Le concept d'objets dans les logiciels de DAO, CAO et de calcul

Roland Billon, Médiaconstruct et Enseignant à l'ENSA de Marseille
Isabelle Fasse, Médiaconstruct, ex Enseignante à l'ENSA de Marseille
Jacques Zoller, Professeur à l'ENSA de Marseille
Pascal Tonarelli, Maître de conférences à l'UVHC
Hafida Boulekbache, Maître de conférences à l'UVHC
Stéphane Duriez, médiatiseur à l'UVHC

Table des matières

Chapitre I. Résumé et prise de conscience 6

Chapitre II. Pourquoi approfondir les concepts de modèles conceptuels de Bâtiments ? 7

A. Vous avez atteint votre premier niveau de connaissances sur les IFC 7
B. Dominer les outils et les pratiques 8

Chapitre III. Le Concept d'objet dans les logiciels en AEC 9

A. Les entités traditionnelles de dessin sont solitaires 9
B. Les entités de CAO s'enferment dans leur club 11
C. Les entités objets adorent se tisser un réseau de relations 12

Chapitre IV. Quand « l'objet » appartient à la fois au langage informatique et au langage du bâtiment 14

A. La puissance des langages à objets au service de l'utilisateur 14
B. Les relations entre classes d'objets 15

Chapitre V. Le modèle conceptuel : une représentation des idées décrivant un phénomène 17

A. Quand on utilise des mots, entre entendre et comprendre 17
B. Un exemple de modèle conceptuel d'objets du bâtiment 18

C. La réalité est souvent plus complexe.....20

ChapitreVI. Arbre d'héritage des objets “produit” des IFC..... 24

A. Arbre d'héritage principal.....24
B. Spécialisation de la classe « Produit ».....25

ChapitreI. Résumé et prise de conscience

Résumé :

Les concepts fondamentaux des langages à objets, leur identification à un langage descriptif des éléments d'un projet de Bâtiment.

Le Modèle Conceptuel, comme base de données représentative de la logique organisationnelle du Bâtiment. Exemple de structuration de données.

Prise de conscience :

L'étudiant doit commencer à maîtriser les concepts et le vocabulaire exploités par les professionnels utilisateurs des systèmes d'information.

ChapitreII. Pourquoi approfondir les concepts de modèles conceptuels de Bâtiments ?

A. Vous avez atteint votre premier niveau de connaissances sur les IFC

Dans l'unité de cours précédent, vous avez intuitivement compris quelques concepts des IFC. Ce vernis est suffisant pour savoir à quoi sert un standard d'échange technique et graphique entre les logiciels des métiers de la Construction. Et comment il fonctionne, en gros.

Et si vous avez retenu

- que cette nouvelle technologie est en constante évolution,
- que l'utilisation des IFC vous oblige à une rigueur certaine dans la saisie des données de votre projet ou de l'exploitation de celles qui vous sont confiées,
- qu'il vous faut vérifier si vos outils informatiques sont capables des performances requises, et entre vous et eux, déterminer qui fait quoi,
- et que les contraintes des échanges vont obligatoirement modifier vos méthodes de travail,

alors le premier objectif de ce cours est atteint.

Si donc vous avez « tout compris », pourquoi perdre son temps à approfondir ?

Vous pouvez consulter directement la section "Décrire son projet pour satisfaire les échanges"et ses unités "Echanger les vues métiers du bâtiment", "Mettre son entreprise en conformité avec la norme" et "La pratique des logiciels normalisés IFC".

Pour trois raisons :

- Votre métier évolue radicalement. Et vous ?
- Remplacer les plans habituels par un fichier de données numériques d'ouvrages, ça ne s'improvise pas,
- Il faut comprendre le jargon des Technologies de l'Information et de la Communication propres au secteur du Bâtiment pour devenir un interlocuteur valable, ou « interopérable ».

B. Dominer les outils et les pratiques.

Nous ajoutons trois bonnes raisons supplémentaires pour approfondir :

1 : **Pour utiliser les IFC, on est obligé de se servir d'un logiciel, et pas n'importe lequel.**

Ce doit être un logiciel de la nouvelle génération, exploitant les technologies dites « *orienté objet* ».

En approfondissant ces nouveaux concepts, vous faites d'une pierre deux coups : mieux connaître les TIC, mais aussi mieux dominer les nouveaux outils de CAO ou logiciels techniques qui progressivement envahissent vos agences et bureaux d'études.

Car les concepts utilisés par un standard d'échange comme les IFC, sont également exploités dans les logiciels de la nouvelle génération.

2 : A travers le modèle conceptuel IFC, conçu initialement en vue des échanges dans un contexte informatique, c'est en fait **la première et certainement unique entente mondiale sur des définitions communes entre les divers métiers du Bâtiment** qui s'est mise en place.

En effet, comment imaginer qu'une autre action normative, un autre standard, un autre langage du Bâtiment soit étudié, qui prendrait le risque de ne pas être compatible avec les problèmes de communication entre logiciels ?

Par ailleurs, le seul organisme qui aurait pu rivaliser avec les IFC, le département *STEP* de l'*ISO*, ne s'y est pas trompé, en laissant officiellement le champ libre à l'IAI, par un accord signé en 1999. Puis plus récemment en reconnaissant aux IFC le statut d'une norme ISO. Les IFC deviennent une norme mondiale et unique.

3 : Pour devenir un **interlocuteur apprécié dans cette nouvelle méthode de travail collaboratif**, vous devez obligatoirement dominer les outils informatiques de votre propre métier, et les échanges techniques entre les métiers qui sont en relation.

Aucun autre moyen ne s'offre à vous que d'entrer dans le fond du sujet, donc de ce cours pour commencer. Ensuite, tout au long de votre activité professionnelle, vous serez par ailleurs obligé de mettre à niveau vos connaissances. Dès demain, ou même pendant ce cours, tant les techniques évoluent rapidement.

ChapitreIII. Le Concept d'objet dans les logiciels en AEC

A. Les entités traditionnelles de dessin sont solitaires

Pour aborder un concept nouveau, on le compare avec celui que l'on utilisait juste avant.

Essayons de saisir les différences entre les entités de dessin (que tout utilisateur a manipulé dans un logiciel qui utilise la structure de « *couches* ou de calque ») et les entités dites « *objet [ref. 5]* ».

Dans logiciel de dessin (2D, 3D) vous pouvez insérer une liste d'entités : des entités vectorielles (lignes, polylignes, cercles), des entités surfaciques (boîtes, pyramides, surfaces extrudées ...), et pour certains plus puissants, des solides (cubes, sphères ...), ou encore des entités plus complexes, résultats d'opérations de traitement (hachures, cotation, lignes cachées coupes, ...).

Ces entités traditionnelles admettent une structure simple. Quand elles sont dites «complexes», elles sont obtenues par juxtaposition ou concaténation d'entités élémentaires.

Par exemple, sur écran ou sur traceur, un arc est constitué d'une polyligne ou de points juxtaposés.

L'utilisateur ne sait pas comment cette information graphique est stockée dans la base de données interne du logiciel (dans sa mémoire). Et il n'a pas besoin de le savoir.

Mais intuitivement, il se rend compte que son logiciel «connaît» un certain nombre de propriétés et de relations concernant chaque entité. L'utilisateur comprend qu'il pourra demander des actions sur cette entité, en agissant exclusivement à partir de ses propriétés si l'éditeur du logiciel a prévu une commande dynamique pour ce faire.

Par exemple, s'agissant d'un arc de cercle, il n'est pas difficile, intuitivement, de prévoir que les informations suivantes se trouvent quelque part dans la base de données du logiciel ou que, du moins, le logiciel sait les retrouver par un calcul ou une recherche :

Informations géométriques :

- Centre (dans l'espace)
- Rayon (distance)
- Angle de début, (ce qui détermine le point origine, dans l'espace)
- Angle de fin, (ce qui détermine le point extrémité, dans l'espace)
- Point milieu sur l'arc de cercle (ce qui donne le sens de la concavité)
- Définition du plan dans l'espace contenant l'arc de cercle, donc altitude dans un cas particulier (lorsque l'arc de cercle est situé dans un plan horizontal)
- Partage de l'espace entre région du coté concave et région du coté convexe.

Paramètres et propriétés propres :

- Paramètre du nombre de segments
- Propriétés des traits (couleur, type de ligne, épaisseur)

Relations :

- Relation de l'arc de cercle avec le calque support et ses propriétés, par transitivité.

L'utilisateur veut par exemple transformer un arc de cercle tracé en plan, pour créer une surface verticale courbe. Dans son imagination, il veut simuler ensuite ce qu'il appelle une cloison courbe, puis réaliser l'intersection de deux cloisons courbes munies d'ouvertures.

Si l'éditeur a prévu une propriété géométrique supplémentaire à cette liste, qui est par exemple la **hauteur de l'arc de cercle** (initialement mise à zéro), la transformation s'effectuera par une simple modification d'un paramètre dans une fiche de propriétés.

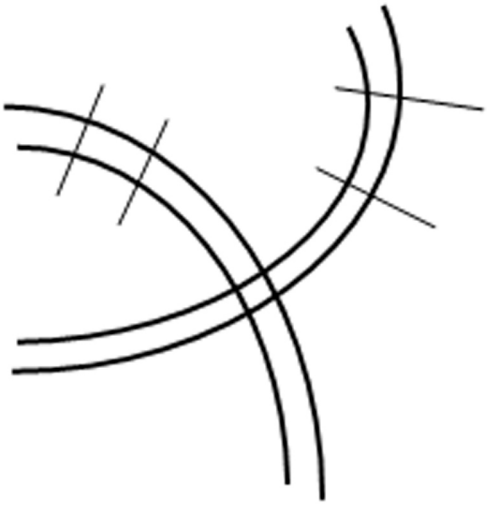
C'est rapide.

Dans le cas contraire, l'utilisateur doit se débrouiller autrement, en utilisant les ressources de son logiciel de dessin. Par exemple, créer une autre entité surface, par déplacement de l'arc de cercle devenu profil, le long d'une génératrice de longueur égale à la hauteur de la future cloison ... C'est plus long.

C'est encore plus long si l'utilisateur veut donner une épaisseur à «sa cloison». Et encore plus long pour pratiquer les percements. Et il risque de se décourager si en plus il veut résoudre correctement le dessin d'assemblage de deux cloisons courbes qui se rencontrent !

En effet, un arc de cercle, une surface de révolution, ignorent ses voisins de même entité, et plus encore s'ils sont de nature différente. L'intersection doit être gérée manuellement par l'utilisateur, ce qui revient pour lui à dire au logiciel qu'il existe une relation entre plusieurs entités, deux à deux. Il doit combler la pauvreté des relations qualifiant les entités.

Il pourra être amené à changer complètement de méthode, abandonner les entités surfaciques pour utiliser les solides des opérations booléennes, finalement mieux adaptées à son problème de modélisation géométrique. Mais ce ne sera pas encore parfait.



Avec des entités de dessin, que d'opérations pour obtenir une représentation de deux murs courbes percés d'ouvertures, que ce soit en plan ou en 3D ! Ici, le début du dessin, avec quatre arcs et la délimitation des ouvertures, en plan.

B. Les entités de CAO s'enferment dans leur club.

On est tenté d'affirmer que pour répondre au problème posé, il faut abandonner la manipulation d'entités de dessin généraliste, au profit d'entités de dessin spécialisées en architecture et construction. C'est ce que font les logiciels de CAO depuis une quinzaine d'années.

En choisissant une commande spécialisée « Mur courbe », et en spécifiant ses paramètres, le dessin devient automatique. Le logiciel gère tout seul l'intersection des murs entre eux, et intègre les ouvertures. C'est le moins que l'on puisse demander à un logiciel graphique dédié au bâtiment.

On pourrait donc formuler cette conclusion, en guise d'explication de la différence entre entité de dessin, et objet :

« Un objet serait une entité de dessin spécialisée pour un domaine d'application ».

Ce serait faux.

Il peut exister des entités de dessin qui sont structurées en objets. Comme il existe des entités spécialisées -les composants du bâtiment de la plupart des logiciels de CAO- qui ne sont pas structurées en objets.

La différence ne porte pas non plus sur la liste des paramètres et des propriétés propres à l'entité. Si une entité est riche en informations, bien évidemment, elle peut potentiellement susciter un grand nombre d'opérations. Mais ce critère n'est pas suffisant pour rendre cette entité « intelligente ».

A force de développement et de commandes spécialisées l'éditeur peut simuler en programmation traditionnelle un comportement intelligent de certains composants.

Il exécute un cahier des charges de plus en plus complexe. Il donne l'impression qu'une entité est intelligente, alors qu'elle répète comme un animal bien dressé une série de gestes programmés à l'avance pour un contexte bien défini. Les commandes (qui ne s'empilent pas) interdisent d'ailleurs à l'utilisateur de sortir des sentiers battus.

Bien évidemment, cette course aux fonctionnalités aboutit à des monstres de logiciels, dont le nombre de commandes finit par indisposer l'utilisateur, et encore plus le développeur, confronté à une complexité croissante du code et de la maintenance du logiciel.

L'obligation d'intégrer les IFC doit révéler rapidement les limites de cette déjà trop vieille

génération de logiciels, juste au moment où ses fonctionnalités atteignent leur apogée.

Trop d'entités différentes, trop de relations à gérer vont faire éclater le club fermé et protocolaire des entités de CAO traditionnelles privées de contact avec l'extérieur.

C. Les entités objets adorent se tisser un réseau de relations.

Dans notre cas, l'intelligence de l'entité, c'est sa capacité à réagir avec son environnement, sans obliger l'opérateur du logiciel à gaspiller son énergie en manipulations laborieuses, et sans obliger l'auteur du logiciel à prévoir d'avance tous les cas et toutes les commandes.

Pour ce faire, l'entité doit savoir quand un événement se produit. Et quoi faire ensuite sans le demander systématiquement à l'utilisateur.

Le contexte de prise en compte de cet événement est en général décrit par un ensemble de relations de voisinage qui sont satisfaites (ou non).

On l'aura deviné (et c'était déjà évoqué dans l'unité de cours "Pourquoi inventer des modèles pour la construction"),

A retenir

Une première différence fondamentale entre entité conventionnelle et entité structurée en *objet [ref. 4]* **porte sur le réseau de relations que l'éditeur d'un logiciel a su mettre en place autour des entités.**

L'intérêt de ce réseau de relations est pratiquement illimité, si l'éditeur dispose de moyens informatiques économiques pour l'exploiter :

- On peut **automatiser le comportement d'une entité vis à vis des entités de même classe**. Par exemple, chaque fois qu'un mur rencontre un autre mur, le calcul et le dessin de la jonction s'effectuent automatiquement.
- On peut **automatiser le comportement d'une entité avec toutes les classes d'entités différentes** qui sont autorisées à interagir sur elle : une menuiserie dans un mur provoque un percement. Un poteau rencontre un mur : que peut-il se passer ?
- On peut aller plus loin encore dans la finesse des comportements, en donnant la possibilité à l'utilisateur de **mettre en place des règles contextuelles**. Si telle configuration existe, alors faire ceci ...

  C'est quelquefois le cas lorsque l'on coche des options dans une boîte de dialogue. Par exemple, pour effectuer des modifications globales dans un projet, en une seule opération : remplacer des équipements, modifier des épaisseurs ...
- On peut également, et nous découvrons tout un nouveau secteur d'application, se servir de cet aspect relationnel pour **transformer une entité d'un état à l'autre**. Par exemple, transformer un volume de l'esquisse en murs et planchers.

On imagine le bénéfice qu'un utilisateur peut retirer de l'utilisation d'un logiciel exploitant ce réseau relationnel, en matière de rentabilité et de qualité.

ChapitreIV. Quand « l'objet » appartient à la fois au langage informatique et au langage du bâtiment ...

A. La puissance des langages à objets au service de l'utilisateur

Les recherches en génie logiciel permettent aujourd'hui d'exploiter des langages de programmation dédiés au type de problème évoqué : **mettre en relation** un grand nombre de *classe* **d'objets**, porteuses de **significations et de comportements différents**. Ce sont les langages, et les bases de données dites «*orienté objet*», largement utilisés dans les *NTIC*.

Sans entrer dans les détails, on peut dire que toute une série de nouveaux concepts de programmation permettent

- **de condenser le code, de le rendre modulaire,**
- **de le rendre général, c'est à dire de diminuer la partie que l'éditeur doit développer, en mettant à sa disposition des bibliothèques de fonctions,**
- **de structurer les données pour les hiérarchiser, pour les rendre facilement évolutives,**
- **de simplifier l'écriture des procédures, car c'est le langage qui se charge de gérer la combinatoire des possibilités, de se rendre compte si une action demandée par l'utilisateur est autorisée ou non.**

On serait tenté d'affirmer que ce sont des avantages au bénéfice des informaticiens qui ne concernent pas l'utilisateur. Erreur !

Ces performances de langage de programmation permettent de réduire les investissements tout en autorisant une puissance de traitement impossible à atteindre en programmation traditionnelle (algoritmique): le code serait volumineux, la durée de développement trop longue, avec pour conséquence des coûts de production élevés, et une fiabilité du logiciel aléatoire.

Le concept de description le plus fondamental des langages à objets est celui de « *classe* », c'est à dire la possibilité de décrire des *propriétés* et un comportement qui s'appliqueront ultérieurement à un ensemble d'entités réelles appelées « *occurrence* ».

L'avantage est de décrire des modèles abstraits, et de décliner un certain nombre d'exceptions avec le concept de « **sous classes** », en utilisant des fonctions de programmation

toutes faites d'une application à l'autre, car générales. D'où une économie d'investissement, alors que la structure de données décrite peut être complexe !

Un concept fondamental, de procédure cette fois, est celui d' « *Evénement* » qui déclenche une « *action* » lorsque certaines conditions se réalisent dans l'environnement d'une entité.

A chaque classe d'entités on peut associer ainsi, par avance, un ensemble d'actions possibles, chaque action pouvant à son tour modifier le contexte de l'environnement d'un autre objet, et ainsi de suite, pour provoquer des actions en cascade. Ce qui est important, c'est que ces actions s'exécutent automatiquement, sans la volonté explicite de celui qui les a provoquées.

Par exemple, l'opérateur ayant volontairement déplacé un mur dans le projet, le logiciel se charge alors tout seul d'allonger ou rétrécir les murs adjacents, et même dans certains cas, de modifier également les planchers, de déplacer certaines ouvertures ...

A retenir

Les bénéfices de la programmation « *objet* » concernent surtout l'utilisateur qui disposera d'une génération de logiciels toujours plus « intelligents », à moindre coût.

B. Les relations entre classes d'objets

Un autre concept décrit l'aspect structurant de l'information. Celui de « l'*Héritage* », qui par analogie, permet à certaines classes d'objets de bénéficier des propriétés et comportements d'objets «**parents** », ce qui simplifie encore l'écriture d'un logiciel, et réduit pour l'utilisateur, le nombre de commandes. L'héritage est une information de type « *Relations* ».

On peut définir dans une démarche de modélisation un grand nombre de types de relations.

Dans les logiciels de CAO bâtiment, on se limite à quelques relations simples, comme l'appartenance, ou la décomposition et son inverse la composition, comme le suivant ou le précédent (relation d'ordre), comme le contact (relation de calage) ...

On peut aussi vouloir spécifier des relations particulières à chaque acteur, à chaque rôle. Le danger est de compliquer à nouveau la programmation. Un juste milieu entre généralité et spécification descriptive des objets doit être trouvé. C'est le rôle du « *modèle conceptuel* ».

Le schéma non formel ci-dessous illustre quelques types de relations.

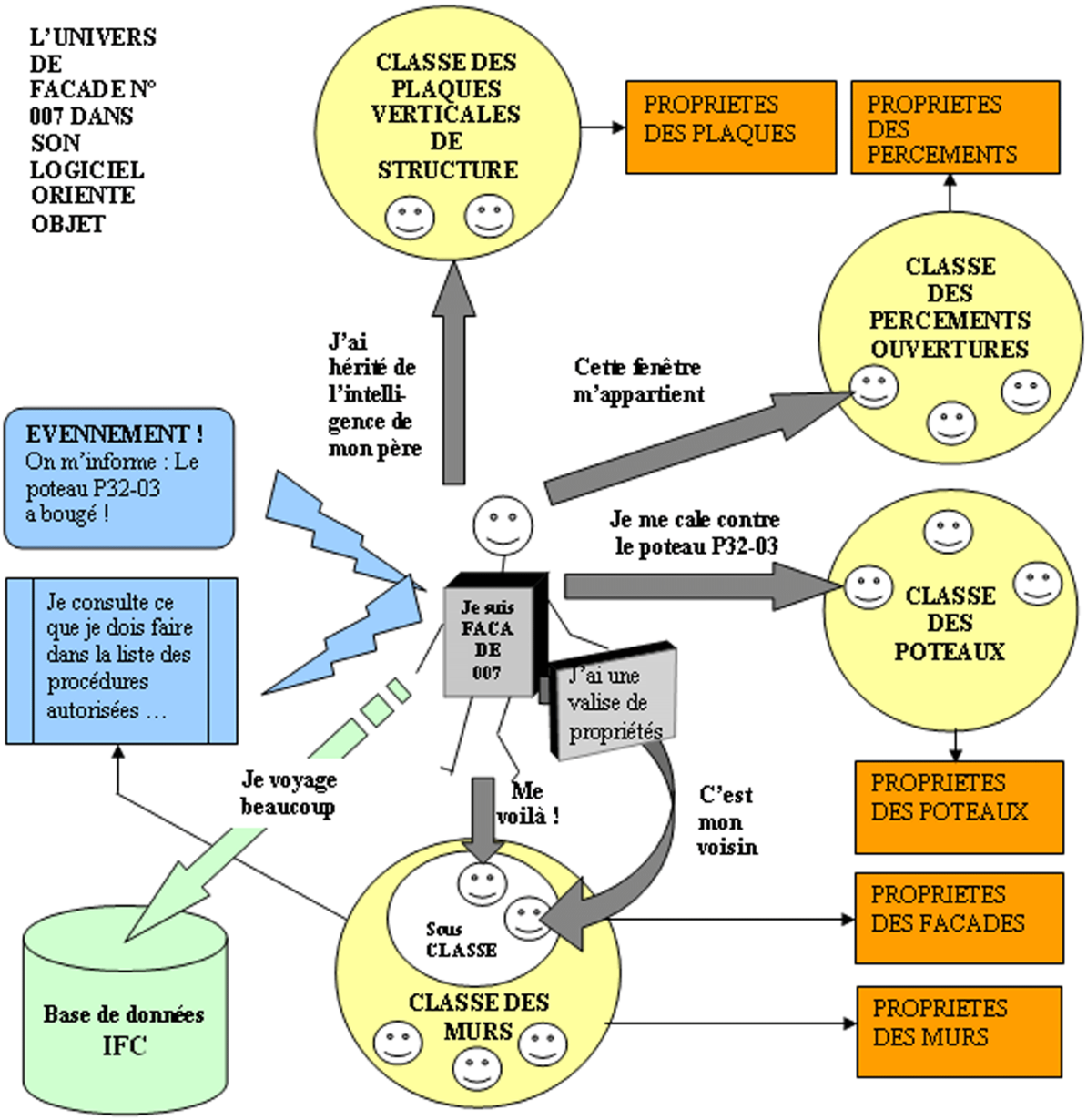


Schéma non formel illustrant quelques types de relations entre classes d'objets

ChapitreV. Le modèle conceptuel : une représentation des idées décrivant un phénomène.

A. Quand on utilise des mots, entre entendre et comprendre ...

Le terme «*modèle*» a déjà été utilisé à propos du « modèle IFC », du « modèle » graphique. Dans les logiciels graphiques, quand on parle de modèle, il s'agit le plus souvent du **modèle géométrique** du projet. C'est à dire sa maquette en 3D, support de calculs de vues cachées, de rendus plus ou moins réalistes, et quelquefois d'animations.

L'utilisateur des IFC devra s'intéresser aussi à une autre sorte de modèle : le *modèle conceptuel*. Il est utilisé aussi bien dans la mémoire d'un logiciel, que dans un fichier d'échange.

Les mêmes concepts de base sont exploités dans les deux cas. Seule l'utilisation est différente. Un concept dans un modèle n'est pas un discours .... dans une langue naturelle.

L'objectif principal d'un modèle conceptuel est de décrire, en levant toute ambiguïté sur la compréhension du phénomène décrit. Pour rendre clairs les concepts utilisés, les scientifiques ont inventé des langages artificiels avec leurs mots (les concepts et les idées), leurs règles. Peu importe si ce langage est pauvre, s'il atteint son objectif.

A retenir

Les *modèle conceptuel* formalisent les concepts les plus fondamentaux des logiciels ou d'un standard d'échange : la signification des données et leur organisation. Le modèle conceptuel permet de construire un fichier ou une base de données.

Les informations contenues dans un modèle sont plus ou moins accessibles à l'utilisateur :

- Le modèle conceptuel utilisé par un logiciel pour représenter en interne le projet a été conçu par l'éditeur du logiciel. C'est son affaire, et souvent son secret.

  L'utilisateur ne pourra pas en général y avoir accès. Il peut tout au plus en deviner

les grandes lignes à travers les performances du logiciel. Mais une chose est sûre : les objets, les concepts manipulés par le logiciel doivent correspondre à ceux que l'utilisateur imagine dans sa tête, pour l'usage qu'il veut en faire. L'utilisateur, sans qu'il en soit conscient, possède un modèle intuitif des concepts qui lui sont familiers. Si les deux modèles divergent trop, l'utilisateur ne pourra pas s'adapter au logiciel. Le rejet est une conséquence d'une incompréhension sur les définitions de chacun. Ne pas définir clairement ses concepts interdit toute communication et aboutit toujours à des catastrophes (dans tous les domaines !).

- Le modèle conceptuel représentant les données d'un projet à échanger est par définition public. Chaque intervenant obligé d'écrire ou de lire des informations du projet doit savoir les interpréter. Soit pour en connaître leur signification, en vue d'une exploitation manuelle, soit pour une exploitation informatisée dans un logiciel technique. Dans ce dernier cas, il faut que les données importées correspondent exactement aux définitions sémantiques des informations à saisir dans le logiciel cible. C'est la condition préalable pour que deux logiciels puissent se comprendre : **s'entendre sur des définitions communes.**

B. Un exemple de modèle conceptuel d'objets du bâtiment

Pour faciliter la démarche pédagogique, nous illustrerons par quelques schémas une petite partie d'un modèle qui décrit une des familles d'objets du bâtiment : les composants.

Dans la terminologie IFC, les composants sont des *produits [ref. 6]* .

Il est important de situer ce concept par rapport aux autres concepts de même niveau manipulés dans les IFC.

Les schémas montrés dans cet ouvrage adoptent eux mêmes des conventions graphiques structurées, normalisées. En attendant d'apprendre ce formalisme, souvent utilisé pour exprimer les schémas conceptuels, nous les expliquons (pour information on utilise le formalisme *NIAM*, qui est développé dans l'unité de cours suivante).

Un modèle conceptuel comprend trois sortes de concepts, déjà examinés par ailleurs :

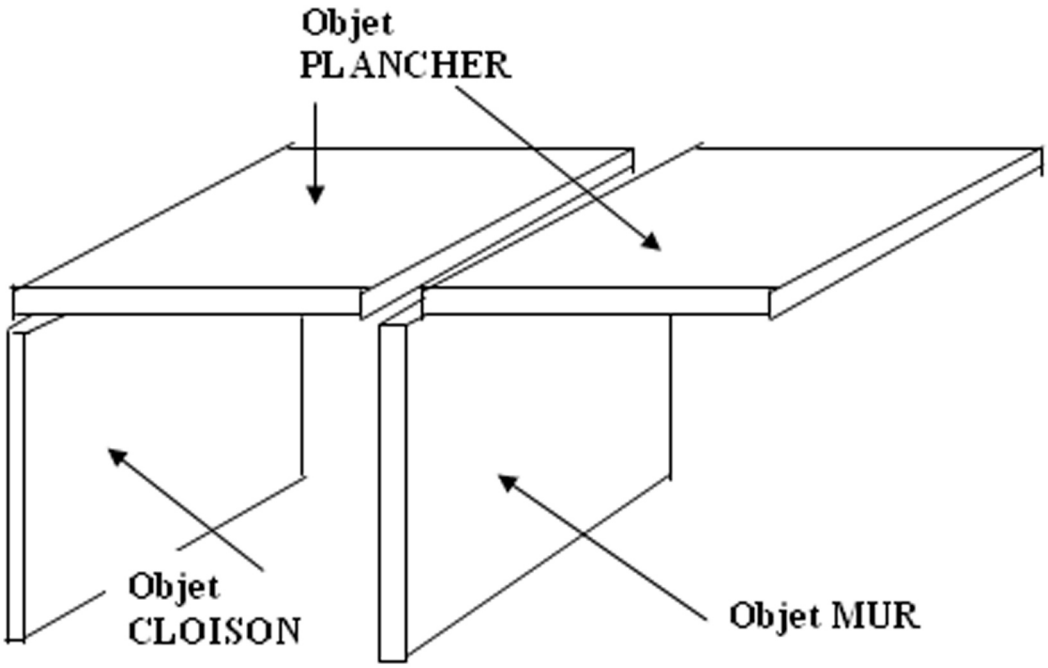
- *objet*, compris dans le sens de *classe*, que l'on appelle parfois aussi tout simplement un *concept*. Exemple : mur est un concept. Pour ne pas provoquer d'ambiguïté de vocabulaire, il faudrait toujours dire « **classe d'objets** », le terme « objet » étant réservé à un individu de la classe (une occurrence).
- les *Relations*, qui relient les objets. Un schéma qui décrit un modèle doit être simple. C'est pourquoi en général on préfère n'utiliser des relations que limitées à deux objets. On appelle ces relations « binaires ».

  Exemple : la relation de calage « repose sur » lie les objets « plancher » et « mur ».

  On s'aperçoit qu'une relation à un sens direct : « repose sur », et un inverse : « est supporté par ».
- les *attributs*, ou *propriétés* qui portent sur les objets, et aussi sur les relations. Exemple : un type de mur est caractérisé par une série de types de propriétés (épaisseur, dimensions, matériaux, couleur ...). De plus, une occurrence de mur, c'est à dire un morceau de mur réel, inséré dans un projet, devient un individu de la classe mur, auquel on peut associer un numéro d'identification (unique), et pour

lequel les propriétés prennent une valeur.

Une partie d'un modèle qui regroupe deux objets, une relation et des propriétés s'appelle une « *idée* ». Une idée est une description élémentaire de telle sorte que pour l'usage que l'on doit en faire ne subsiste aucune ambiguïté. Chaque personne qui essaie de comprendre une idée doit en tirer la même signification.

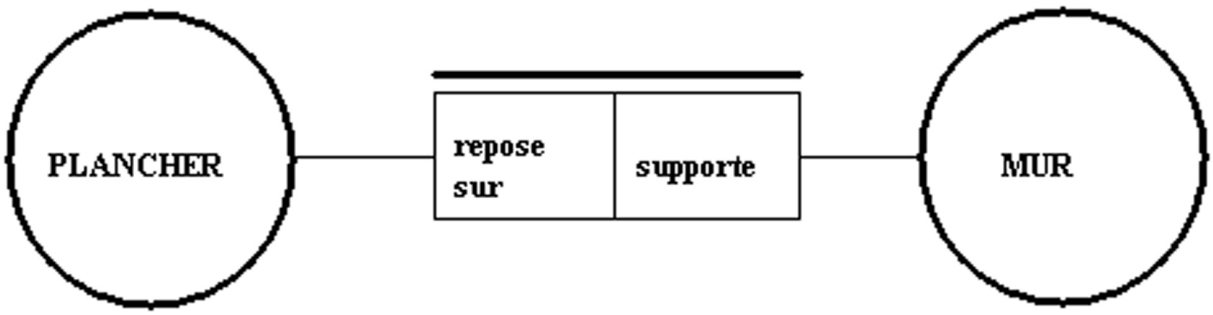


Exprimer l'idée qu'un plancher repose sur des murs sans qu'il y ait d'ambiguïté n'est pas si simple.

Tout d'abord il faut définir un plancher type par ses constituants : faces inférieure, supérieure, ses rives, l'existence d'un ou plusieurs sens de portée ...

Ensuite il faut définir de la même façon un mur type, définir une sous classe de mur qui est la cloison (laquelle n'est pas porteuse, c'est à dire admet des propriétés différentes).

Enfin, on peut exprimer l'idée qu'un plancher peut porter sur des murs, selon un contexte de voisinage également à définir ...

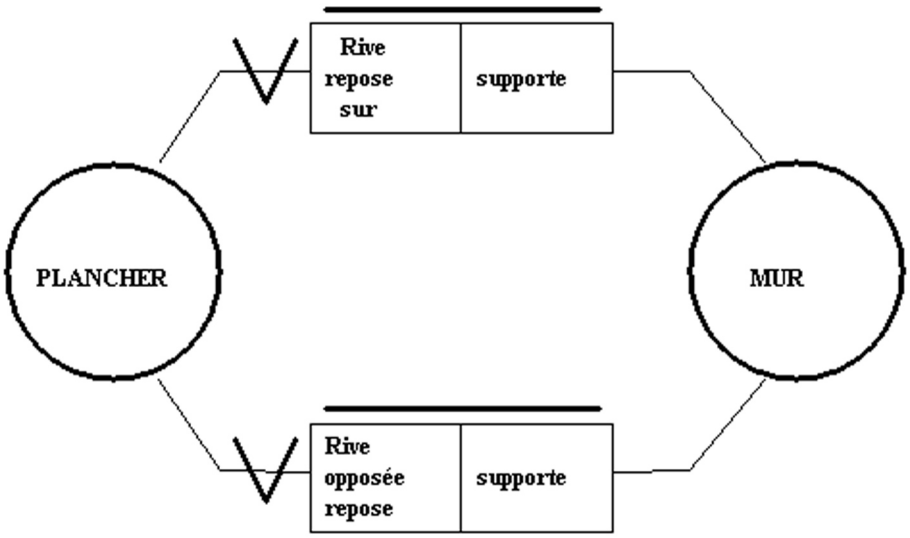


Le schéma ci-dessus exprime l'idée à approfondir ou à laisser en l'état, pour indiquer à un logiciel de calcul de descente de charge comment s'y prendre. Traduction :

Un plancher peut reposer sur aucun mur (parce que la solution poutre existe) ou sur un seul mur (parce que le plancher peut être en porte à faux) ou sur plusieurs murs, tandis qu'en lecture inverse, un mur peut supporter un ou plusieurs planchers ou aucun.

C. La réalité est souvent plus complexe

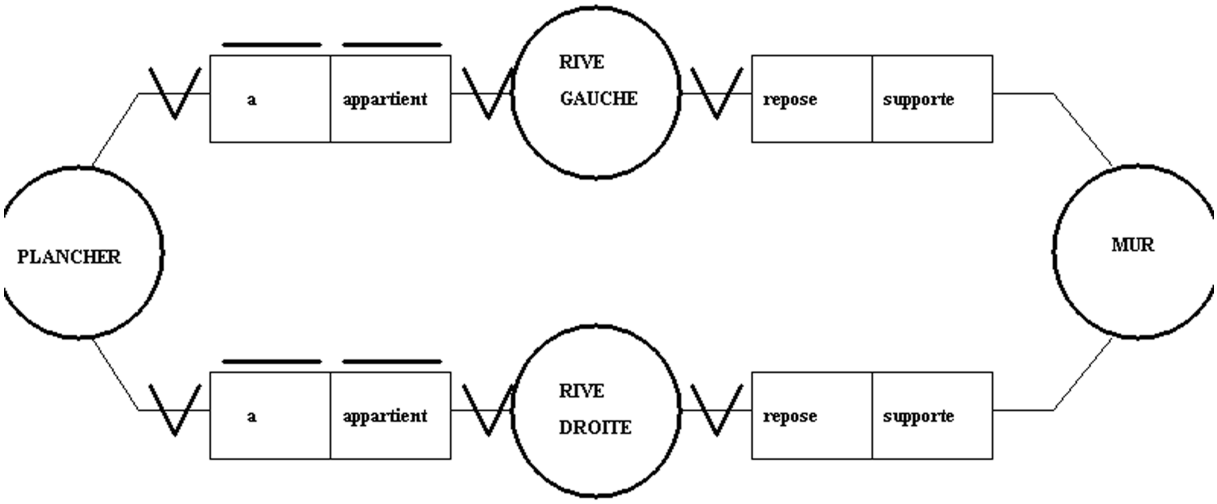
Par exemple, si on veut exprimer que le procédé de construction restreint les possibilités à des planchers unidirectionnels à quatre rives, le schéma devient :



Le schéma précédent introduit dans le modèle deux relations compliquées :

- repose par l'intermédiaire d'une rive ...
- repose par l'intermédiaire de la rive opposée...

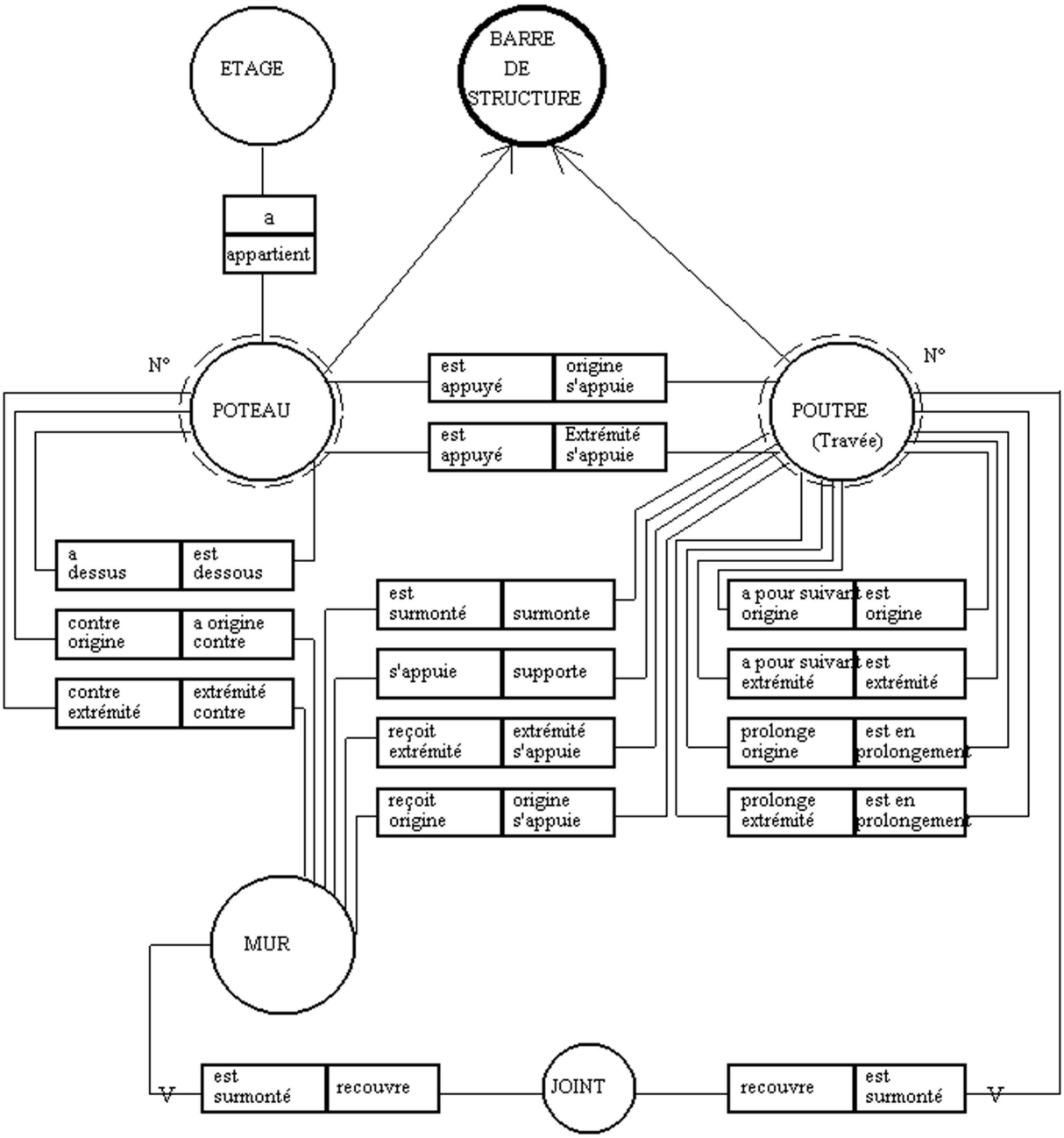
Si le concept de « rive » est utilisé plusieurs fois dans le logiciel et/ou que ce concept risque d'être manipulé dans d'autres logiciels techniques, alors il est préférable d'attribuer à la rive un statut d'objet indépendant, qui devient un constituant du plancher :



Tout plancher possède obligatoirement à la fois deux rives porteuses opposées, que l'on distingue comme des types d'objets séparés. Chaque rive repose obligatoirement sur un ou plusieurs murs (il n'y a pas découpage d'un seul mur sous chaque plancher).

Ce dernier schéma exprime quatre *idée*.

On peut bien sûr exprimer des schémas plus complexes pour rendre compte des multiples relations liant des composants de structure entre-eux, mais aussi avec des familles d'objets différentes (ce schéma n'appartient pas aux IFC, mais pourrait le devenir) :

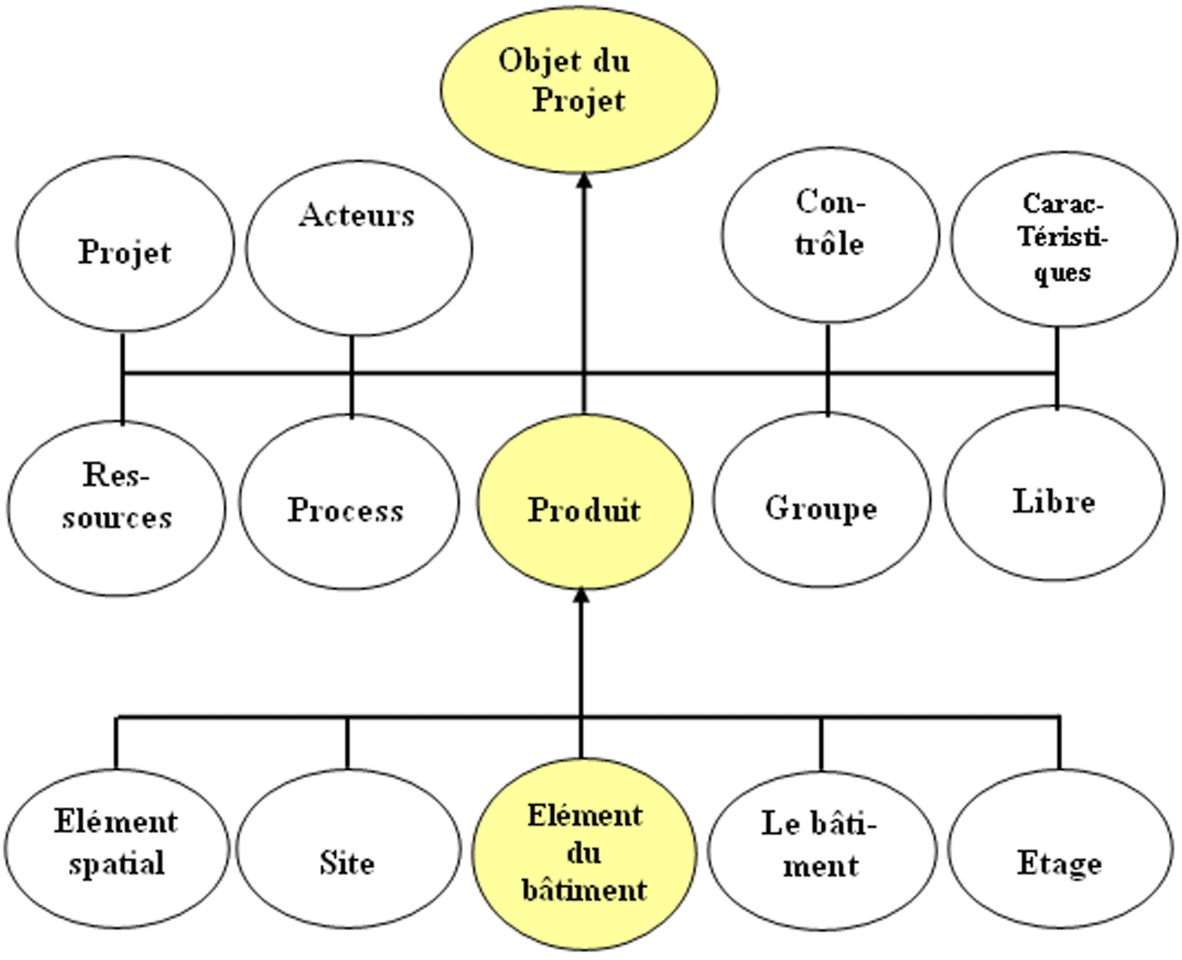


Un modèle conceptuel n'est autre que l'ensemble des *idée* qui décrivent d'une façon abstraite un phénomène, un mécanisme, un dispositif, une collection d'objets matériels ou abstraits comme une comptabilité, des stocks de matériaux, de denrées, une population d'individus, un service, ... et bien évidemment un bâtiment.

ChapitreVI. Arbre d'héritage des objets “produit” des IFC.

A. Arbre d'héritage principal

Pour l'instant, nous ne nous intéresserons qu'à un sous-ensemble des informations qui gravitent autour du concept d'objet : La *classe*, et une seule relation, la plus structurante : l'*Héritage*.



Une flèche formalise la relation d'héritage, qui se lit dans le sens de la flèche « est une sorte de ». Un objet de la classe « a pour parent ». Dans le sens inverse l'héritage se lit« se spécialise en ». Un objet « a pour descendant ». Ce qui revient à définir pour une classe

donnée, son sur-type et ses sous-types.

Noter qu'une classe n'a toujours qu'un sur-type (qu'un seul papa, et pas de maman).

C'est triste, mais plus facile à gérer, car l'héritage devient une arborescence !

Ne pas confondre cette famille de relation avec la composition et son inverse l'appartenance.

Un produit **n'appartient pas** à un objet du projet, mais **est** une sorte d'objet du projet.

Pourquoi être si tatillon ? Si vous saisissez bien cette différence, plus aucun schéma conceptuel n'aura de secret pour vous.

A quoi cela vous sert de pouvoir les lire ? Je ne répondrai plus !

B. Spécialisation de la classe « Produit »

Les 14 grandes familles d'entités IFC du précédent schéma sont donc des « Objets » au sens des structures et langages informatiques, mais sont aussi des objets du projet, au sens de nos métiers de la Construction.

Le noyau des IFC comprend 9 entités, dont celle qui nous intéresse pour l'instant : le « *produits [ref. 6]* ». Ce qui la distingue des autres, c'est que le produit est considéré comme une entité physique, qui participe directement au bâti (au contraire des documents, process, informations sur le projet ... et autres concepts abstraits).

Le Produit se spécialise à son tour en 5 classes d'entités qui méritent un commentaire pour la signification donnée par les IFC.

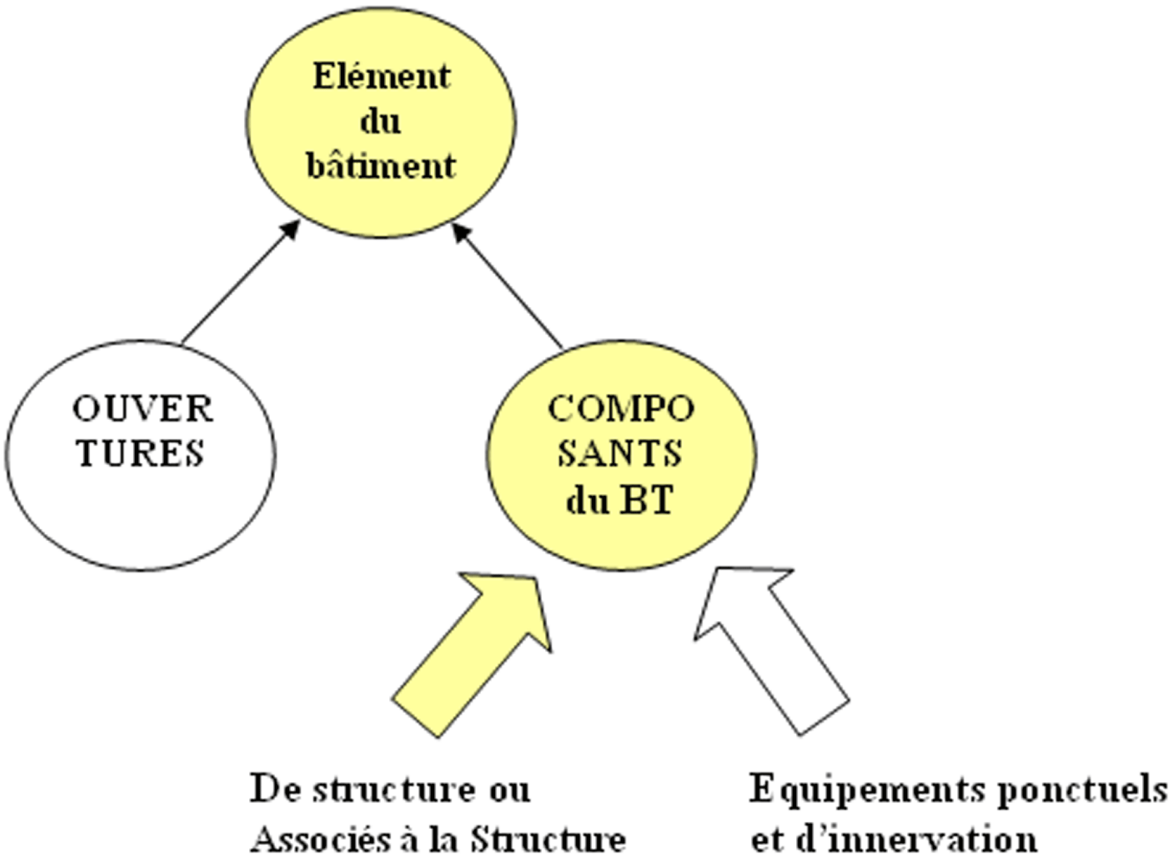
Les « *Eléments spatiaux [ref. 2]* » sont les locaux du projet. Ce concept revêt une grande importance dans les pratiques anglo-saxones de la Construction. Il devient important en Europe, pour les responsables de la gestion de locaux et de la *GTP*, car c'est le concept principal exploité durant toute la vie du Bâtiment. L'utilisateur devra s'attacher à renseigner convenablement les entités de ce sous-modèle.

« *Le bâtiment [ref. 1]* » est compris ici dans sa globalité. Peu d'informations y sont associées. De la même façon, « *l'Etage [ref. 3]* » regroupe peu d'informations, mais la liste de tous les étages est fréquemment consultée.

Le « *Site [ref. 7]* » regroupe toutes les informations du terrain, des aménagements extérieurs, des *VRD*.

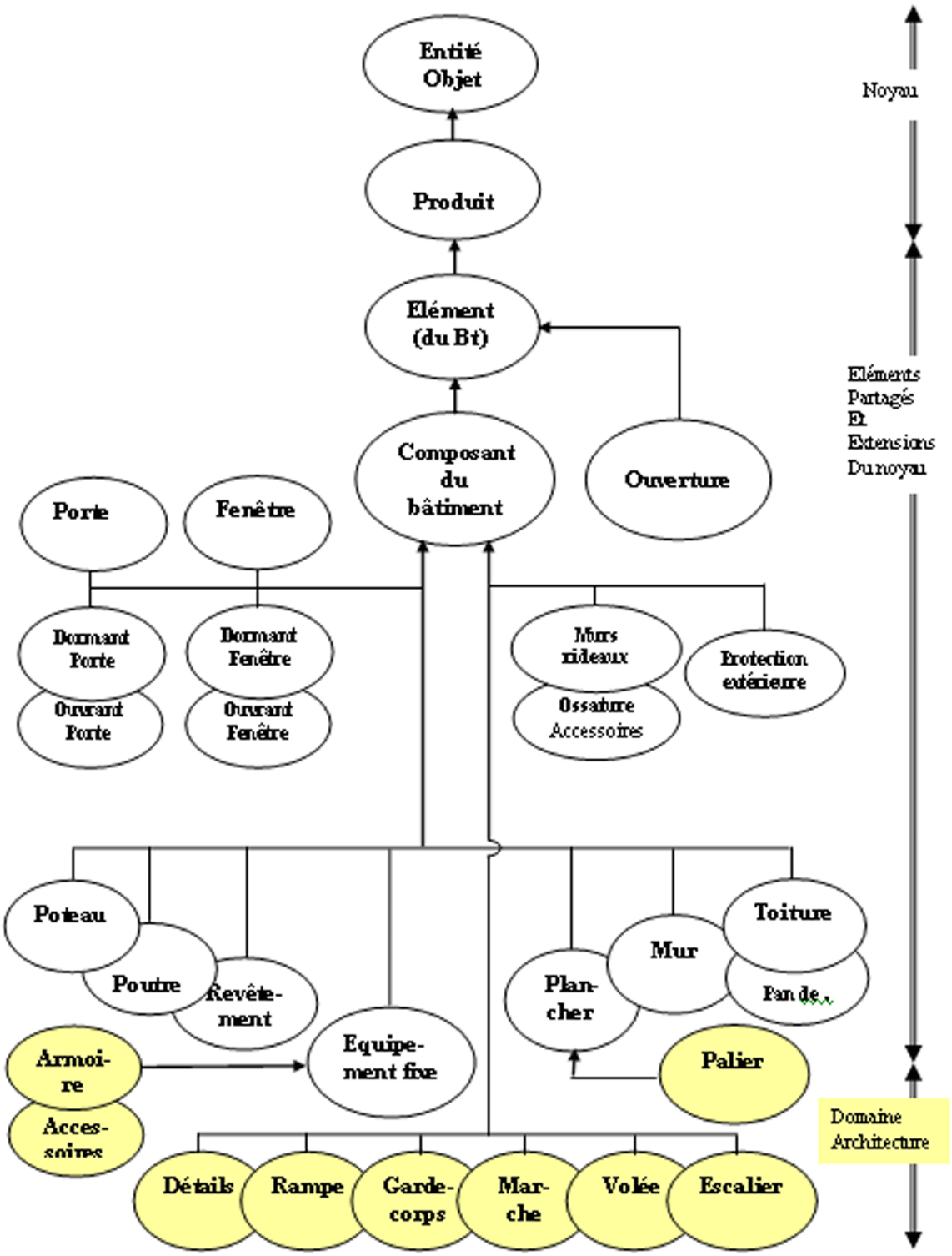
Enfin, « l'Elément du bâtiment » concerne le reste, c'est à dire l'essentiel de ce qui est contenu dans un bâtiment.

Ce concept intermédiaire du niveau « **interopérable** » du modèle IFC donne naissance avec ses extensions à l'arborescence de classes d'entités la plus riche des IFC. De structure ou Equipements ponctuels Associés à la Structure et d'innervation



Curieusement, les IFC n'ont pas créé de concepts intermédiaires pour séparer les Equipements des Composants de structure ou associés, ceux qui nous intéressent.

L'arbre d'héritage complet en suivant la flèche de gauche devient :



Selon les définitions des IFC, les composants d'escalier font partie du domaine de l'architecture. Ils ne sont pas partagés avec les autres domaines.

L'utilisateur retiendra de cette arborescence d'entités IFC –une parmi la dizaine qui est actuellement décrite- deux conclusions :

- Pour une famille donnée d'entités –ici les composants de structure et associés- il

dispose d'une **liste précise de classes normalisées**. Selon son rôle, l'utilisateur est concerné par la totalité ou certaines d'entre elles. Pour un architecte, il est clair qu'il devra décrire la totalité de ces classes de composants, s'ils existent dans son projet. Son logiciel le lui permettra-t-il ?

- Chaque classe de composant est décrite également par un **ensemble d'informations normalisées**. En supposant que le logiciel utilisé soit complètement conforme à la norme IFC, cette description pour celui qui transmet, ou son décodage, pour celui qui reçoit, constitue la tâche essentielle de l'utilisateur.

C'est l'objet de la deuxième partie de l'ouvrage d'en évoquer plus concrètement les méthodes.

A retenir

Un utilisateur peut limiter sa connaissance des IFC à deux concepts :

- **les classes d'objets représentant les *ouvrages* et *composants* de son domaine d'intervention,**
- **les classes de *propriétés* associées.**

Il appartient à l'utilisateur de renseigner ou d'interpréter ces informations normalisées pour chaque individu (occurrence) présent dans son projet.
Ce sont les logiciels applicatifs qui se chargent du reste. Du moins, on l'espère.

Dictionnaire

action :

Voir l'unité "Quelques exemples d'élaboration de modèles conceptuels".

Dans un modèle dynamique, action ponctuelle qui peut être exécutée à l'entrée, à la sortie ou pendant une transition

attributs :

Voir l'unité "Le concept d'objets dans les logiciels de DAO, CAO et de calcul".

Même sens que "Propriété", c'est à dire description associée uniquement à un objet.

classe :

Voir l'unité "L'IA et les IFC : introduction".

Dans un LOO, et dans les IFC, une classe regroupe des objets de même type, possédant des propriétés et un comportement semblable.

composants :

Voir l'unité "L'IAI et les IFC : introduction".

Terme générique pour désigner un objet physique du bâtiment. Il est assemblé ou préfabriqué. S'il est construit sur place, il se dénomme ouvrage.

concept :

Voir l'unité "Le concept d'objets dans les logiciels de DAO, CAO et de calcul".

Dans un modèle conceptuel, c'est une classe (unique) d'objets pertinente pour décrire le système d'information et son organisation sémantique.

couches :

Voir l'unité "Pour inventer des modèles pour la Construction".

Voir "Structuration en couche".

- Structuration en couches :
  Voir l'unité "Pourquoi inventer des modèles pour la Construction ?".
  L'information, surtout graphique, est dessinée dans une série de "calques" indépendants. Problèmes : standardiser la structure des calques.

Evénement :

Voir l'unité "Quelques exemples d'élaboration de modèles conceptuels".

Action brusque qui n'a pas de durée, déclenchée par toute cause intérieure ou extérieure au système d'information.

GTP :

Gestion Technique de Patrimoine (immobilier)

Héritage :

Voir l'unité "Le concept d'objet dans les logiciels de DAO, CAO et de calcul".

Dans un LOO, et dans les IFC, mécanisme qui permet à des objets d'une sous-classe de bénéficier des propriétés des classes "parents"

idée :

Voir l'unité "L'IAI et les IFC : introduction".

Terme utilisé dans la méthode de spécification NIAM : information mettant en jeu une seule relation (ensembliste) entre deux concepts.

ISO :

"International Organization for standardization". Organisation internationale de normalisation, qui spécifie en particulier les normes de communication en informatique. L'ISO dépend de l'ONU.

modèle :

Voir l'unité ''Le concept d'objet dans les logiciels de DAO, CAO et de calcul".

Sens restrictif : Description d'un système d'information à l'aide de méthodes ou de langages de spécification formelle pour une vue donnée.

modèle conceptuel :

Ce terme est développé dans l'unité "Le concept d'objet dans les logiciels de DAO, CAO et de calcul".

Description formelle des concepts véhiculés dans un modèle de base de données, focalisée sur l'aspect sémantique (statique) du système d'information.

NIAM :

"Nijssen Information Analysis Method". Méthode de spécification formelle de données, utilisable dans n'importe quel domaine pour décrire sans ambiguïté une organisation de concepts, d'objets, y compris les relations et attributs associés. Cette méthode est normalisée (STEP et ISO en 1983)

NTIC :

Nouvelles Technologies de l'Information et de la Communication.

Regroupent les EDI, les réseaux de communication, Internet et les sites WEB, les réseaux locaux, les protocoles d'échange, le multimédia, les matériels et logiciels nécessaires, ...On dit aussi TIC

objet :

Voir l'unité "Le concept d'objet dans les logiciels de DAO, CAO et de calcul".

Nomme indifféremment un type d'objet, ou une occurrence de la classe.

Voir "Orienté Objet" et "Occurence (ou instance)".

occurrence :

Voir l'unité "Le concept d'objet dans les logiciels de DAO, CAO et de calcul".

Dans une classe, objet individualisé de l'ensemble par un nom ou numéro. Chaque objet vérifie les propriétés et comportement de sa classe.

orienté objet :

Voir l'unité "Pourquoi inventer des modèles pour la Construction ?".

Se dit d'un langage ou d'un logiciel utilisant une structuration d'informations exploitant les concepts de classes, attributs, événements, relations ...

ouvrages :

Voir l'unité "l'IA et les IFC : introduction".

Par opposition à composant, désigne un élément du bâtiment construit sur place, faisant appel à de la main d'oeuvre et des matériaux

propriétés :

Voir l'unité "L'IAI et les IFC : introduction".

Dans un LOO, et dans les IFC, une propriété qualifie un objet d'une classe : propriété propre, ou propriété contextuelle.

Relations :

Voir l'unité "L'IA et les IFC : introduction".

Dans un LOO, et dans les IFC, une relation est un lien formel entre deux objets de même classe ou de classe différentes, ou avec une autre relation.

STEP :

"Standard for Exchange of Product Data Model". Norme ISO de spécification et d'échange de modèles de produits. STEP propose un cadre méthodologique, un formalisme et des outils EDI.

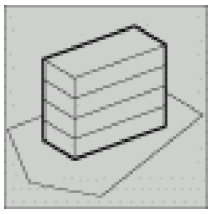
VRD :

Voirie Réseaux Divers

Concepts IFC

Référence 1

Bâtiment

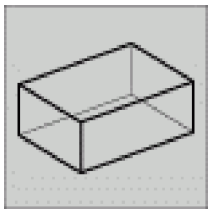


Entité construite indépendante structurellement ou selon l'organisation du projet.

ATTRIBUTS peu nombreux : nom court et long, emprise au sol, volume, hauteur totale, altitude absolue (/mer) altitudes des emprises au sol, services connectés ; Peut appartenir à un groupe de bâtiments.

Référence 2

Elément spatial

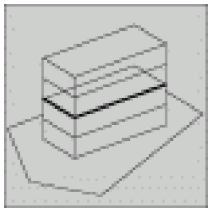


Concept abstrait pour définir un élément spatial composé soit d'un espace mesurable (possédant une surface et un volume, c'est à dire un "local"), soit des limites (frontières) de l'espace (c'est à dire la face visible des murs, planchers et sous faces de plafonds du local, ou "nu de local"). Ces deux types d'éléments doivent obligatoirement admettre une représentation géométrique.

Pas d'ATTRIBUTS à ce niveau.

Référence 3

Etage (ou niveau)



Partie du "bâtiment" associée à un plan horizontal (même si l'étage comporte des duplex).

ATTRIBUTS : admet une liste similaire de propriétés en plus de celles du "Produit" :

noms courts et longs, altitude de référence, hauteur, surface, volume, description géométrique. Peut éventuellement être associé à une liste de "local".

Référence 4

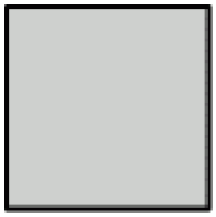
Objet (du projet)

Classe abstraite qui définit tout objet physique, virtuel ou concept abstrait relatif au projet. Déjà ce stade général, un certain nombre d'ATTRIBUTS peuvent être consignés qui qualifient tous les objets de l'arbre d'héritage, mais qui sont l'affaire des développeurs de logiciels.

L'utilisateur doit renseigner un seul ATTRIBUT : la liste des documents externes attachés au projet (par exemple les documents contractuels).

Référence 5

Objet utilisateur

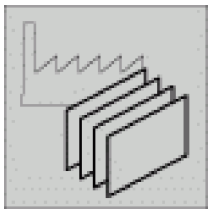


Les IFC ne prétendant pas normaliser la totalité des types d'objets que l'on peut rencontrer dans un projet. Cette classe est destinée à recevoir la description de "procédures" ou de "produits" définis par l'utilisateur, comportant ou non un dessin, porteur ou non d'une sémantique explicite.

ATTRIBUTS disponibles : définition (sémantique) de l'objet utilisateur. Si l'objet est un produit : localisation de l'objet et représentation géométrique et/ou topologique.

Référence 6

Produit

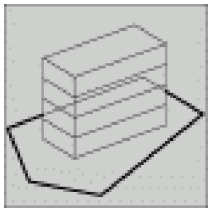


Classe abstraite pour représenter tout produit manufacturé, découpé ou fabriqué in situ, ou encore résultat de frontières matérielles comme les espaces ("local").

ATTRIBUTS : localisation, représentation géométrique ou forme associée, liste de spécifications

Référence 7

Site



Terrain défini (parcelles cadastrales par exemple) sur lequel le "bâtiment(s)" est construit. Peut inclure une représentation géométrique du terrain (courbes de niveaux) et des équipements extérieurs (parking, VRD, végétation, ...) Le site peut être le résultat d'un ensemble de sites.

ATTRIBUTS : point de référence défini en latitude, longitude et altitude. Définition géométrique du périmètre du site. Surface.