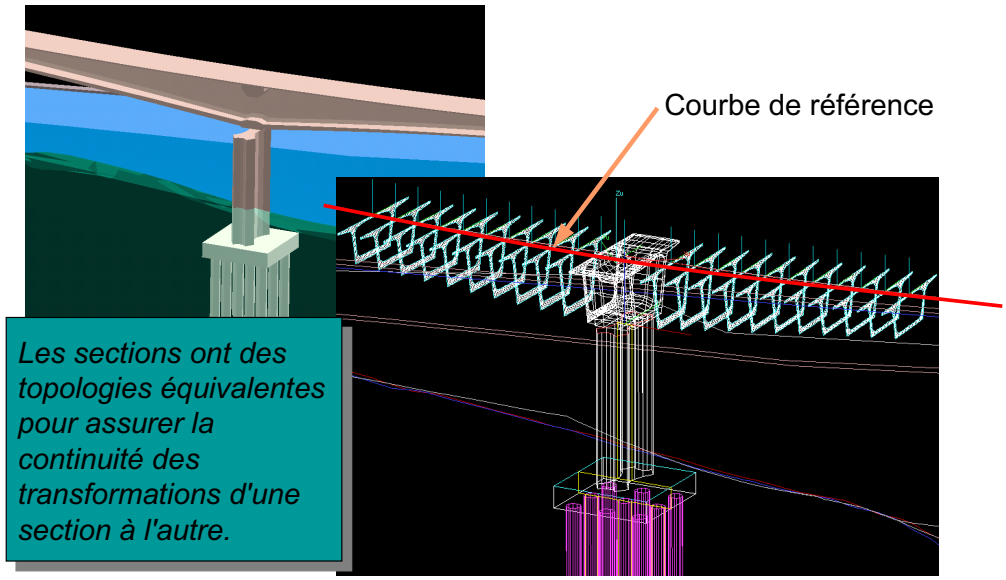


FIGURE A.7 : Introduction à la notion de courbes de référence.

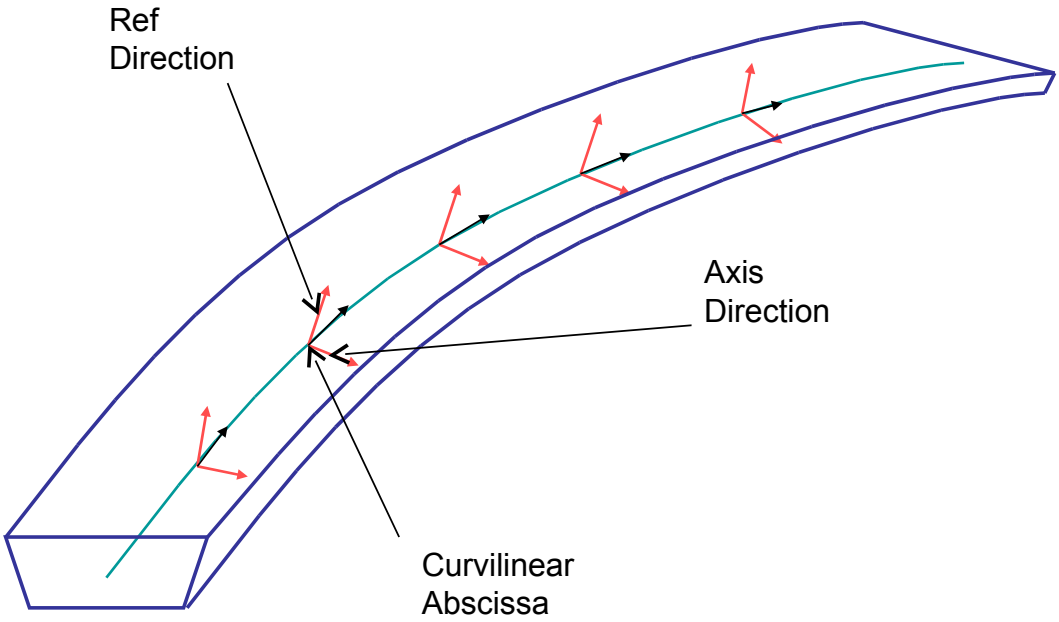


FIGURE A.8 : Systèmes d'axes locaux sur une courbe de référence.

## A.6 Localisation d'éléments

Il s'agit d'utiliser ici les éléments d'alignements définis précédemment grâce à trois nouvelles entités de localisation, dans le modèle IFC-Bridge. Voici le code EXPRESS de ces entités :

```
ENTITY IfcReferencePlacement
 ABSTRACT SUPERTYPE OF (ONEOF
(IfcReferenceCurvePlacement))
 SUBTYPE OF (IfcObjectPlacement);
END_ENTITY;

ENTITY IfcReferenceCurvePlacement;
    SUBTYPE OF (IfcReferencePlacement);
    CurvilinearAbscissa : IfcLengthMeasure;
    Axis : IfcDirection;
    RefDirection : IfcDirection;
    RelativeTo : IfcReferenceCurvePlacementSystem;
END_ENTITY;

ENTITY IfcReferenceCurvePlacementSystem
 SUBTYPE OF (IfcReferencePlacement);
Label : STRING;
BasedOn : IfcReferenceCurve;
 INVERSE
Includes : SET [1:?] OF
IfcReferenceCurvePlacement FOR RelativeTo;
END_ENTITY;
```

La figure A.10 représente le diagramme d'héritage associé.

## A.7 Les formes géométriques

Le modèle IFC-Bridge introduit deux nouvelles entités géométriques : la clothoide en tant que nouvelle courbe, et l'extrusion d'une courbe référencée, de section variable. Le code EXPRESS associé est le suivant :

```
ENTITY IfcClothoid
SUBTYPE OF (IfcCurve);
Position : IfcAxis2Placement;
ClothoidConstant : IfcLengthMeasure;
END_ENTITY;

ENTITY IfcReferencedSectionedSpine
 SUBTYPE OF (IfcSolidModel);
SpineCurve : IfcReferenceCurve;
CrossSections : LIST [2:?] OF IfcProfileDef;
CrossSectionPositions : LIST [2:?] OF IfcReferencePlacement;
END_ENTITY;
```

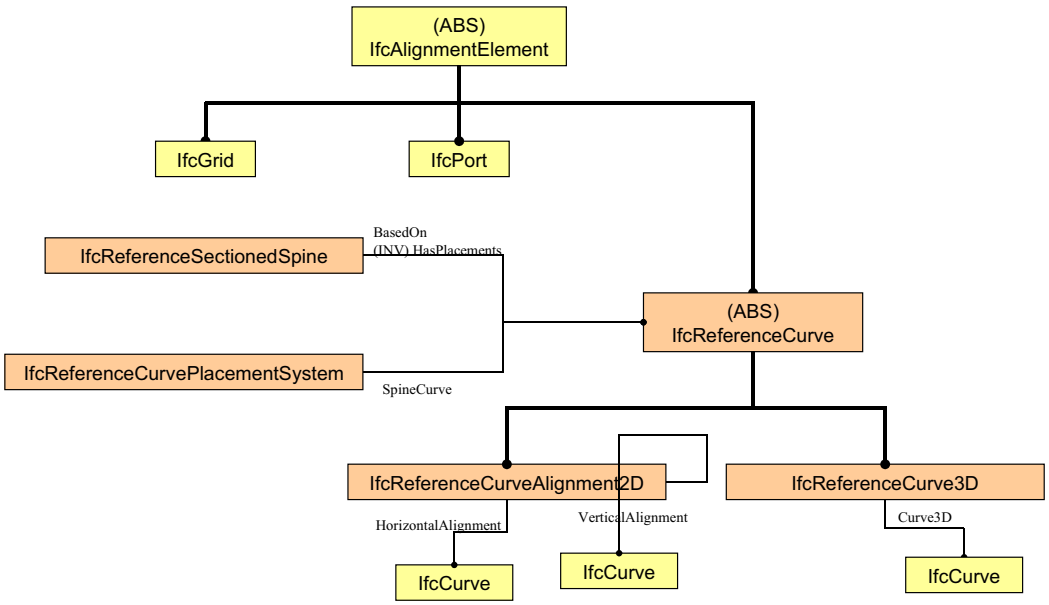


**FIGURE A.9 :** Diagramme d'héritage des éléments d'alignement.

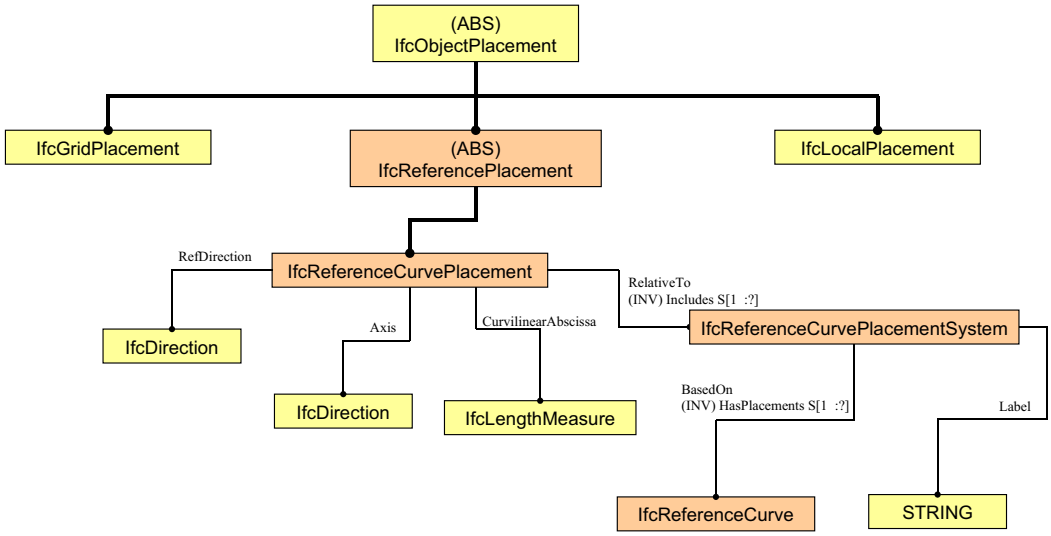


**FIGURE A.10 :** Diagramme d'héritage des éléments de localisation.

# Annexe B

# Transformations de modèles d'héritage EXPRESS

L'objectif de cette partie consiste à transformer n'importe quel modèle de données EXPRESS en un modèle de données qui utilise seulement l'héritage simple et dont la structure est équivalente, dans la mesure du possible. Nous rappelons que le langage EXPRESS exploite quatre types d'héritage :

- Héritage simple (**ONEOF**) : Un supertype référence un ou plusieurs sous-types. Un objet ne peut alors être l'instance que d'un seul de ces sous-types, il instanciera de plus le supertype de manière implicite.
- Héritage multiple (implicite) : Plusieurs supertypes référencent un sous-type. Si un objet instancie ce sous-type, il instancie implicitement chacun des supertypes.
- Héritage **ANDOR** : Un supertype référence plusieurs sous-types de cette manière. Un objet peut être l'instance d'un ou plusieurs de ces sous-types.
- Héritage **AND** : Un supertype référence plusieurs sous-types de cette manière. Si l'objet est l'instance d'un des sous-types, il devient alors instance de tous les autres sous-types.

Des mécanismes d'héritage différents peuvent aussi être combinés au sein du même supertype. Nous avons choisi un exemple représentatif de schéma EXPRESS, combinant tous les types d'héritage :

```
ENTITY a
SUPERTYPE OF (ONEOF(b,c) AND (d ANDOR e));
END_ENTITY;

ENTITY b
SUBTYPE OF (a);
END_ENTITY;

ENTITY c
SUBTYPE OF (a);
END_ENTITY;

ENTITY d
SUBTYPE OF (a);
END_ENTITY;
```

```
ENTITY e
SUBTYPE OF (a);
END_ENTITY;

ENTITY f
SUBTYPE OF (c,d);
END_ENTITY;
```

Afin de réaliser la transformation, le travail consiste d'abord à associer ce modèle à un graphe logique (section B.1), ensuite ce graphe logique subit une réduction afin de supprimer les héritages AND et ANDOR (section B.2). Enfin, une implémentation du modèle obtenue permet d'émuler l'héritage restant, en utilisant seulement des mécanismes d'héritage simple (section B.3).

## B.1 Du modèle EXPRESS au graphe de fonctions logiques

Le formalisme EXPRESS-G permet de représenter graphiquement n'importe quel modèle EXPRESS (cf. premier chapitre, section sec :subSTEP, figure 1.3), en particulier le diagramme d'héritage entre les différentes entités. Ce diagramme est un ***hypergraphe*** orienté, dont les sommets représentent les entités et les ***hyperarcs*** orientés le lien d'héritage entre un supertype et ses sous-types. La figure B.1 représente cet hypergraphe pour le modèle EXPRESS étudié.

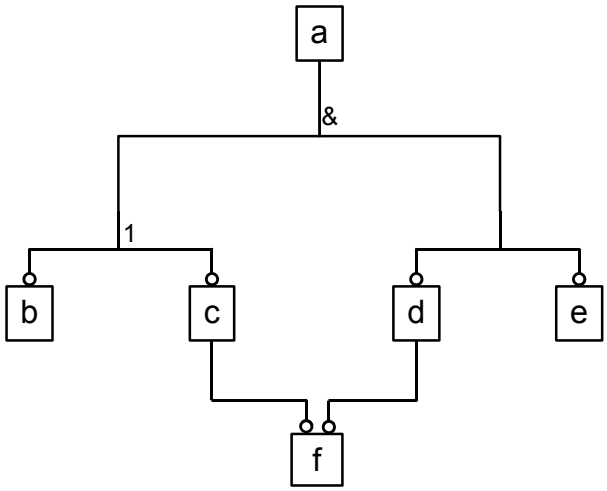


**FIGURE B.1 :** Diagramme d'héritage EXPRESS-G.

Dans ce formalisme, les hyperarcs sont munis de plusieurs symboles indiquant le type d'héritage : **&** correspond à un héritage **AND**, **1** correspond à un héritage **ONEOF** et aucun symbole ne correspond à un héritage par défaut **ANDOR**. Cet hypergraphe semble difficile à traiter en l'état. Par conséquent, il est transformé en un graphe de fonctions logiques : Les sommets de ce graphe regroupent les entités et des fonctions logiques élémentaires. Tous les hyperarcs reliant seulement deux sommets sont transformés en arêtes orientées dans l'autre sens. Les autres hyperarcs sont transformés en sous-graphes ne contenant que des fonctions logiques élémentaires.

Afin de respecter la norme européenne EN 60617-12 :1999 sur la représentation des symboles logiques, les types d'héritage sont associés aux symboles suivants : **&** pour l'héritage **AND**, $\geq 1$ pour l'héritage **ANDOR**, et **=1** pour l'héritage **ONEOF**.

Il s'agit d'un graphe orienté acyclique (***DAG***). De plus, un sommet d'entité ne peut avoir qu'une seule arête entrante, et un sommet de fonction logique ne peut avoir qu'une arête sortante. La figure B.2 représente le graphe du modèle EXPRESS étudié.

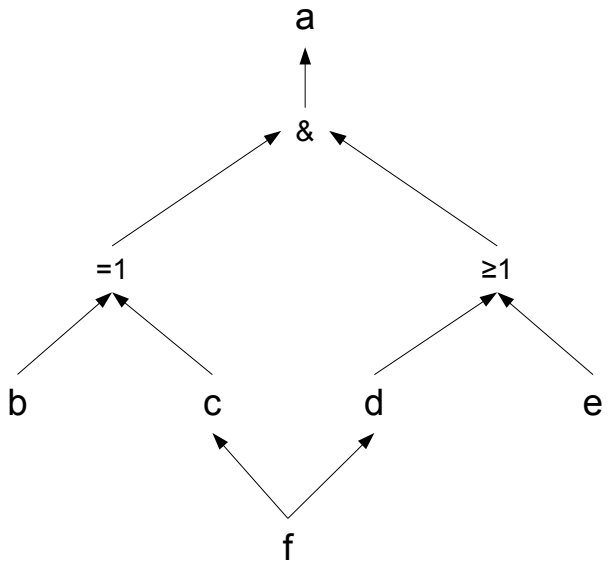


**FIGURE B.2 :** Graphe de fonctions logiques, associé au modèle EXPRESS étudié.

Il convient de préciser que les fonctions logiques ne sont pas forcément des opérations binaires, mais peuvent contenir plus de paramètres. Sur le graphe, cela se traduit par la possibilité pour un sommet de fonction logique de posséder plus de deux arêtes entrantes.

L'analyse des mécanismes d'héritage devient plus aisée grâce à ce formalisme. Il convient dès lors de rappeler qu'un objet du modèle d'ouvrage peut instancier explicitement plusieurs entités du modèle de données, à cause des héritages AND et ANDOR. Cependant, n'importe quelle combinaison n'est pas valide. Le graphe de fonctions logiques permet de vérifier facilement si une combinaison est viable. Par exemple, un objet peut-il instancier `c` et `e` ? Il convient pour cela de définir la notion de combinaison.

**Définition 15.** *Soit $C = \{e_0, e_1, ..., e_n\}$ une partie non vide de l'ensemble des sommets d'entités du graphe logique $S_E$. $P$ est une combinaison de $S_E$ si et seulement si, quelque soit le couple $(e_i, e_j)$ appartenant à $C^2$, où $i \neq j$, il n'existe aucun chemin orienté reliant $e_i$ et $e_j$. Dans le formalisme EXPRESS, la combinaison s'écrit $e_0 \& e_1 \& ... \& e_n$.*

En effet, si un chemin reliait $e_i$ à $e_j$ (dans le sens $e_i$ vers $e_j$), alors $e_i$ hérite de $e_j$, donc un objet qui instancie $e_i$ instancie implicitement $e_j$. Dans l'exemple étudié, `c&g` ne constitue pas une

combinaison car `g` est sous-type de `c`, une arête directe relie les deux entités.

Ensuite, il convient de vérifier si une combinaison $C$ est *valide*. Pour cela, il suffit :

1. d'affecter 1 aux noeuds appartenant à $C$,
2. de supprimer les arêtes entrantes de touts les noeuds de $C$,
3. d'affecter 0 aux feuilles n'appartenant pas à $C$.

Ensuite, un parcours logique suivant les arêtes est effectué. Une arête conserve la valeur du sommet d'origine (comme dans un circuit électrique), les fonctions logiques effectuent leur travail : **&** est un ET logique, $\geq 1$ est un OU logique, et **=1** est un OU exclusif (XOR). Au moment où le parcours se termine (car il s'agit d'un graphe acyclique), si le dernier sommet visité possède la valeur 1, l'objet est instanciable, sinon il ne l'est pas. Dans l'exemple précédent `c&e`, `a` est visité en dernier et possède la valeur 1, l'objet `c&e` est instanciable. Par contre, l'objet `b&f` ne l'est pas (fontion OU exclusif).

Les langages orientés objets usuels, comme Java et C++, n'autorisent pas une instanciation multiple de classes pour définir un objet. Il convient donc de transformer ce graphe, en supprimant toutes les fonctions logiques, par réduction de l'expression logique associée.

## B.2 Réduction du graphe logique

Réduire le graphe logique consiste à énumérer toutes les combinaisons valides pour un graphe donné. L'annexe B de [ISO94a] propose un algorithme de réduction de l'expression logique d'un héritage. Les combinaisons valides pour le modèle EXPRESS étudié sont les suivantes : `a`, `b&d`, `b&e`, `b&d&e`, `c&d`, `c&e`, `c&d&e`, `f`, `f&e`.

L'objectif consiste alors à construire un diagramme d'héritage UML classique, en introduisant les combinaisons valides comme nouvelles entités du modèle. D'un point de vue implémentation, à chaque combinaison valide contenant au moins deux éléments est associé une nouvelle classe qui effectue un héritage multiple de chaque entité constituant la combinaison. Chaque entité élémentaire est aussi associée à une classe, même si certaines ne peuvent être instanciées seules comme `b`, `c`, `d` et `e`. Cette approche suit les techniques d'implémentation proposées par la norme STEP-22 (cf. annexe A de [ISO95]).

Ce diagramme est directement implémentable en C++, qui supporte l'héritage multiple (cf. la norme STEP-23 [ISO96]). Grâce à l'implémentation multiple d'interfaces, Java permet aussi d'instancier un tel diagramme (cf. la norme STEP-27 [ISO01]). De plus, un nouveau modèle de données EXPRESS permet d'obtenir un tel diagramme d'héritage :

```
ENTITY a
SUPERTYPE OF (ONEOF(b,c,d,e));
END_ENTITY;
```

```
ENTITY b
SUBTYPE OF (a);
END_ENTITY;

ENTITY c
SUBTYPE OF (a);
END_ENTITY;

ENTITY d
SUBTYPE OF (a);
END_ENTITY;

ENTITY e
SUBTYPE OF (a);
END_ENTITY;

ENTITY f
SUBTYPE OF (c,d);
END_ENTITY;

ENTITY b_d
SUBTYPE OF (b,d);
END_ENTITY;

ENTITY b_e
SUBTYPE OF (b,e);
END_ENTITY;

ENTITY b_d_e
SUBTYPE OF (b,d,e);
END_ENTITY;

ENTITY c_d
SUBTYPE OF (c,d);
END_ENTITY;

ENTITY c_e
SUBTYPE OF (c,e);
END_ENTITY;

ENTITY c_d_e
SUBTYPE OF (c,d,e);
END_ENTITY;

ENTITY f_e
SUBTYPE OF (f,e);
END_ENTITY;
```

Cependant, nos travaux sont basés sur des modèles à héritage simple. De plus, l'implémentation de visiteurs (cf. chapitre 4, section 4.2.2) se révèle difficile en héritage multiple. Pour ces raisons, nous suggérons une méthode d'émulation de l'héritage multiple.

## B.3 Émulation de l'héritage multiple

Il s'agit d'obtenir le modèle structurellement le plus proche, sans héritage multiple. Il s'agit plus d'une problématique d'implémentation que de modélisation. En effet, il n'existe pas de modèle EXPRESS sans héritage multiple, équivalent à celui étudié. Par exemple, `g` hérite de `c` et `d`. En héritage simple, pour que `g` hérite de `c` et `d`, une solution consisterait pour `g` à hériter de `c`, et `c` hériterait de `d`. Mais cette solution modifie la structure de `c`, car il hérite à présent de `d`. Cela n'est pas sémantiquement acceptable : si l'on reprend l'exemple du chapitre 2, sur le `cheval`, l'`homme` et le `centaure`, la solution consiste à ce que `cheval` hérite de `homme` ou l'inverse.

C'est pourquoi, il s'agit d'une émulation, au niveau de l'implémentation. Notre méthode s'inspire et généralise la technique proposée par [Sutter02]. D'abord, il s'agit de scinder chaque entité héritant de plusieurs autres. La transformation s'effectue de la manière suivante :

Soit `entity` l'entité en cours, `parentA`, `parentB`, `parentC` trois supertypes de `entity` et `childA`, `childB`, `childC` trois sous-types de `entity`.

1. Création d'entités virtuelles `entity1` et `entity2`. Si $n$ est le nombre de supertypes de `entity`, $n-1$ entités virtuelles sont créées.
2. `entity` demeure le supertype de tous ses sous-types, mais seule `parentA` reste son supertype. A chaque entité virtuelle est attribuée une entité parente.

La figure B.3 illustre le procédé général.

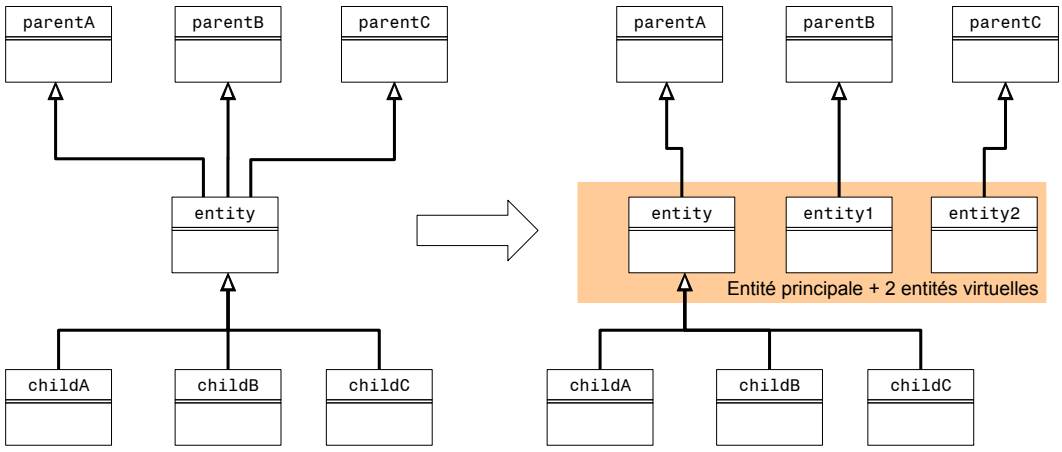


**FIGURE B.3 :** Principe de l'émulation de l'héritage multiple.

Ensuite, l'implémentation des entités scindées revêt une forme particulière. En effet, un instance de `entity` contient une instance de chaque entité virtuelle `entity<n>`. Chacune des entités virtuelles contient une référence vers `entity` et vers chacune des autres entités virtuelles. Le nombre de références est assez explosif : Soit $n$ le nombre d'entités (principale et virtuelles). On a $n \times (n-1) - (n-1) = (n-1)^2$ références (l'entité principale ne référence personne). Dans le modèle EXPRESS étudié, il y a au plus deux entités virtuelles, c'est-à-dire $n = 3$.

[Sutter02] propose une implémentation C++ pour un héritage double (une entité virtuelle, $n = 2$). Appliquons cette implémentation pour l'entité `f`, qui hérite de `c` et `d` :

```
<code C++ manquant>
```

Pour $n = 3$, en étudiant l'exemple de l'entité `c&d&e`, nous proposons l'implémentation suivante :

```
<code C++ manquant>
```

Il existe plusieurs limitations à cette implémentation. D'abord, ce mécanisme imite l'héritage multiple, notamment dans la façon dont l'information est stockée en mémoire. Malgré l'adjonction d'opérateurs, cette technique n'est pas syntaxiquement transparente pour l'utilisateur de cette classe, par rapport à l'héritage multiple.

# Table des figures

# Liste des tableaux

# Glossaire

**Artefact**

En gestion des processus, un artefact représente tout document (règlement, graphique, procédure, ...) identifié au sein d'un processus. Dans le cadre de ces travaux, il s'agit de l'ensemble des modèles d'ouvrages des documents de projet. La MNC est un exemple d'artefact.

**ASCII**

American Standard Code for Information Interchange, norme de codage de caractères en informatique la plus connue et la plus largement compatible.

**CFAO**

Conception et Fabrication Assistées par Ordinateur

**Hachage**

Voir *hashing*.

**Hashing**

Technique consistant à encoder une structure quelconque en une suite finie de caractères. Deux hashings différents correspondent à deux structures distinctes. Cependant l'inverse n'est pas vrai, mais un bon hashing minimise la probabilité que deux structures différentes ait le même hashing. Cette technique peut être ainsi utilisée pour optimiser la comparaison de structures complexes.

**Hyperlien**

Un hyperlien ou lien hypertexte ou simplement lien, est une référence dans un système hypertexte permettant de passer automatiquement d'un document consulté à un document lié.

**Hypertexte**

Système contenant des documents liés entre eux par des hyperliens

**Héritage**

Concept de la POO qui permet de définir un objet comme une spécialisation d'un autre objet : la voiture et la moto sont deux véhicules. L'objet voiture et l'objet moto héritent de l'objet véhicule.

**Instance**

Une instance d'un modèle de données est un ensemble hiérarchisé d'objets dont la structure et les liens sont en accord avec la définition et les règles du modèle.

**Loi de Moore**

Loi empirique qui postule le doublement annuel des performances des circuits intégrés (mémoires et processeurs).

**Mapping**

Dans le cadre de la modélisation de données, il s'agit d'un mécanisme de mise en correspondance entre deux éléments distincts. C'est une notion proche de la fonction en Mathématiques.

**Modèle d'information**

Il représente des informations d'un domaine dans un certain contexte sous une forme compréhensible par un humain. Il est indépendant des applications et des implémentations.

**Modèle d'ouvrage**

Il s'agit d'une représentation de l'ouvrage à concevoir et à construire, conforme à un modèle de données spécifique. Synonyme d'instance.

**Modèle de données**

Il décrit de façon abstraite et logique comment sont représentées les données dans une organisation métier ou un système d'information.

**Moteur d'inférence**

algorithme de simulation des raisonnements déductifs.

**Métadonnée**

Donnée servant à définir ou décrire une autre donnée quel que soit son support (papier ou électronique).

**Méthode**

fonction appartenant à un objet et lui définissant un comportement.

**Persistance**

Mécanisme responsable de la sauvegarde et de la restauration de données, afin qu'un programme puisse se terminer sans que ses données ni son état d'exécution soient perdus.

**POO**

Programmation Orientée Objets.

**STEP - STandard for Exchange of Product model data**

Projet ayant pour mission la construction d'un standard permettant de traiter la représentation et l'échange de données de modèles de produit en couvrant leur cycle de vie.

**Système multi-agents**

Ensemble d'agents situés dans un certain environnement et interagissant selon une certaine organisation. Un agent est une entité caractérisée par le fait qu'elle est, au moins partiellement, autonome. Ce peut-être un processus, un robot, un être humain, etc.

**Sérialisation**

Procédé qui consiste à pouvoir prendre un objet en mémoire et à en sauvegarder l'état sur un flux de données, un fichier par exemple. Le procédé inverse, appelé désérialisation, permet de recréer le même objet en mémoire à partir du flux entrant.

**Univers du discours**

Il se réfère généralement à tout ensemble de termes utilisés dans un discours spécifique, c'est-à-dire une famille de termes sémantiques spécifiques au domaine concerné. Il s'agit aussi d'un outil analytique utilisé en logique déductive et en logique des prédicats.

**Veille stratégique**

Elle regroupe les techniques de recherche documentaire et de traitement de l'information permettant la prise de décision stratégique pour une entreprise ou un pays

## Résumé

La Maquette Numérique (MN) est une représentation numérique, centralisée et hiérarchisée de l'ensemble des productions des acteurs au cours d'un projet. Dans le cadre du secteur BTP, la Maquette Numérique de Construction (MNC) constitue l'adaptation de la technologie précédente à ce secteur. Cependant, la conception en génie civil semble se distinguer de celle des produits manufacturés, notamment à cause de la multiplicité et l'hétérogénéité des acteurs, mais aussi de la création d'un ouvrage unique. Par conséquent, les processus en jeu durant cette phase nécessitent une analyse particulière. Notre travail consiste donc à proposer des mécanismes d'aide à la synchronisation de la MNC, après avoir montré que les outils existants, adaptés à la conception de produits manufacturés, ne peuvent être directement transposés au secteur du BTP. Pour cela, cette thèse se focalise sur le suivi des modifications de la MNC en cours de conception, par l'intermédiaire de comparaisons d'information structurées (comparaisons dites *structurelles*), tout en prenant en compte la signification des concepts utilisés (comparaisons sémantiques). L'analyse a été portée au niveau du méta-modèle (langage EXPRESS) plutôt que le modèle de données lui-même (IFC, IFC-Bridge, etc.). Cette généricité assure une robustesse face aux évolutions futures des MNC. A partir d'une configuration sur la base d'un assistant structurel (seulement quelques lignes de code XML), le constructeur d'application génère automatiquement une bibliothèque de comparaison structurelle, adaptée au modèle de données métiers. L'analyse sémantique rend, quant à elle, le système très flexible : possibilité d'extraire partiellement l'information, définition d'équivalents sémantiques et de tolérances de valeurs numériques. Le système conçu ne dépend d'aucun outil de persistance des données. De plus, l'extraction d'information s'effectue sans transformation de modèles. L'implémentation, en C++, de la comparaison structurelle et sémantique utilise des outils de génération automatique de code source. Enfin, ces travaux apportent de nombreuses perspectives s'ils sont liés aux activités prédictives et réactives de conception.



**Mots-clés** : Maquette Numérique, BTP, ponts, synchronisation, comparaison d'instances, comparaison sémantique, IFC, IFC-BRIDGE.


## Title

Structural and semantic comparison to synchronize Digital Mock-Up for Construction


## Abstract

Digital Mock-Up (DMU) is a digital, centralized and structured representation of all productions from project actors. Concerning AEC sector Digital Mock-Up for Construction (DMUC) refers to the adaptation of previous technology to this industrial field. Civil design seems however different from design of manufactured products, because of : multiplicity and heterogeneity of actors, creation of a unique product. Therefore involved processes need to be specifically analysed. This work aims to propose mechanisms to aid DMUC synchronization. Before that, we show that existing tools cannot be directly adapted to AEC sector. To do so, this thesis focuses on tracking changes within a DMUC during design process, thanks to comparison of structured information (structural information). We consider also the meaning of used concepts (semantic comparison). This analysis is done at the meta-model level (EXPRESS language) instead of a specific data model (IFC, IFC-BRIDGE). This generic approach guarantees some robustness for future evolutions of DMUC. From a configuration based on a structural helper (only a few lines of XML code), the software engineer automatically generates a library of structural comparison, adapted to the application data model. Semantic analysis makes the system very flexible : partial extract of information, definition of semantic equivalences and tolerance on numerical values. Designed system does not depend on any data persistence tool. Besides, extraction of information is carried out without any model mapping. C++ implementation of structural and semantic comparison uses tools of automatic generation of source code. Last this work provides several perspectives when they are associated to predictive and reactive design activities.



**Keywords** : Digital Mock-Up, AEC, bridges, synchronization, product models comparison, semantic comparison, IFC, IFC-BRIDGE.